AFRL-RI-RS-TR-2020-135

A PICTURE IS WORTH A BILLION BITS: ADAPTIVE VISUALIZATION OF BIG DATA

UNIVERSITY OF WASHINGTON

JULY 2020

FINAL TECHNICAL REPORT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

STINFO COPY

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE

■ **AIR FORCE MATERIEL COMMAND** ■ **UNITED STATES AIR FORCE** ■ **ROME, NY 13441**

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09. This report is available to the general public, including foreign nations. Copies may be obtained from the Defense Technical Information Center (DTIC) (http://www.dtic.mil).

AFRL-RI-RS-TR-2020-135 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE CHIEF ENGINEER:

/ S /
STEVEN DRAGER
Work Unit Manager

/ S /
GREGORY HADYNSKI
Assistant Technical Advisor
Computing & Communications Division
Information Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
JULY 2020	FINAL TECHNICAL REPORT	OCT 2015 – DEC 2019

4. TITLE AND SUBTITLE

A PICTURE IS WORTH A BILLION BITS: ADAPTIVE VISUALIZATION OF BIG DATA

5a. CONTRACT NUMBER
FA8750-16-2-0032

5b. GRANT NUMBER
N/A

5c. PROGRAM ELEMENT NUMBER
61101E

6. AUTHOR(S)

Rastislav Bodik,Luis Ceze, Alvin Cheung, Michael Ernst,
Dan Grossman, Zachary Tatlock, Emina Torlak, and Xi Wang

5d. PROJECT NUMBER
BRAS

5e. TASK NUMBER
SU

5f. WORK UNIT NUMBER
WA

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

University Of Washington
Office Of Sponsored Programs
4333 Brooklyn Ave Ne
Seattle WA 98195-0001

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Air Force Research Laboratory/RITA
525 Brooks Road
Rome NY 13441-4505

DARPA
675 North Randolph St
Arlington VA 22203-2114

10. SPONSOR/MONITOR'S ACRONYM(S)
AFRL/RI

11. SPONSOR/MONITOR'S REPORT NUMBER
AFRL-RI-RS-TR-2020-135

12. DISTRIBUTION AVAILABILITY STATEMENT
Approved for Public Release; Distribution Unlimited. This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This report documents the results achieved by the University of Washington SandCat team under the Defense Advanced Research Projects Agency Building Resource Adaptive Software Systems program. This effort focused on developing static and dynamic adaptations of computer systems, as a modern system stack is forced to (1) accommodate the growth in data volume and data bandwidth at rates exceeding Moore's Law; and (2) support energy-efficient mobile devices that can neither store all necessary data nor perform the computation at the desired real-time latency. To meet these two goals, SandCat technology automates adaptation by aggressively approximating both the data and the computation, as well as by performing the computations incrementally. In particular, the Sandcat project responded to two needs for adaptation: (1) the rapid change in hardware, systems software, and application workloads; and (2) the increased complexity of computer systems, especially at the interfaces, which were intended to hide the complexity. A major milestone in the research area of program synthesis was achieved, as the synthesis technology developed by SandCat reached parity with human programmers on about a dozen different programming tasks. Several components of SandCat technology have been transitioned into the commercial marketplace, including verified lifting which ships in a commercial photo editing product; the Rosette architecture which is being used in the Synthetic Minds startup; and the machine learning kernels which are being commercialized by OctoML.

15. SUBJECT TERMS
Adaptation; system verification; program synthesis

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			**STEVEN DRAGER**
U	U	U	UU	30	19b. TELEPHONE NUMBER (*Include area code*) **NA**

TABLE OF CONTENTS

1. Summary

The University of Washington (UW) SandCat project, an effort within the Defense Advanced Research Projects Agency (DARPA) Building Resource Adaptive Software Systems (BRASS) program, responded to (1) the rapid change in hardware, systems software, and application workloads; and (2) the trend towards increased complexity of computer systems, especially at the interfaces which were intended to hide the complexity. The rapid change requires adapting existing applications and infrastructure but the complexity makes the adaptation more difficult. The SandCat project developed methods that analyze existing components and synthesize new ones. The SandCat adaptations were semantic and mostly performed offline. The goal was to aid programmers and system designers who currently adapt code by manually rewriting the code. The project selected problems from across the entire systems stack, from the central processing unit (CPU) to compute kernels and data structures, and from disk devices to file systems, and databases.

The synthesis technology developed by SandCat reached parity with human programmers on about a dozen different programming tasks, which is a major milestone in the research area of program synthesis. Some problems solved by SandCat have been automatically solved for the first time. The tools developed by the team enabled new adaptations of software to hardware, including the synthesis of:

1. surprising optimizations for floating-point expressions (Herbie) [1, 2, 3, 4, 5],

2. new mappings of computations onto graphics processing units (GPUs) not achievable by existing compilers (Swizzle Inventor) [6, 7], and

3. ultra-low-precision machine learning (ML) kernels (TVM) [8, 9].

The technology also synthesized artifacts that semantically summarize existing implementations, enabling:

1. performance portability to new platforms and hardware (verified lifting) [10, 11, 12, 13], and

2. automatic authoring of memory models and file system crash consistency models (memSynth and Ferrite) [14, 15, 16]

These advances were enabled by new infrastructure, including:

- the Cozy synthesizer of data structures [17, 18],
- the Rosette generator of synthesizers [19, 20, 21, 22], and
- the STAle Configuration and Consistency Analysis Tool (Staccato) automatic checker of adaptation mechanisms [23, 24, 25].

The technology has transitioned to practice, as the results of verified lifting ship in a commercial photo editing product (Adobe Photo-shop); the Rosette architecture is being used in the Synthetic Minds startup; and the ML kernels are being commercialized by OctoML.

2. Introduction

Problem Description: Big-data analytics is overwhelmed by the volume of data to process and store. On the compute side, today's analytics is already slow and power-hungry, executed in a batch job on a data center, rather than in real-time on a laptop or a sensor device. On the storage side, the data volume forces us to discard old logs and give up precision with data compression. If data creation trends continue, these challenges will only become exacerbated.

The University of Washington SandCat project investigates how analytics can keep up with growing datasets and more sophisticated analytics algorithms, and how it will deliver interactive data analysis.

The premise of our research plan was that improvements in computation and storage will come with increasingly unbalanced tradeoffs: processors will increase parallelism at the cost of more complex shared-memory models spanning CPU, GPU, and field programmable gate array (FPGA) integrated on the same chip; file systems will increase their throughput by sacrificing data consistency after system crashes; and storage devices may increase their capacity at the cost of increasing the error rate. Across the entire system stack, we will see the proliferation of software specialized to an application domain and a class of hardware, forcing us to migrate applications and reformulate algorithms to new programming models.

Research Goals: We have designed components of a computer stack that adapt to the adversity that arises when data volume impairs analysis latency or exceeds disk capacity. For example, we specialize the system execution to the quality of service (QoS) specification, by approximating the computation and the stored data [26].

We also adapt to the opportunity provided by new technologies that sacrifice correctness or introduce complexity to increase performance, by restoring the correctness of affected system components and by developing programmer tools that help overcome complex system interfaces, such as memory models.

We explore these adaptations on a system stack comprising data visualization, data analytics, image store, a file system, storage devices, and data servers distributed across a network. We are especially interested in an adaptation that spans multiple layers of the stack, such as adapting the image store to data analytics algorithms by tuning on the fly the quality of retrieved images.

Impact: This project developed technologies for specializing system components and adapting them to similar changes in other components. The adaptations restored component correctness, took advantage of new resources, and traded quality for efficiency. Beyond specialization and adaptation, we delivered methods and tools for designing, implementing, and verifying systems for the era of growing system complexity. Our tools produce not only programs but other artifacts required for system design, such as semantic models and specifications of component interfaces.

3. Methods, Assumptions, and Procedures

Our project has primarily relied on two classes of techniques — program synthesis and refinement. During the four years of UW SandCat under the DARPA BRASS program, we have developed them into surprisingly versatile technologies.

3.1 Program Synthesis

A program synthesizer produces a program that satisfies a given specification. The specification can be a reference implementation; a validation test suite; a designer-supplied example execution; a desired safety property; or a multi-modal combination of these artifacts. The synthesized program will be found by searching the programs in a Domain-Specific Language (DSL) designed by the synthesizer author with the help of a domain expert.

We have synthesized both high-performance implementations of system components [27] and specifications of system artifacts, such as legacy analytics programs and system interfaces [12]. Synthesis of specifications is obtained for free by viewing specifications as programs from a specially designed DSL. This enables inference of specifications from execution traces and other readily obtained artifacts. The synthesized specification can in turn be used to synthesize implementations or other specifications. For example, we have automatically obtained a specification of a shared-memory model from litmus tests, which are small multithreaded programs describing the behavior of the memory model; this memory model can be used to synthesize a fence-based synchronization of a multithreaded program running on hardware with this model [14].

Compared to a compiler, a synthesizer produces the program by searching the space of DSL programs, looking for one that satisfies the specification. This search can be performed by reduction to satisfiability modulo theories (SMT) constraint solving, for example. In contrast, a compiler uses rewrite rules to mechanically lower the source program to its implementation. Whenever a compiler can be used for a given domain, it is preferred over synthesis, which is usually slower than compilation. However, (1) synthesis requires no rewrite rules, just the DSL; and (2) synthesis specifications need not be full source-code programs — synthesis can generate code from tests, traces, and other partial-specifications that may be easier to obtain than a full executable specification. This is especially beneficial for synthesis of semantic specifications such as models of shared memory, crash models of file systems, and interfaces for other components.

3.2 Refinement

Refinement is a methodology for proving program correctness gradually rather than in a single monolithic proof. The monolithic proof is decomposed into several simpler proofs, which act as refinement steps in lowering the specification (i.e., an abstract version of the program) into an implementation (i.e., a fully concrete program). A well-designed refinement methodology provides abstractions for each step, facilitating fully automatic proofs, and (in future work) and the automatic synthesis of the intermediate programs and the final implementation. We have used refinement to automatically verify a file system, a distributed protocol, and also to prove the absence of bugs in a dynamic software adaptation protocol.

In our work on refinement, we focused on developing frameworks for programmers, so that they can develop their advanced systems and simultaneously obtain proofs of correctness [28]. Our specific priorities are developing reusable abstractions for the intermediate layers, i.e., the refinement variants of the source-level program. We also seek to automate the verification, at each layer, so that we can propose adapted code variants automatically (with a synthesizer) and reliably accept them if they verify [29].

4. Results and Discussion

This report cannot describe in depth all results developed by the SandCat project. Publications [30] [31] [6] [32] [33] [34] [35] [36] [37] [38] [39] [13] [40] [41] [42] [17] [43] [44] [45] [22] [28] [15] [9] provide such depth. This report highlights those projects that have been evaluated by the BRASS technical area (TA) 4 performers in the three phases of the project.

In Phase I, we developed and evaluated an adaptive large scale visualization (Challenge Problem (CP) I.1). This work was based on our work on automatically generating visualization layout engines [46, 47]. We also designed and evaluated how an application can adapt to a file system with weaker consistency properties (CP I.2).

In Phase II, we demonstrated adaptations using data-processing applications (CP II.1) that read from and write to a QoS-adaptive image store (CP II.2). Both CP II.1 and CP II.2 take advantage of an adaptive file system (CP II.3), as detailed in this subsection.

In Phase III, we solved three Challenge Problems. CP III.1 is motivated by the current state-of-the-art in numerical adaptation, which is characterized by painstaking manual effort of a numerical methods expert. Our goal was to demonstrate the feasibility of replacing important parts of such efforts using automated analysis and rewriting with Herbie. In CP III.2, we provided for evaluation Staccato, our dynamic-analysis tool for detecting and repairing errors in systems that adapt via dynamic reconfiguration. Lastly, in CP III.3, we solved a Constraint Satisfaction Problem using the case study of Flight Test Adaptation provided by Southwest Research Institute (SwRI).

4.1 Challenge Problem I.1: Adaptive Large-Scale Visualization

In this problem, we optimized two visualizations of hierarchical data sets. The two visualizations were a treemap and circlepacking.

Data Sets: The test data sets contain students and their grade point average (GPAs), Scholastic Aptitude Test (SAT) scores, and birthdates. The data set is tree-hierarchical in that it groups students into classrooms, schools, and city. There are thus four layers of data. This was a synthetic data set, generated randomly using the parameters for 103,252 students in 64 schools and 2,461 classes and expected to be tested on data sets of 10,000s of students.

Interactive data exploration (the perturbation): The user can explore questions such as how the GPA correlates with the SAT score, and how the birthday affects GPA and SAT. The size of each student's rectangle (in treemap) or circle (in circlepacking) corresponds to a

linear mix of GPA and SAT scores, which can be controlled using a slider. The size of the rectangles and circles for classes, schools, and cities is the aggregate of student score mixtures. The set of visualized students can also be sub-selected interactively based on their birthdays; the second slider selects the range of birthdays.

In response to slider movement, the analysis and visualization are recomputed. On a small data set, the graphics adjusts in real time as the slider moves. On large data files, the computation cannot keep up with the animation frame rate of 60 frames per second; the computation takes longer than the frame interval of 16ms. We thus consider the user-driven slider movement to be a perturbation to which the visualization needs to adapt.

Adaptation: To maintain responsiveness of the analysis and visualization, we approximate the computation in two ways: (1) we do not analyze the entire tree, skipping over the details in the lower layers of the tree; and (2) we avoid the layout and drawing of visual elements on which analysis was not performed. So, by measuring the time of the analysis and layout, we elide the students, classrooms, and schools as needed to maintain the frame rate.

4.2 Challenge Problem I.2: Adapting an Application to a new File System

Applications such as SandCat's big-data analytics system depend on persistent storage to record their state. Ensuring this data remains consistent after a system crash requires the application writer to understand subtle, file-system-dependent crash consistency guarantees. Rather than navigate these challenges directly, most applications delegate to a library layer offering higher-level guarantees, such as an atomic operation to replace the contents of a file. The library layer (such as that implemented by SQLite) strives to provide these higher-level operations with the minimal synchronization necessary to both preserve data integrity and maximize performance on the particular file system they run on.

Looming changes in persistent storage media and in file systems, and the corresponding crash guarantees they provide, will require re-implementing these library layers. On the hardware side, new shingled magnetic recording (SMR) disks offer increased density but allow writes to overlap existing unrelated data. Non-volatile memory technologies such as Intel's Optane, which entered the market in 2017, change the granularity of write atomicity and so require different synchronization patterns. On the file system side, developers continually update file system technology to better match today's workloads. For example, Apple Inc. recently upgraded millions of iOS devices to the new Apple File System (APFS). Each of these changes include different crash guarantees that libraries must understand and correctly address.

Given this proliferation of different crash guarantees, manually updating libraries to be correct for each platform will be tedious and error-prone. Instead, systems should be able to automatically adapt to new crash consistency models provided by the underlying storage stack. This adaptation could take as input the library implementation for an operation (e.g., atomic replacement), the top-level specification that operation offers (e.g., atomicity), and the crash consistency model for the new storage stack. Our adaptation engine, Ferrite, automatically inserts synchronization into the implementation to make it crash-safe on the given stack, and either guarantees it has inserted only the minimal synchronization necessary, or reports that the implementation cannot be made safe with synchronization alone (i.e., algorithmic changes are necessary) [15].

The setup: Imagine a big-data analytics system is running on a cluster of Linux nodes, each using some file system. This file system has certain crash consistency properties that are translated by a library layer for use the application running on the file system.

Change: Imagine that we load a new file system. This could happen as a result of porting the data set to new technology drives that demand a new file system. This new file system could be faster but could be less resilient to crashes. As a result, the library can no longer protect the data integrity of the data set on the new file system.

This challenge problem exercises changes in the internal consistency models resulting from installation of new components. Specifically, the perturbation in this CP will be the upgrade to a new file system which are developed for new kinds of disks and new work-loads. File systems differ in how they behave in the presence of crashes (in the file system itself or in the components above the file system or due to power losses or kernel panics). This is the result of aggressively caching and reordering data operations to improve performance. If a crash occurs between reordered operations, the disk may store an inconsistent data structure, leading to catastrophic data losses. Unfortunately, the Portable Operating System Interface (POSIX) standard has been largely silent on the behavior of file systems after a system crash.

Adaptation Scenario: A new file system will be added to existing file systems. The stack above the file system (database, data analytics, and the web server) will be able to use both the new and the existing file systems. The new file system may be faster but offers fewer guarantees. In this case, the applications needs to decide which of their storage needs can correctly use the file system. If the file system offers stronger guarantees, then the applications can remove synchronization between file system operations.

To ensure data integrity, components running on top of the file system need to be written to account for the crash model of the underlying file system. We formalize the idea of the crash model, and developed techniques for checking if a client component is correct with respect to

a file system's crash model. The crash model is analogous to the memory model developed for shared memory systems. We described the background in our paper on the Ferrite checker [15].

Testing: The evaluation system offers knobs that will configure properties of the new file systems. We configured the file system by defining its behavior by means of a set of litmus tests (see paper for examples of litmus tests [15]). This behavior was implemented with the model checker that we developed for Ferrite. The performance and correctness was evaluated by examining the data integrity of applications after crashes induced artificially in the test system.

4.3 Challenge Problem II.1: Automatic Adaptation of Legacy Code for Parallel Execution using Verified Lifting

The platform in this challenge problem comprises a file system, a data processing application, and its clients. The mission scenario was to preserve the semantics of the original application while adapting the runtime environment to massively parallel, distributed data processing frameworks.

This challenge problem demonstrates that it is possible to automatically rewrite originally sequential, non-distributed applications to leverage modern distributed data processing frameworks (e.g., Hadoop MapReduce, Spark, Flink), without manual code modification. Because the automatic rewrite preserves all original application invariants, it also greatly reduces the debugging and testing burden associated with a manual rewrite.

The CP platform uses a new compiler construction technique called verified lifting to automatically rewrite application code. Verified lifting uses program synthesis and verification (instead of syntax-directed rules) to search for efficient rewrites and proofs of their correctness.

Success was measured by generating provably correct translations of code fragments in the original program to MapReduce constructs, and evaluating the correctness and performance of the generated program running on the target framework.

4.4 Challenge Problem II.2: A QoS-Adaptive Image Store

The platform in this challenge problem comprises a key-value store of image data, its clients, and its underlying file system. The mission scenario is to maximize both client throughput and the quality of retrieved images, under constraints of limited memory and storage media bandwidth, and limited storage capacity.

This challenge problem demonstrates the ability to preserve intent (maximizing image quality and client throughput) under perturbations in the CPU, memory, storage media, and client components of the ecosystem. Specifically, the platform will dynamically adapt to increases in offered load (from clients or competing tenants of its memory, storage, and compute resources) by reducing the quality of retrieved images, within configured quality constraints. Additionally, the platform will dynamically adapt to limited storage capacity (e.g., in space-limited IoT devices or expensive media like flash, Random Access Memory (RAM), or non-Volatile Random Access Memory (NVRAM) by pruning data corresponding to image quality above the minimum configured level. These adaptations also work the other way, of course: an increase in available memory/storage bandwidth (which could be from improved hardware as well as reduced client demand) allows the platform to increase the quality of retrieved images, and an increase in available storage capacity allows us to store more high-frequency data per image (although it does not guarantee it will be used).

The key technology enabling this challenge problem is a customized progressive Joint Photographic Experts Group (JPEG) format, which reduces both the storage and bandwidth costs of rendering a single stored image at multiple resolutions. Rather than storing multiple versions of the same image for each supported resolution (or storing a single high-resolution version of the image and transcoding to lower resolutions on the fly), the format stores all frequency coefficients of the (maximum-resolution) image in scan order. The result is that any prefix of the full sequence can be used to reconstruct the image, with resolution determined by the length of the prefix (a mapping of file offsets to supported resolutions is stored)–the more of the sequence you read, the higher-quality the resulting image. No data needs to be duplicated to store different resolutions of the same image, and the transcoding process needs to read only the data actually required to reconstruct the image at a desired resolution (rather than the entire original image). This storage- and bandwidth-optimized format gives rise to two distinct adaptations: storage constraints can be addressed by simply truncating the sequence of frequency coefficients (to an offset corresponding to the minimum supported resolution), and bandwidth constraints (in memory or storage) can be addressed by reading a shorter prefix of the sequence of frequency coefficients (again corresponding to some supported resolution).

Success was measured by verifying that the adaptations described above successfully preserve intent (maximizing image quality and client throughput) under perturbations in client load and storage capacity. In particular, the platform should be able to realize a significant gain in throughput and storage utilization over conventional approaches (e.g., maintaining duplicate images at multiple resolutions, or a single version with on-the-fly transcoding), while satisfying a minimal image quality constraint (measured in peak signal-to-noise ratio).

The relative optimality of the challenge stage approach can be quantified by comparing its space and computation requirements to the two extreme baseline approaches: 1) store pre-rendered versions of all images at each supported resolution and return them directly, or 2) store a single high-resolution version of each image and transcode an image at the requested resolution on the fly. Approach 1) gives the best possible computational performance, at the worst possible storage cost, while approach 2) gives the best possible storage cost, with the worst possible computational performance. If we plot these three approaches as points on a computation-storage graph, the challenge stage approach should strictly dominate the two baseline approaches (i.e., its point lies below the line between the other two points), and its relative optimality is given by the orthogonal distance (in normalized units) of its point from the line between the two baseline points.

4.5 Challenge Problem II.3: Adaptive File Systems

The main motivation of CP 3 is that long-running computer systems need file systems that can adapt to hardware changes to meet their storage requirements over time. For example, when a storage device is upgraded to media with different characteristics (e.g., smaller/larger atomic write unit size or a battery-backed disk with a stronger consistency guarantee), the file system should be able to take advantage of these changes to achieve better performance while guaranteeing correctness.

In Phase 1, we have demonstrated how to adapt applications to changes in file system consistency models, in which we envision how future file systems may relax/reorder operations to strike a balance between consistency and performance. Our system in phase 1 guarantees 1) correctness, that applications will achieve the same level of consistency when adapting to a new file system; and 2) optimality, that applications will use the minimum number of file system flushes.

In Phase 2, we focused on how to implement a set of such new file systems. In other words, we generated a set of file system implementations with different consistency requirements with respect to hardware changes.

The key enabling technology for adaptive file systems was push-button formal verification. The file system is carefully designed and decomposed into layers; each layer allowed us to gradually adapt file systems into a more sophisticated, performant implementation. Our system checks the correctness of these implementations through a satisfiability modulo theories solver, Z3. The modular design of the file system allows Z3 to efficiently reason about the correctness of file systems before and after adaptations. We also use Z3 to optimize the file system, for instance, by checking whether it is correct to remove certain disk operations for better performance when adapting a file system to a new type of disk.

Success was measured in two ways: 1) correctness, that a new file system achieves the required consistency level; and 2) optimality, that a new file system implementation uses the minimal number of disk barrier operations.

4.6 Challenge Problem III.1: Numerical Adaptations with Herbie

The original version of Herbie, a system to detect inaccurate floating point expressions and finds more accurate replacements, regime inference was often inaccurate (it was missing some regimes and inferring poor branches for others). As part of the BRASS SandCat effort, we improved regime inference to support effective adaptation to new input ranges. As we continued to improve regime inference, Phase 3 CP 1 evaluated Herbie's ability to help adapt programs to compute accurately on new inputs. The motivation for this style of adaptation comes from a classic problem with writing floating-point code: developers naively translate real-number expressions from engineering references (or sometimes even just Wikipedia) and test the code on a few inputs. This often produces results which seem accurate enough on a handful of simple test cases. However, once deployed, the system receives additional inputs which trigger bad rounding errors for key expressions (kernels) in the computation. Our goal was to demonstrate that Herbie's synthesis and regime inference can adapt such kernels automatically (offline) to fix such issues.

For each kernel benchmark, we specified a set of "typical inputs" informed by the domain the kernel is drawn from and the papers and/or unit tests associated with the kernel's application. We will also specified a set of "target inputs" for each kernel that corresponded to deployment in a more general context (e.g., if the kernel was extracted from a particular application and then included as part of a library). Success depended on the number of benchmarks Herbie adapted to handle the broader range of target inputs and the degree to which accuracy in those new regimes is maintained or improved. Our goal was to successfully adapt more than 25% of cases automatically.

4.7 Challenge Problem III.2: Finding Bugs in Dynamic Adaptation Systems with Staccato

In this Challenge Problem, we provided the evaluators real-world web applications that contained dynamic-reconfiguration errors; our Staccato dynamic-analysis tool for detecting and repairing such errors; and a testing framework for analyzing Staccato's effectiveness on these applications.

In particular, the CP targeted adaptable systems that support online configuration changes without a restart of the system, which we termed a dynamic configuration update (DCU). We focused on two types of configuration adaptation errors: 1) failure to propagate con-

figuration updates to all system components, and 2) failure to apply configuration updates atomically. In addition, we demonstrated that Staccato can automatically repair some types of errors at runtime without interrupting normal execution.

4.8 Staccato: Debugging Systems that Adapt via Dynamic Configuration Updates

Mission-critical systems must be adaptable to different environments, mission parameters, and resource constraints. To promote usability across a wide array of deployment scenarios, adaptable software will often support a set of configuration options, which allow a trained user to set parameters that control a program's execution. For example, a system may allow the user to configure the location of the database used for durable storage, or the number of threads to use for data processing. However, a program's execution environment is often dynamic, and can change mid-execution, which in turn necessitates changing the values for some configuration options. For example, database servers may become unavailable, or the number of threads used for processing must be reduced in response to resource availability changes. Mission-critical software is often subject to stringent uptime requirements and restarting the system to effect a configuration change is infeasible. Thus, to accommodate configurability and robustness, many systems support configuration options that can be changed at runtime. We call a configuration change at runtime a dynamic configuration update.

Unfortunately, it is difficult to implement dynamic configuration updates correctly. A bug in Solr, an open-source text search and indexing server, demonstrated this. The user may configure a set of analyzers that process text entering the system. These analyzers can remove stop words, apply tense normalization, etc. The user may also dynamically reconfigure the set of analyzers used, e.g., to remove or add stop words. Although Solr reported that changes to the relevant configuration option were successfully applied, the analyzers were not actually updated to reflect the new settings. Solr would then silently misprocess data and incorrectly answer user queries. The only indication that something was wrong was Solr's output, which required careful inspection on the part of the user. We found similar defects in multiple applications.

Staccato is a tool developed at UW that uses runtime monitoring to detect errors in dynamic configuration update mechanisms. Our tool checks one of two alternative correctness conditions for DCU systems, as chosen by the programmer. The first condition states that old versions of configuration options may not be used after a configuration up-date. The second states that only one version of a configuration option may be used during a single method execution. Both conditions provide a possible specification of program behavior in the

presence of dynamic configuration updates. The choice of correctness condition is application specific, and given by the programmer in the form of a few high-level annotations.

In addition to bug finding, Staccato provides support for program repair and bug avoidance. Staccato can transparently repair program schedules to hide insufficient synchronization around configuration accesses. In addition, when Staccato detects a value built from an out-of-date configuration, it calls a programmer-provided callback to update the stale value. This feature can even be used to support dynamic configuration updates for options that previously required a restart.

We demonstrated that: 1) Staccato is effective at finding configuration adaptation errors in real programs with minimal annotations, and 2) the repair callback mechanism can avoid bugs discovered in practice.

In the challenge problem of Runtime Monitoring and Bug Finding, for each application, we developed a test suite that simultaneously exercises the core functionality of the web application while randomly mutating the configuration. We provided a test harness that runs these test suites along with the Staccato dynamic analysis without automatic repair enabled. The test harness allowed setting the parameters that control the random mutation.

Prior investigation has identified a set of incorrect dynamic-configuration behaviors possible in these applications. Each test-suite execution that produces a Staccato "found bug" will produce a stack-trace indicating where the monitoring detected a violation of the appropriate correctness condition. We provided a script that classifies these stack-traces to indicate whether the reports are due to a known incorrect behavior (a "true positive") and conservatively assume any other report is a "false positive". Success was defined as finding at least one error in each application with a false positive rate under 5%.

In the challenge problem Repair and Update, we identified violations reported by the above test suite that can be mitigated with Staccato's automatic repair mechanism. In addition, we identified areas in the three benchmarks where the repair mechanism can be used to transparently introduce dynamic configuration update support. We added the necessary repair/update callbacks and provided a test harness that runs an extended version of the test suite above with Staccato's automatic repair enabled. This extended test suite will additionally mutate the configuration options for which we added update support. We defined success for the repair callbacks as correctly introducing an update without leading to new violations. Success for a repair callback was defined as avoiding a true positive observed when Staccato is run without repair support. Additional success criteria were defined as

allowing the application to continue running successfully, and correct response to future requests. Overall success was defined as a 90% success rate across both types of callbacks.

5. Conclusions

The University of Washington SandCat project responded to (1) the rapid change in hardware, systems software, and application workloads; and (2) the trend towards increased complexity of computer systems, especially at the interfaces which were intended to hide the complexity, by developing methods to automate analyzing existing components and synthesizing new ones. The SandCat adaptations are semantic and mostly performed in an offline mode. The technology itself aids programmers and system designers' productivity through automation, by overcoming existing manual code rewriting methods to adapt code. The results have been demonstrated on challenge problems from across the entire systems stack, from the CPU to compute kernels and data structures, and from disk devices to file systems, and databases.

A major milestone in the research area of program synthesis was achieved, as the synthesis technology developed by SandCat reached parity with human programmers on about a dozen different programming tasks. Several components of SandCat technology have been transitioned into the commercial marketplace, including verified lifting which ships in a commercial photo editing product (Adobe Photoshop); the Rosette architecture which is being used in the Synthetic Minds startup; and the ML kernels which are being commercialized by OctoML.

References

[1] Panchekha P, Sanchez-Stern A, Wilcox JR, Tatlock Z (2015) Automatically improving accuracy for floating point expressions. Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation, Portland, OR, USA, June 15-17, 2015, eds Grove D, Blackburn S (ACM), pp 1–11. https://doi.org/ 10.1145/2737924.2737959.
Available at https://doi.org/10.1145/2737924.2737959

[2] Sanchez-Stern A, Panchekha P, Lerner S, Tatlock Z (2018) Finding root causes of floating point error. Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2018, Philadelphia, PA, USA, June 18-22, 2018, eds Foster JS, Grossman D (ACM), pp 256–269. https://doi.org/ 10.1145/3192366.3192411.
Available at https://doi.org/10.1145/3192366.3192411

[3] Thien D, Zorn B, Panchekha P, Tatlock Z (2019) Toward multi-precision, multi-format numerics. 2019 IEEE/ACM 3rd International Workshop on Software Correctness for HPC Applications (Correctness), Denver, CO, USA, November 18, 2019, eds Laguna I, Rubio-Gonzalez´ C (IEEE), pp 19–26. https://doi.org/10.1109/Correctness49594.2019.00008. Available at https://doi.org/ 10.1109/Correctness49594.2019.00008

[4] Becker H, Panchekha P, Darulova E, Tatlock Z (2018) Combining tools for optimization and analysis of floating-point computations. Formal Methods - 22nd International Symposium, FM 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 15-17, 2018, Proceedings, eds Havelund K, Peleska J, Roscoe B, de Vink EP (Springer), Lecture Notes in Computer Science, Vol. 10951, pp 355–363. https://doi.org/10.1007/978-3-319-95582-7n 21.
Available at https://doi.org/10.1007/978-3-319-95582-7 21

[5] Damouche N, Martel M, Panchekha P, Qiu C, Sanchez-Stern A, Tatlock Z (2016) Toward a standard benchmark format and suite for floating-point analysis. Numerical Software Verification - 9th International Workshop, NSV 2016, Toronto, ON, Canada, July 17-18, 2016, [collocated with CAV 2016], Revised Selected Papers, eds Bogomolov S, Martel M, Prabhakar P, Lecture Notes in Computer Science, Vol. 10152, pp 63–77. https://doi.org/10.1007/978-3-319-54292-8n 6.
Available at https://doi.org/10.1007/978-3-319-54292-8 6

[6] Phothilimthana PM, Elliott AS, Wang A, Jangda A, Hagedorn B, Barthels H, Kaufman SJ, Grover V, Torlak E, Bod´ık R (2019) Swizzle inventor: Data movement synthesis for

GPU kernels. Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, ASP-LOS 2019, Providence, RI, USA, April 13-17, 2019, eds Bahar I, Herlihy M, Witchel E, Lebeck AR (ACM), pp 65–78. https://doi.org/10.1145/3297858.3304059.
Available at https://doi.org/10.1145/3297858.3304059

[7] Phothilimthana PM, Thakur A, Bod´ık R, Dhurjati D (2016) Scaling up superoptimization. Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS '16, Atlanta, GA, USA, April 2-6, 2016, eds Conte T, Zhou Y (ACM), pp 297–310. https://doi.org/10.1145/2872362.2872387.
Available at https://doi.org/10.1145/2872362.2872387

[8] Cowan M, Moreau T, Chen T, Bornholt J, Ceze L (2020) Automatic generation of high-performance quantized machine learning kernels. CGO '20: 18th ACM/IEEE International Symposium on Code Generation and Optimization, San Diego, CA, USA, February, 2020 (ACM), pp 305–316. https://doi.org/10.1145/3368826.3377912.
Available at https://doi.org/10.1145/3368826.3377912

[9] Bornholt J, Torlak E, Grossman D, Ceze L (2016) Optimizing synthesis with metasketches. Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016, eds Bod´ık R, Majumdar R (ACM), pp 775–788. https://doi.org/ 10.1145/2837614.2837666.
Available at https://doi.org/10.1145/2837614.2837666

[10] Ahmad MBS, Cheung A (2016) Leveraging parallel data processing frameworks with verified lifting. Proceedings Fifth Workshop on Synthesis, SYNT@CAV 2016, Toronto, Canada, July 17-18, 2016, eds Piskac R, Dimitrova R, EPTCS, Vol. 229, pp 67–83. https://doi.org/10.4204/EPTCS.229.7.
Available at https://doi.org/10.4204/ EPTCS.229.7

[11] Ahmad MBS, Cheung A (2017) Optimizing data-intensive applications automatically by leveraging parallel data processing frameworks. Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017, eds Salihoglu S, Zhou W, Chirkova R, Yang J, Suciu D (ACM), pp 1675–1678. https://doi.org/10.1145/3035918.3056440.
Available at https://doi.org/10.1145/3035918.3056440

[12] Ahmad MBS, Cheung A (2018) Automatically leveraging mapreduce frameworks for data-intensive applications. Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018, eds Das G, Jermaine CM, Bernstein PA (ACM), pp 1205–1220. https://doi.org/10.1145/3183713.3196891.
Available at https://doi.org/10.1145/3183713.3196891

[13] Ahmad MBS, Ragan-Kelley J, Cheung A, Kamil S (2019) Automatically translating image processing libraries to halide. ACM Trans Graph 38(6):204:1– 204:13. https://doi.org/10.1145/3355089.3356549.
Available at https://doi.org/10.1145/3355089.3356549

[14] Bornholt J, Torlak E (2017) Synthesizing memory models from framework sketches and litmus tests. Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2017, Barcelona, Spain, June 18-23, 2017, eds Cohen A, Vechev MT (ACM), pp 467–481. https://doi.org/10.1145/3062341.3062353.
Available at https://doi.org/10.1145/3062341.3062353

[15] Bornholt J, Kaufmann A, Li J, Krishnamurthy A, Torlak E, Wang X (2016) Specifying and checking file system crash-consistency models. Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS '16, Atlanta, GA, USA, April 2-6, 2016, eds Conte T, Zhou Y (ACM), pp 83–98. https://doi.org/10.1145/2872362.2872406.
Available at https://doi.org/10.1145/2872362.2872406

[16] Sigurbjarnarson H, Bornholt J, Torlak E, Wang X (2016) Push-button verification of file systems via crash refinement. 12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, Savannah, GA, USA, November 2-4, 2016, eds Keeton K, Roscoe T (USENIX Association), pp 1–16.
Available at https://www.usenix.org/conference/osdi16/technical-sessions/presentation/sigurbjarnarson.

[17] Loncaric C, Torlak E, Ernst MD (2016) Fast synthesis of fast collections. Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2016, Santa Barbara, CA, USA, June 13-17, 2016, eds Krintz C, Berger E (ACM), pp 355–368. https://doi.org/10.1145/2908080.2908122.
Available at https://doi.org/10.1145/2908080.2908122

[18] Loncaric C, Ernst MD, Torlak E (2018) Generalized data structure synthesis. Proceedings of the 40th International Conference on Software Engineering, ICSE 2018,

Gothenburg, Sweden, May 27 - June 03, 2018, eds Chaudron M, Crnkovic I, Chechik M, Harman M (ACM), pp 958–968. https://doi.org/10.1145/3180155.3180211.
Available at https://doi.org/10.1145/3180155.3180211

[19] Torlak E, Bod´ık R (2014) A lightweight symbolic virtual machine for solver-aided host languages. ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '14, Edinburgh, United Kingdom - June 09 - 11, 2014, eds O'Boyle MFP, Pingali K (ACM), pp 530–541. https://doi.org/10.1145/2594291. 2594340. Available at https://doi.org/10.1145/2594291.2594340

[20] Nelson L, Bornholt J, Gu R, Baumann A, Torlak E, Wang X (2019) Scaling symbolic evaluation for automated verification of systems code with serval. Proceedings of the 27th ACM Symposium on Operating Systems Principles, SOSP 2019, Huntsville, ON, Canada, October 27-30, 2019, eds Brecht T, Williamson C (ACM), pp 225– 242. https://doi.org/10.1145/3341301.3359641.
Available at https://doi.org/10.1145/3341301.3359641

[21] Bornholt J, Torlak E (2018) Finding code that explodes under symbolic evaluation. PACMPL 2(OOPSLA):149:1–149:26. https://doi.org/10.1145/3276519.
Available at https://doi.org/10.1145/3276519

[22] Porncharoenwase S, Bornholt J, Torlak E (2020) Fixing code that explodes under symbolic evaluation. Verification, Model Checking, and Abstract Interpretation - 21st International Conference, VMCAI 2020, New Orleans, LA, USA, January 16-21, 2020, Proceedings, eds Beyer D, Zufferey D (Springer), Lecture Notes in Computer Science, Vol. 11990, pp 44–67. https://doi.org/10.1007/978-3-030-39322-9n 3.
Available at https://doi.org/10.1007/978-3-030-39322-9 3

[23] Toman J, Grossman D (2016) Staccato: A bug finder for dynamic configuration updates. 30th European Conference on Object-Oriented Programming, ECOOP 2016, July 18-22, 2016, Rome, Italy, eds Krishnamurthi S, Lerner BS (Schloss Dagstuhl - Leibniz-Zentrum fur¨ Informatik), LIPIcs, Vol. 56, pp 24:1–24:25. https://doi.org/10. 4230/LIPIcs.ECOOP.2016.24.
Available at https://doi.org/10.4230/LIPIcs.ECOOP. 2016.24

[24] Toman J, Grossman D (2018) Legato: An at-most-once analysis with applications to dynamic configuration updates. 32nd European Conference on Object-Oriented Programming, ECOOP 2018, July 16-21, 2018, Amsterdam, The Netherlands, ed Millstein TD (Schloss Dagstuhl - Leibniz-Zentrum fur¨ Informatik), LIPIcs, Vol. 109, pp 24:1–24:32. https://doi.org/10.4230/LIPIcs.ECOOP.2018.24.

Available at https:// doi.org/10.4230/LIPIcs.ECOOP.2018.24

[25] Toman J, Grossman D (2017) Taming the static analysis beast. 2nd Summit on Advances in Programming Languages, SNAPL 2017, May 7-10, 2017, Asilomar, CA, USA, eds Lerner BS, Bod´ık R, Krishnamurthi S (Schloss Dagstuhl - Leibniz-Zentrum fur¨ Informatik), LIPIcs, Vol. 71, pp 18:1–18:14. https://doi.org/10.4230/ LIPIcs.SNAPL.2017.18.
Available at https://doi.org/10.4230/LIPIcs.SNAPL.2017.18

[26] Mazumdar A, Moreau T, Kim S, Cowan M, Alaghi A, Ceze L, Oskin M, Sathe V (2017) Exploring computation-communication tradeoffs in camera systems. 2017 IEEE International Symposium on Workload Characterization, IISWC 2017, Seattle, WA, USA, October 1-3, 2017 (IEEE Computer Society), pp 177–186. https://doi.org/10.1109/IISWC.2017.8167775.
Available at https://doi.org/10.1109/IISWC.2017.8167775

[27] Phothilimthana PM, Liu M, Kaufmann A, Peter S, Bod´ık R, Anderson TE (2018) Floem: A programming system for nic-accelerated network applications. 13th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2018, Carlsbad, CA, USA, October 8-10, 2018, eds Arpaci-Dusseau AC, Voelker G (USENIX Association), pp 663–679.
Available at https://www.usenix.org/conference/osdi18/presentation/phothilimthana.

[28] Sigurbjarnarson H, Nelson L, Castro-Karney B, Bornholt J, Torlak E, Wang X (2018) Nickel: A framework for design and verification of information flow control systems. 13th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2018, Carlsbad, CA, USA, October 8-10, 2018, eds Arpaci-Dusseau AC, Voelker G (USENIX Association), pp 287–305.
Available at https://www.usenix.org/conference/osdi18/presentation/sigurbjarnarson.

[29] Nelson L, Sigurbjarnarson H, Zhang K, Johnson D, Bornholt J, Torlak E, Wang X (2017) Hyperkernel: Push-button verification of an OS kernel. Proceedings of the 26th Symposium on Operating Systems Principles, Shanghai, China, October 28-31, 2017 (ACM), pp 252–269. https://doi.org/10.1145/3132747.3132748.
Available at https://doi.org/10.1145/3132747.3132748

[30] Wang C, Feng Y, Bod´ık R, Cheung A, Dillig I (2020) Visualization by example. PACMPL 4(POPL):49:1–49:28. https://doi.org/10.1145/3371117.
Available at https://doi.org/10.1145/3371117

[31] Hagedorn B, Elliott AS, Barthels H, Bod´ık R, Grover V (2020) Fireiron: A scheduling language for high-performance linear algebra on gpus. CoRR abs/2003.06324. 2003. 06324 Available at https://arxiv.org/abs/2003.06324.

[32] Chasins SE, Mueller M, Bod´ık R (2018) Rousillon: Scraping distributed hierarchical web data. The 31st Annual ACM Symposium on User Interface Software and Technology, UIST 2018, Berlin, Germany, October 14-17, 2018, eds Baudisch P, Schmidt A, Wilson A (ACM), pp 963–975. https://doi.org/10.1145/3242587.3242661.
Available at https://doi.org/10.1145/3242587.3242661

[33] Haynes B, Mazumdar A, Balazinska M, Ceze L, Cheung A (2019) Visual road: A video data management benchmark. Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019, eds Boncz PA, Manegold S, Ailamaki A, Deshpande A, Kraska T (ACM), pp 972–987. https://doi.org/10.1145/3299869.3324955.
Available at https://doi.org/10.1145/3299869.3324955

[34] Moreau T, Chen T, Ceze L (2018) Leveraging the VTA-TVM hardware-software stack for FPGA acceleration of 8-bit resnet-18 inference. Proceedings of the 1st on Reproducible Quality-Efficient Systems Tournament on Co-designing Pareto-efficient Deep Learning, ReQuEST@ASPLOS 2018, Williamsburg, VA, USA, March 24, 2018, eds Ceze L, Jerger NDE, Falsafi B, Fursin G, Lokhmotov A, Moreau T, Sampson A, Stanley-Marbell P (ACM), p 5. https://doi.org/10.1145/3229762.3229766.
Available at https://doi.org/10.1145/3229762.3229766

[35] Moreau T, Chen T, Vega L, Roesch J, Yan EQ, Zheng L, Fromm J, Jiang Z, Ceze L, Guestrin C, Krishnamurthy A (2019) A hardware-software blueprint for flexible deep learning specialization. IEEE Micro 39(5):8–16. https://doi.org/10.1109/MM.2019. 2928962.
Available at https://doi.org/10.1109/MM.2019.2928962

[36] Luo L, Nelson J, Ceze L, Phanishayee A, Krishnamurthy A (2018) Parameter hub: a rack-scale parameter server for distributed deep neural network training. Proceedings of the ACM Symposium on Cloud Computing, SoCC 2018, Carlsbad, CA, USA, October 11-13, 2018 (ACM), pp 41–54. https://doi.org/10.1145/3267809.3267840.
Available at https://doi.org/10.1145/3267809.3267840

[37] Chen T, Zheng L, Yan EQ, Jiang Z, Moreau T, Ceze L, Guestrin C, Krishnamurthy A (2018) Learning to optimize tensor programs. Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS

2018, 3-8 December 2018, Montreal,´ Canada, eds Bengio S, Wallach HM, Larochelle H, Grauman K, Cesa-Bianchi N, Garnett R, pp 3393–3404.
Available at http://papers.nips.cc/paper/7599-learning-to-optimize-tensor-programs.

[38] Lee VT, Mazumdar A, del Mundo CC, Alaghi A, Ceze L, Oskin M (2017) POSTER: application-driven near-data processing for similarity search. 26th Inter-national Conference on Parallel Architectures and Compilation Techniques, PACT 2017, Portland, OR, USA, September 9-13, 2017 (IEEE Computer Society), pp 132–133. https://doi.org/10.1109/PACT.2017.25.
Available at https://doi.org/10.1109/ PACT.2017.25

[39] Willsey M, Lee VT, Cheung A, Bod´ık R, Ceze L (2019) Iterative search for reconfigurable accelerator blocks with a compiler in the loop. IEEE Trans on CAD of Integrated Circuits and Systems 38(3):407–418. https://doi.org/10.1109/TCAD.2018. 2878194.
Available at https://doi.org/10.1109/TCAD.2018.2878194

[40] Iyer S, Konstas I, Cheung A, Zettlemoyer L (2018) Mapping language to code in programmatic context. Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018, eds Riloff E, Chiang D, Hockenmaier J, Tsujii J (Association for Computational Linguistics), , pp 1643–1652. https://doi.org/10.18653/v1/d18-1192.
Available at https://doi.org/10.18653/v1/d18-1192

[41] Wang C, Cheung A, Bod´ık R (2018) Speeding up symbolic reasoning for relational queries. PACMPL 2(OOPSLA):157:1–157:25. https://doi.org/10.1145/ 3276527.
Available at https://doi.org/10.1145/3276527

[42] Kamil S, Cheung A, Itzhaky S, Solar-Lezama A (2016) Verified lifting of stencil computations. Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2016, Santa Barbara, CA, USA, June 13-17, 2016, eds Krintz C, Berger E (ACM), pp 711–726. https://doi.org/10.1145/2908080.2908117.
Available at https://doi.org/10.1145/2908080.2908117

[43] Ernst MD, Macedonio D, Merro M, Spoto F (2016) Semantics for locking specifications. NASA Formal Methods - 8th International Symposium, NFM 2016, Minneapolis, MN, USA, June 7-9, 2016, Proceedings, eds Rayadurgam S, Tkachuk O (Springer), Lecture Notes in Computer Science, Vol. 9690, pp 355-372. 372. https://doi.org/10.1007/978-3-319-40648-0_27.

Available at https://doi.org/10.1007/978-3-319-40648-0_27

[44] Mullen E, Zuniga D, Tatlock Z, Grossman D (2016) Verified peephole optimizations for compcert. Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2016, Santa Barbara, CA, USA, June 13-17, 2016, eds Krintz C, Berger E (ACM), pp 448–461. https://doi.org/10.1145/2908080.2908109.
Available at https://doi.org/10.1145/2908080.2908109

[45] Woos D, Wilcox JR, Anton S, Tatlock Z, Ernst MD, Anderson TE (2016) Planning for change in a formal verification of the raft consensus protocol. Proceedings of the 5th ACM SIGPLAN Conference on Certified Programs and Proofs, Saint Peters-burg, FL, USA, January 20-22, 2016, eds Avigad J, Chlipala A (ACM), pp 154–165. 165. https://doi.org/10.1145/2854065.2854081.
Available at https://doi.org/10.1145/ 2854065.2854081

[46] Meyerovich LA, Torok ME, Atkinson E, Bod´ık R (2013) Parallel schedule synthesis for attribute grammars. ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPoPP '13, Shenzhen, China, February 23-27, 2013, eds Nicolau A, Shen X, Amarasinghe SP, Vuduc RW (ACM), pp 187–196. https://doi.org/ 10.1145/2442516.2442535.
Available at https://doi.org/10.1145/2442516.2442535

[47] Hottelier T, Bod´ık R (2015) Synthesis of layout engines from relational constraints. Proceedings of the 2015 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2015, part of SPLASH 2015, Pittsburgh, PA, USA, October 25-30, 2015, eds Aldrich J, Eugster P (ACM), pp 74–88. https://doi.org/10.1145/2814270.2814291.
Available at https://doi.org/10.1145/2814270.2814291

List of Symbols, Abbreviations, and Acronyms

APFS	Apple File System
BRASS	Building Resource Adaptive Software Systems
Cozy	not an acronym, a data structure synthesizer
CP	Challenge Problem
CPU	Central Processing Unit
DARPA	Defense Advanced Research Projects Agency
DCU	Dynamic Configuration Update
DSL	Domain Specific Language
Ferrite	not an acronym, framework to develop file system crash-consistency models
FPGA	Field Programmable Gate Array
GPA	Grade Point Average
GPU	Graphics Processing Unit
Herbie	not an acronym, automatically rewrites floating point expressions for accuracy
IoT	Internet of Things
JPEG	Joint Photographic Experts Group
MemSynth	not an acronym, language and tool for verifying, synthesizing, and disambiguating memory consistency models
ML	Machine Learning
NVRAM	non-Volatile Random Access Memory
POSIX	Portable Operating System Interface
QoS	Quality of Service
RAM	Random Access Memory
Rosette	not an acronym, programming language to build verification and synthesis tools
SandCat	not an acronym, a whole-stack platform for exploring software adaptation
SAT	Scholastic Aptitude Test
SMR	Shingled Magnetic Recording
SMT	Satisfiability Modulo Theories
Staccato	STAle Configuration and Consistency Analysis Tool
SwRI	Southwest Research Institute
TA	Technical Area
UW	University of Washington