IN PARTNERSHIP WITH:
CNRS

Université Pierre et Marie Curie (Paris 6)

Activity Report 2016

Project-Team WHISPER

Well Honed Infrastructure Software for Programming Environments and Runtimes

IN COLLABORATION WITH: Laboratoire d'informatique de Paris 6 (LIP6)

RESEARCH CENTER
Paris

THEME
Distributed Systems and middleware

Table of contents

Project-Team WHISPER

Creation of the Team: 2014 May 15, updated into Project-Team: 2015 December 01

Keywords:

Computer Science and Digital Science:

1. - Architectures, systems and networks
1.1.1. - Multicore
2. - Software
2.1.6. - Concurrent programming
2.1.10. - Domain-specific languages
2.1.11. - Proof languages
2.2.1. - Static analysis
2.2.3. - Run-time systems
2.3.1. - Embedded systems
2.3.3. - Real-time systems
2.4. - Verification, reliability, certification
2.4.3. - Proofs
2.5. - Software engineering
2.6. - Infrastructure software
2.6.1. - Operating systems
2.6.2. - Middleware
2.6.3. - Virtual machines

Other Research Topics and Application Domains:

5. - Industry of the future
5.2.1. - Road vehicles
5.2.3. - Aviation
5.2.4. - Aerospace
6.1. - Software industry
6.1.1. - Software engineering
6.1.2. - Software evolution, maintenance
6.3.3. - Network Management
6.5. - Information systems
6.6. - Embedded systems

1. Members

Research Scientists

Gilles Muller [Team leader, Inria, Senior Researcher, HDR]
Julia Lawall [Inria, Senior Researcher, HDR]
Pierre-Évariste Dagand [CNRS, Researcher]

Faculty Member

Bertil Folliot [Univ. Paris VI, Professor, HDR]

Engineer
Quentin Lambert [Inria, until Oct 2016]

PhD Students
Cédric Courtaud [Thales, from Mar 2016, granted by CIFRE]
Redha Gouicem [Univ. Paris VI, from Oct 2016]
Antoine Blin [Inria, until July 2016, Engineer from November 2016]

Visiting Scientist
Greg Kroah-Hartman [Linux Foundation, from October 2016]

Administrative Assistants
Helene Milome [Inria]
Eugène Kamdem [UPMC]

2. Overall Objectives

2.1. Overall Objectives

The focus of Whisper is on how to develop (new) and improve (existing) infrastructure software. Infrastructure software (also called systems software) is the software that underlies all computing. Such software allows applications to access resources and provides essential services such as memory management, synchronization and inter-process interactions. Starting bottom-up from the hardware, examples include virtual machine hypervisors, operating systems, managed runtime environments, standard libraries, and browsers, which amount to the new operating system layer for Internet applications. For such software, efficiency and correctness are fundamental. Any overhead will impact the performance of all supported applications. Any failure will prevent the supported applications from running correctly. Since computing now pervades our society, with few paper backup solutions, correctness of software at all levels is critical. Formal methods are increasingly being applied to operating systems code in the research community [44], [49], [87]. Still, such efforts require a huge amount of manpower and a high degree of expertise which makes this work difficult to replicate in standard infrastructure-software development.

In terms of methodology, Whisper is at the interface of the domains of operating systems, software engineering and programming languages. Our approach is to combine the study of problems in the development of real-world infrastructure software with concepts in programming language design and implementation, *e.g.*, of domain-specific languages, and knowledge of low-level system behavior. A focus of our work is on providing support for legacy code, while taking the needs and competences of ordinary system developers into account.

We aim at providing solutions that can be easily learned and adopted by system developers in the short term. Such solutions can be tools, such as Coccinelle [1], [8], [9] for transforming C programs (see Section 6.2), or domain-specific languages such as Devil [7] and Bossa [6] for designing drivers and kernel schedulers. Due to the small size of the team, Whisper mainly targets operating system kernels and runtimes for programming languages. We put an emphasis on achieving measurable improvements in performance and safety in practice, and on feeding these improvements back to the infrastructure software developer community.

3. Research Program

3.1. Scientific Foundations

3.1.1. Program analysis

A fundamental goal of the research in the Whisper team is to elicit and exploit the knowledge found in existing code. To do this in a way that scales to a large code base, systematic methods are needed to infer code properties. We may build on either static [33], [36], [39] or dynamic analysis [57], [61], [67]. Static

analysis consists of approximating the behavior of the source code from the source code alone, while dynamic analysis draws conclusions from observations of sample executions, typically of test cases. While dynamic analysis can be more accurate, because it has access to information about actual program behavior, obtaining adequate test cases is difficult. This difficulty is compounded for infrastructure software, where many, often obscure, cases must be handled, and external effects such as timing can have a significant impact. Thus, we expect to primarily use static analyses. Static analyses come in a range of flavors, varying in the extent to which the analysis is *sound*, *i.e.*, the extent to which the results are guaranteed to reflect possible run-time behaviors.

One form of sound static analysis is *abstract interpretation* [36]. In abstract interpretation, atomic terms are interpreted as sound abstractions of their values, and operators are interpreted as functions that soundly manipulate these abstract values. The analysis is then performed by interpreting the program in a compositional manner using these abstracted values and operators. Alternatively, *dataflow analysis* [48] iteratively infers connections between variable definitions and uses, in terms of local transition rules that describe how various kinds of program constructs may impact variable values. Schmidt has explored the relationship between abstract interpretation and dataflow analysis [76]. More recently, more general forms of symbolic execution [33] have emerged as a means of understanding complex code. In symbolic execution, concrete values are used when available, and these are complemented by constraints that are inferred from terms for which only partial information is available. Reasoning about these constraints is then used to prune infeasible paths, and obtain more precise results. A number of works apply symbolic execution to operating systems code [29], [31].

While sound approaches are guaranteed to give correct results, they typically do not scale to the very diverse code bases that are prevalent in infrastructure software. An important insight of Engler et al. [41] was that valuable information could be obtained even when sacrificing soundness, and that sacrificing soundness could make it possible to treat software at the scales of the kernels of the Linux or BSD operating systems. Indeed, for certain types of problems, on certain code bases, that may mostly follow certain coding conventions, it may mostly be safe to *e.g.*, ignore the effects of aliases, assume that variable values are unchanged by calls to unanalyzed functions, etc. Real code has to be understood by developers and thus cannot be too complicated, so such simplifying assumptions are likely to hold in practice. Nevertheless, approaches that sacrifice soundness also require the user to manually validate the results. Still, it is likely to be much more efficient for the user to perform a potentially complex manual analysis in a specific case, rather than to implement all possible required analyses and apply them everywhere in the code base. A refinement of unsound analysis is the CEGAR approach [34], in which a highly approximate analysis is complemented by a sound analysis that checks the individual reports of the approximate analysis, and then any errors in reasoning detected by the sound analysis are used to refine the approximate analysis. The CEGAR approach has been applied effectively on device driver code in tools developed at Microsoft [21]. The environment in which the driver executes, however, is still represented by possibly unsound approximations.

Going further in the direction of sacrificing soundness for scalability, the software engineering community has recently explored a number of approaches to code understanding based on techniques developed in the areas of natural language understanding, data mining, and information retrieval. These approaches view code, as well as other software-reated artifacts, such as documentation and postings on mailing lists, as bags of words structured in various ways. Statistical methods are then used to collect words or phrases that seem to be highly correlated, independently of the semantics of the program constructs that connect them. The obliviousness to program semantics can lead to many false positives (invalid conclusions) [53], but can also highlight trends that are not apparent at the low level of individual program statements. We have previously explored combining such statistical methods with more traditional static analysis in identifying faults in the usage of constants in Linux kernel code [52].

3.1.2. Domain Specific Languages

Writing low-level infrastructure code is tedious and difficult, and verifying it is even more so. To produce non-trivial programs, we could benefit from moving up the abstraction stack to enable both programming and proving as quickly as possible. Domain-specific languages (DSLs), also known as *little languages*, are a means to that end [5] [62].

3.1.2.1. Traditional approach.

Using little languages to aid in software development is a tried-and-trusted technique [79] by which programmers can express high-level ideas about the system at hand and avoid writing large quantities of formulaic C boilerplate.

This approach is typified by the Devil language for hardware access [7]. An OS programmer describes the register set of a hardware device in the high-level Devil language, which is then compiled into a library providing C functions to read and write values from the device registers. In doing so, Devil frees the programmer from having to write extensive bit-manipulation macros or inline functions to map between the values the OS code deals with, and the bit-representation used by the hardware: Devil generates code to do this automatically.

However, DSLs are not restricted to being "stub" compilers from declarative specifications. The Bossa language [6] is a prime example of a DSL involving imperative code (syntactically close to C) while offering a high-level of abstraction. This design of Bossa enables the developer to implement new process scheduling policies at a level of abstraction tailored to the application domain.

Conceptually, a DSL both abstracts away low-level details and justifies the abstraction by its semantics. In principle, it reduces development time by allowing the programmer to focus on high-level abstractions. The programmer needs to write less code, in a language with syntax and type checks adapted to the problem at hand, thus reducing the likelihood of errors.

3.1.2.2. Embedding DSLs.

The idea of a DSL has yet to realize its full potential in the OS community. Indeed, with the notable exception of interface definition languages for remote procedure call (RPC) stubs, most OS code is still written in a low-level language, such as C. Where DSL code generators are used in an OS, they tend to be extremely simple in both syntax and semantics. We conjecture that the effort to implement a given DSL usually outweighs its benefit. We identify several serious obstacles to using DSLs to build a modern OS: specifying what the generated code will look like, evolving the DSL over time, debugging generated code, implementing a bug-free code generator, and testing the DSL compiler.

Filet-o-Fish (FoF) [3] addresses these issues by providing a framework in which to build correct code generators from semantic specifications. This framework is presented as a Haskell library, enabling DSL writers to *embed* their languages within Haskell. DSL compilers built using FoF are quick to write, simple, and compact, but encode rigorous semantics for the generated code. They allow formal proofs of the run-time behavior of generated code, and automated testing of the code generator based on randomized inputs, providing greater test coverage than is usually feasible in a DSL. The use of FoF results in DSL compilers that OS developers can quickly implement and evolve, and that generate provably correct code. FoF has been used to build a number of domain-specific languages used in Barrelfish, [22] an OS for heterogeneous multicore systems developed at ETH Zurich.

The development of an embedded DSL requires a few supporting abstractions in the host programming language. FoF was developed in the purely functional language Haskell, thus benefiting from the type class mechanism for overloading, a flexible parser offering convenient syntactic sugar, and purity enabling a more algebraic approach based on small, composable combinators. Object-oriented languages – such as Smalltalk [42] and its descendant Pharo [26] – or multi-paradigm languages – such as the Scala programming language [64] – also offer a wide range of mechanisms enabling the development of embedded DSLs. Perhaps suprisingly, a low-level imperative language – such as C – can also be extended so as to enable the development of embedded compilers [23].

3.1.2.3. Certifying DSLs.

Whilst automated and interactive software verification tools are progressively being applied to larger and larger programs, we have not yet reached the point where large-scale, legacy software – such as the Linux kernel – could formally be proved "correct". DSLs enable a pragmatic approach, by which one could realistically strengthen a large legacy software by first narrowing down its critical component(s) and then focus our verification efforts onto these components.

Dependently-typed languages, such as Coq or Idris, offer an ideal environment for embedding DSLs [32], [27] in a unified framework enabling verification. Dependent types support the type-safe embedding of object languages and Coq's mixfix notation system enables reasonably idiomatic domain-specific concrete syntax. Coq's powerful abstraction facilities provide a flexible framework in which to not only implement and verify a range of domain-specific compilers [3], but also to combine them, and reason about their combination.

Working with many DSLs optimizes the "horizontal" compositionality of systems, and favors reuse of building blocks, by contrast with the "vertical" composition of the traditional compiler pipeline, involving a stack of comparatively large intermediate languages that are harder to reuse the higher one goes. The idea of building compilers from reusable building blocks is a common one, of course. But the interface contracts of such blocks tend to be complex, so combinations are hard to get right. We believe that being able to write and verify formal specifications for the pieces will make it possible to know when components can be combined, and should help in designing good interfaces.

Furthermore, the fact that Coq is also a system for formalizing mathematics enables one to establish a close, formal connection between embedded DSLs and non-trivial domain-specific models. The possibility of developing software in a truly "model-driven" way is an exciting one. Following this methodology, we have implemented a certified compiler from regular expressions to x86 machine code [4]. Interestingly, our development crucially relied on an existing Coq formalization, due to Braibant and Pous, [28] of the theory of Kleene algebras.

While these individual experiments seem to converge toward embedding domain-specific languages in rich type theories, further experimental validation is required. Indeed, Barrelfish is an extremely small software compared to the Linux kernel. The challenge lies in scaling this methodology up to large software systems. Doing so calls for a unified platform enabling the development of a myriad of DSLs, supporting code reuse across DSLs as well as providing support for mechanically-verified proofs.

3.2. Research direction: Tools for improving legacy infrastructure software

A cornerstone of our work on legacy infrastructure software is the Coccinelle program matching and transformation tool for C code. Coccinelle has been in continuous development since 2005. Today, Coccinelle is extensively used in the context of Linux kernel development, as well as in the development of other software, such as wine, python, kvm, and systemd. Currently, Coccinelle is a mature software project, and no research is being conducted on Coccinelle itself. Instead, we leverage Coccinelle in other research projects [24], [25], [65], [68], [72], [74], [78][10], [20], both for code exploration, to better understand at a large scale problems in Linux development, and as an essential component in tools that require program matching and transformation. The continuing development and use of Coccinelle is also a source of visibility in the Linux kernel developer community. We submitted the first patches to the Linux kernel based on Coccinelle in 2007. Since then, over 4500 patches have been accepted into the Linux kernel based on the use of Coccinelle, including around 3000 by over 500 developers from outside our research group.

Our recent work has focused on driver porting. Specifically, we have considered the problem of porting a Linux device driver across versions, particularly backporting, in which a modern driver needs to be used by a client who, typically for reasons of stability, is not able to update their Linux kernel to the most recent version. When multiple drivers need to be backported, they typically need many common changes, suggesting that Coccinelle could be applicable. Using Coccinelle, however, requires writing backporting transformation rules. In order to more fully automate the backporting (or symmetrically forward porting) process, these rules should be generated automatically. We have carried out a preliminary study in this direction with David Lo of Singapore Management University; this work, published at ICSME 2016 [17], is limited to a port from one version to the next one, in the case where the amount of change required is limited to a single line of code. Whisper has been awarded an ANR PRCI grant, to start in March 2017, to collaborate with the group of David Lo on scaling up the rule inference process and proposing a fully automatic porting solution.

3.3. Research direction: developing infrastructure software using Domain Specific Languages

We wish to pursue a *declarative* approach to developing infrastructure software. Indeed, there exists a significant gap between the high-level objectives of these systems and their implementation in low-level, imperative programming languages. To bridge that gap, we propose an approach based on domain-specific languages (DSLs). By abstracting away boilerplate code, DSLs increase the productivity of systems programmers. By providing a more declarative language, DSLs reduce the complexity of code, thus the likelihood of bugs.

Traditionally, systems are built by accretion of several, independent DSLs. For example, one might use Devil [7] to interact with devices, Bossa [6] to implement the scheduling policies. However, much effort is duplicated in implementing the back-ends of the individual DSLs. Our long term goal is to design a unified framework for developing and composing DSLs, following our work on Filet-o-Fish [3]. By providing a single conceptual framework, we hope to amortize the development cost of a myriad of DSLs through a principled approach to reusing and composing them.

Beyond the software engineering aspects, a unified platform brings us closer to the implementation of mechanically-verified DSLs. Dagand's recent work using the Coq proof assistant as an x86 macro-assembler [4] is a step in that direction, which belongs to a larger trend of hosting DSLs in dependent type theories [27], [63], [32]. A key benefit of those approaches is to provide – by construction – a formal, mechanized semantics to the DSLs thus developed. This semantics offers a foundation on which to base further verification efforts, whilst allowing interaction with non-verified code. We advocate a methodology based on incremental, piece-wise verification. Whilst building fully-certified systems from the top-down is a worthwhile endeavor [49], we wish to explore a bottom-up approach by which one focuses first and foremost on crucial subsystems and their associated properties.

Our current work on DSLs has two complementary goals: (i) the design of a unified framework for developing and composing DSLs, following our work on Filet-o-Fish, and (ii) the design of domain-specific languages for domains where there is a critical need for code correctness, and corresponding methodologies for proving properties of the run-time behavior of the system.

4. Application Domains

4.1. Linux

Linux is an open-source operating system that is used in settings ranging from embedded systems to supercomputers. The most recent release of the Linux kernel, v4.9, comprises over 14 million lines of code, and supports 31 different families of CPU architectures, 73 file systems, and thousands of device drivers. Linux is also in a rapid stage of development, with new versions being released roughly every 2.5 months. Recent versions have each incorporated around 13,500 commits, from around 1500 developers. These developers have a wide range of expertise, with some providing hundreds of patches per release, while others have contributed only one. Overall, the Linux kernel is critical software, but software in which the quality of the developed source code is highly variable. These features, combined with the fact that the Linux community is open to contributions and to the use of tools, make the Linux kernel an attractive target for software researchers. Tools that result from research can be directly integrated into the development of real software, where it can have a high, visible impact.

Starting from the work of Engler et al. [40], numerous research tools have been applied to the Linux kernel, typically for finding bugs [39], [56], [69], [80] or for computing software metrics [46], [85]. In our work, we have studied generic C bugs in Linux code [9], bugs in function protocol usage [50], [51], issues related to the processing of bug reports [73] and crash dumps [45], and the problem of backporting [68], illustrating the variety of issues that can be explored on this code base. Unique among research groups working in this area, we have furthermore developed numerous contacts in the Linux developer community. These contacts provide insights into the problems actually faced by developers and serve as a means of validating the practical relevance of our work. Section 6.3 presents our dissemination efforts to the Linux community.

4.2. Device Drivers

Device drivers are essential to modern computing, to provide applications with access, via the operating system, to physical devices such as keyboards, disks, networks, and cameras. Development of new computing paradigms, such as the internet of things, is hampered because device driver development is challenging and error-prone, requiring a high level of expertise in both the targeted OS and the specific device. Furthermore, implementing just one driver is often not sufficient; today's computing landscape is characterized by a number of OSes, *e.g.*, Linux, Windows, MacOS, BSD and many real time OSes, and each is found in a wide range of variants and versions. All of these factors make the development, porting, backporting, and maintenance of device drivers a critical problem for device manufacturers, industry that requires specific devices, and even for ordinary users.

The last fifteen years have seen a number of approaches directed towards easing device driver development. Réveillère, who was supervised by G. Muller, proposes Devil [7], a domain-specific language for describing the low-level interface of a device. Chipounov *et al.* propose RevNic, [31] a template-based approach for porting device drivers from one OS to another. Ryzhyk *et al.* propose Termite, [70], [71] an approach for synthesizing device driver code from a specification of an OS and a device. Currently, these approaches have been successfully applied to only a small number of toy drivers. Indeed, Kadav and Swift [47] observe that these approaches make assumptions that are not satisfied by many drivers; for example, the assumption that a driver involves little computation other than the direct interaction between the OS and the device. At the same time, a number of tools have been developed for finding bugs in driver code. These tools include SDV [21], Coverity [40], CP-Miner, [55] PR-Miner [56], and Coccinelle [8]. These approaches, however, focus on analyzing existing code, and do not provide guidelines on structuring drivers.

In summary, there is still a need for a methodology that first helps the developer understand the software architecture of drivers for commonly used operating systems, and then provides tools for the maintenance of existing drivers.

5. Highlights of the Year

5.1. Highlights of the Year

The main highlight of the year is the continuous spreading of Coccinelle within the developer community of the Linux kernel. We submitted the first patches to the Linux kernel based on Coccinelle in 2007. Since then, over 4500 patches have been accepted into the Linux kernel based on the use of Coccinelle, including around 3000 by over 500 developers from outside our research group. Another testimonial of the impact of our work is the signature of a Memorendum Of Understanding (MOU) with the Linux Foundation. As part of the MOU, Greg Kroah-Hartman will spend a year with Whisper starting in October 2016. Kroah-Hartman is one of the leading developers of the Linux kernel, and is one of only a few developers employed by the Linux Foundation, with another being Linus Torvalds. Greg participated in the activities of the Whisper team around the use of Coccinelle and research projects related to the Linux kernel, and he is a convinced ambassador of our research work.

Our work on Remote Core Locking (RCL) [10] was accepted in ACM Transaction in Computer Systems (TOCS) which is the most prestigious journal in systems. RCL is currently one of the most efficient locks for multicore architectures.

6. New Software and Platforms

6.1. Prequel

KEYWORDS: Code quality - Evolution - Infrastructure software

FUNCTIONAL DESCRIPTION

The commit history of a large, actively developed code base such as the Linux kernel is a gold mine of information on how evolutions should be made, how bugs should be fixed, etc. Nevertheless, the high volume of commits available and the rudimentary filtering tools provided imply that it is often necessary to wade through a lot of irrelevant information before finding example commits that can help with a specific software development problem. To address this issue, we have developed Prequel (Patch Query Language) [20]. Prequel builds on the semantic patch lamguage SmPL developed for Coccinelle, which is now well known to the Linux kernel developer community, to allow developers to scan the changes in a source code development history, taking into account not only the specific changes made, but also the context in which these changes occur. As the history of a code base under active development quickly becomes large, with the Linux kernel incorporating around 13,000 commits on each 2-3 month release cycle, a particular goal in the development of Prequel has been to provide reasonable performance. Currently, most queries in our experiments complete in under minute when running on a single core on a standard laptop. So far, we have applied Prequel to the problem of understanding how to eliminate uses of deprecated functions [20], and are investigating how it may be useful in a systematic driver porting methodology.

Prequel is publicly available under GPLv2. The development of Prequel is supported by OSADL, and Julia Lawall presented Prequel at the 2016 OSADL networking day (https://www.osadl.org/OSADL-Networking-Day-2016.networking-day-2016.0.html).

- Participants: Julia Lawall and Gilles Muller
- Partners: IRILL - LIP6
- Contact: Julia Lawall
- URL: http://prequel-pql.gforge.inria.fr/

6.2. Coccinelle

KEYWORDS: Code quality - Evolution - Infrastructure software
FUNCTIONAL DESCRIPTION

Coccinelle is a tool for C code program matching and transformation that has been developed by members of the Whisper team over the last 10 years [8]. Coccinelle is widely used by the Linux kernel developer community and for other C software projects. Over the last three years, Coccinelle has benefited from the support of an engineer from the SED. Major improvements in 2016 include support for Python 3, independence from a no-longer-supported interface between Python and OCaml, better support for parallelism, and better support for integrating arbitrary predicates into the matching process. These features significantly improve performance and improve the uniformity of the rule specification language, thus providing a better experience for users. Coccinelle is at the foundation of much of our research work, including the ANR ITrans project, and these improvements will enhance and facilitate our research, accordingly.

Coccinelle is publicly available under GPLv2. In 2016, Julia Lawall presented Coccinelle in an invited keynote at the Linux Security Summit (http://events.linuxfoundation.org/events/archive/2016/linux-security-summit) and at a “birds of a feather” session at Linuxcon Europe (http://events.linuxfoundation.org/events/LinuxCon-europe).

- Participants: Julia Lawall, Gilles Muller, and Thierry Martinez
- Partners: IRILL - LIP6
- Contact: Julia Lawall
- URL: http://coccinelle.lip6.fr

6.3. Hector (BtrLinux)

KEYWORDS: Code quality - Evolution - Infrastructure software
FUNCTIONAL DESCRIPTION

A major source of errors in systems code is resource-release omission, which can lead to memory leaks and to crashes, if the system ends up in an inconsistent state. Currently, many tools exist that detect common patterns in software and detect faults as deviations from those patterns, but most suffer from high rates of false positives. Hector takes the novel approach of detecting inconsistencies local to a single function, and thus has been able to find over 300 faults in Linux kernel code and other C infrastructure software, with a rate of false positives of only 23%. Hector was originally the subject of the PhD thesis of Suman Saha [75]. Over the past two years, improving the robustness of the implementation of Hector has been the focus of ADT (young engineer position) BtrLinux supported by Inria, with the goal of making Hector publicly available and popularizing its use in the Linux kernel developer community. Some Linux kernel patches based on the use of Hector have been integrated into the Linux kernel, and the public release of Hector is in progress. The ADT position also involved the creation and maintenance of the website https://btrlinux.inria.fr/ as a showcase for the work of the Whisper team around Linux kernel development tools.

Building on his experience acquired in the ADT position, Quentin Lambert has recently been offered a position as an engineer at Wolfram MathCore AB.

- Participants: Quentin Lambert, Julia Lawall, and Gilles Muller
- Partners: IRILL - LIP6
- Contact: Julia Lawall
- URL: https://btrlinux.inria.fr/

6.4. ssrbit

FUNCTIONAL DESCRIPTION

ssrbit is a Coq library offering an efficient formalization of bit vectors, a refinement framework for abstractly reasoning about bitsets, and a trustworthy extraction of bit vectors to OCaml integers. Initially developed by Whisper members (Pierre-Évariste Dagand, Julia Lawall), the development has attracted an external contributor (Emilio Jesús Gallego Arias, postdoctoral researcher in CRI Mines-ParisTech), which led to significant improvements. We plan to improve the overall support and documentation so as to provide a full-featured library.

- Participants: Pierre Évariste Dagand, Julia Lawall, and Emilio Jesús Gallego Arias
- Contact: Pierre Évariste Dagand
- URL: https://github.com/ejgallego/ssrbit/

7. New Results

7.1. Software engineering for infrastructure software

Our main work in this area has focused on driver porting. We aim at fully automating the backporting (or symmetrically forward porting) process: given any driver for one Linux kernel version, one would like to obtain a driver that has the same functionality for another kernel version. This requires identifying the changes that are needed, obtaining examples of how to carry these changes out, and inferring from these examples a change that is appropriate for the given driver code. We have carried out a preliminary study in this direction with David Lo of Singapore Management University; this work, published at ICSME 2016 [17], is limited to a port from one version to the next one, in the case where the amount of change required is limited to a single line of code.

More general automation of backporting requires more extensive search for relevant examples. This raises issues of scalability, because the Linux kernel code history is very large, and of expressivity, because we need to be able to express complex patterns to obtain change examples that are most relevant to a particular backporting problem. To this end, we have been adapted the notation used by Coccinelle, which describes how a change should be carried out, into a *patch query language* that allows describing patterns of changes that have been previously performed. The associated tool, Prequel, can find patches that match a particular pattern among several hundred thousand commits, often in tens of seconds [20]. This work is supported in part by OSADL, a consortium of companies, mostly in Germany, supporting the use and development of open source software in automation and other industries.

We will continue research in this direction over the next three years as part of the ANR PRCI ITrans project, awarded in 2016 and to be carried out in 2017-2020.

7.2. Developing infrastructure software using Domain Specific Languages

To bootstrap our long-term effort in designing safe and composable domain-specific languages, we have initiated two exploratory actions involving a combination of advanced type-theoretic concepts and domain-specific compilation techniques. Both actions are complementary, the first adopts a bottom-up approach – going from low-level artifacts to high-level abstractions – while the second follows a top-down approach – offering a safe translation of high-level guarantees to low-level executable code.

Our first line of inquiry, of which some early results have been published at FLOPS 2016 [13], aims at bridging the formalization gap between low-level, bit-twiddling code and high-level, mathematical abstractions. As such, it provided us with an opportunity to experiment with using an interactive theorem prover to design abstractions in a bottom-up manner. We have developed a library (`ssrbit`, publicly available under an open-source license) for modeling and computing with bit vectors in the Coq [35] proof assistant. Because ease of proving and efficiency in computing are often incompatible objectives, this library offers a two pronged approach by offering an abstract specification for proving and an efficient implementation for computing; we have shown that the latter is correct with respect to the former. Using this model of bit-level operations, we have implemented a bitset library and proved its correctness with respect to the formalization of sets of finite types provided by the Ssreflect library [43], which is part of the Mathematical Components framework developed at the MSR-Inria joint center. This library thus enables a seamless interaction of sets for computing and sets for proving. This library also supports the trustworthy extraction of bitsets down to OCaml's machine integers: we gained greater confidence in our model by adopting a methodology based on exhaustive testing. This enabled us to implement three bit-twiddling applications in Coq (Bloom filter, n-queens, and the efficient enumeration of all k-combinations of a set), prove their correctness and obtain efficient low-level OCaml code.

Our second line of inquiry is influenced by the realization that domain-specific languages are often treating the symptoms rather than providing a cure. Infrastructure software is often developed in C, which suffers from many semantic kludges and is, as a result, hardly amenable to formal reasoning. Many domain-specific languages are born out of the frustration of being unable to guarantee static properties of one's code: more often than not, the resulting language is little more than a domain-specific variant of Pascal supporting custom static analyses and some form of transliteration to C. To achieve safety and composability, we believe that a more holistic approach is called for, involving not only the design of a domain-specific *syntax* but also of a domain-specific *semantics*. Concretely, we are exploring the design of *certified domain-specific compilers* that integrate, from the ground up, a denotational and domain-specific semantics as part of the design of a domain-specific language. This vision is illustrated by our work on the safe compilation of Coq programs into secure OCaml code [14], [18]. It combines ideas from gradual typing – through which types are compiled into run-time assertions – and the theory of ornaments [37] – through which Coq datatypes can be related to OCaml datatypes. Within this formal framework, we enable a secure interaction, termed *dependent interoperability*, between correct-by-construction software and untrusted programs, be it system calls or legacy libraries. To do so, we trade static guarantees for runtime checks, thus allowing OCaml values to be safely coerced to dependently-typed Coq values and, conversely, to expose dependently-typed Coq programs defensively as OCaml programs. Our framework is developed in Coq: it is constructive and verified in the strictest sense of

the terms. It thus becomes possible to internalize and hand-tune the extraction of dependently-typed programs to interoperable OCaml programs within Coq itself. This work is part of a collaboration with Eric Tanter, from the University of Chile, and Nicolas Tabareau, from the Ascola Inria project-team.

To further explore the realm of domain-specific compilers, we have been involved in the design and implementation of a certified compiler for the Lustre [30] synchronous dataflow language. Synchronous dataflow languages are widely used for the design of embedded systems: they allow a high-level description of the system and naturally lend themselves to a hierarchical design. This on-going work, in collaboration with members of the Parkas team and Gallium team of Inria Paris, formalizes the compilation of a synchronous data-flow language into an imperative sequential language, which is eventually translated to Cminor [54], one of CompCert's intermediate languages. This project illustrates perfectly our methodological position: the design of synchronous dataflow languages is first governed by semantic considerations (Kahn process networks and the synchrony hypothesis) that are then reifed into syntactic artefacts. The implementation of a certified compiler highlights this dependency on semantics, forcing us to give as crisp a semantics as possible for the proof effort to be manageable. This work is part of an on-going collaboration with Marc Pouzet and Tim Bourke, from the Parkas team of Inria Paris, Lionel Rieg, postdoc at Collège de France, and Xavier Leroy, from the Gallium Inria project-team.

In terms of DSL design for domains where correctness is critical, our current focus is on process scheduling and multicore architectures. Ten years ago, we developed Bossa, targeting process scheduling on unicore processors, and primarily focusing on the correctness of a scheduling policy with respect to the requirements of the target kernel. At that time, the main use cases were soft real-time applications, such as video playback. Bossa was and still continues to be used in teaching, because the associated verifications allow a student to develop a kernel-level process scheduling policy without the risk of a kernel crash. Today, however, there is again a need for the development of new scheduling policies, now targeting multicore architectures. As identified by Lozi *et al.* [59], large-scale server applications, having specific resource access properties, can exhibit pathological properties when run with the Linux kernel's various load balancing heuristics. We are working on a new domain-specific language, Ipanema, to allow expressing load balancing properties, and to enable verification of critical scheduling properties such as liveness; for the latter, we are exploring the use of tools such as the Z3 theorem prover from Microsoft, and the Leon theorem prover from EPFL. A first version of the language has been designed and we expect to have a prototype of Ipanema working next year. The work around Ipanema is the subject of a very active collaboration between researchers at four institutions (Inria, University of Nice, University of Grenoble, and EPFL (groups of V. Kuncak and W. Zwaenepoel)). Baptiste Lepers (EPFL) will be supported in 2017 as a postdoc as part of the Inria-EPFL joint laboratory.

Finally, in the context of the Multicore IPL, we are working with Jens Gustedt and Mariem Saeid of the Inria Camus project-team on developing a domain-specific language that eases programming with the ordered read-write lock (ORWL) execution model. The goal of this work is to provide a single execution model for parallel programs and to allow them to be deployed on multicore machines with varying architectures [16].

7.3. Run-time environments for multicore architectures

In the recent past, we acquired a solid expertise in multicore systems through the PhD of Jean-Pierre Lozi [60] and Florian David [38]. This expertise has led us to initiate several collaborations with industry partners, in the form of CIFRE PhD support. We first targeted real-time multicore systems with the goal of improving resource usage, through a cooperation with Renault and the PhD of Antoine Blin. Recently, we have started another cooperation on multicore real-time systems for avionics and space with Thales TRT, that is the topic of the PhD of Cédric Courtaud.

The PhD of Jean-Pierre Lozi [60] was on improving the performance locks on large multicore architectures. In an paper published at Usenix ATC 2012 [58], and more recently in an article published in 2016 in ACM Transactions on Computer Systems (TOCS) [10], we proposed a new locking technique, Remote Core Locking (RCL), that aims to accelerate the execution of critical sections in legacy applications on multicore architectures. RCL is currently one of the most efficient locking technique and the ATC 2012 paper has currently 67 citations on Google scholar. The idea of RCL is to replace lock acquisitions by optimized remote

procedure calls to a dedicated server hardware thread. RCL limits the performance collapse observed with other lock algorithms when many threads try to acquire a lock concurrently and removes the need to transfer lock-protected shared data to the hardware thread acquiring the lock because such data can typically remain in the server's cache. Eighteen applications were used to evaluate RCL from standard multicore benchmark suites, such as SPLASH-2 and Phoenix 2. By using RCL instead of Linux POSIX locks, performance is improved by up to 2.5 times on Memcached, and up to 11.6 times on Berkeley DB with the TPC-C client. On a SPARC machine with two Sun Ultrasparc T2+ processors and 128 hardware threads, performance is improved by up to 1.3 times with respect to Solaris POSIX locks on Memcached, and up to 7.9 times on Berkeley DB with the TPC-C client.

The PhD of Antoine Blin is on modern complex embedded systems that involve a mix of real-time and best-effort applications. The recent emergence of low-cost multicore processors raises the possibility of running both kinds of applications on a single machine, with virtualization ensuring isolation. Nevertheless, memory contention can introduce other sources of delay, that can lead to missed deadlines. We first investigated the source of memory contention for the Mibench benchmark in a paper published at NETYS 2016 [12]. Then, in a paper published at ECRTS 2016 [11], we present a combined offline/online memory bandwidth monitoring approach. Our approach estimates and limits the impact of the memory contention incurred by the best-effort applications on the execution time of the real-time application. Using our approach, the system designer can limit the overhead on the real-time application to under 5% of its expected execution time, while still enabling progress of the best-effort applications.

8. Bilateral Contracts and Grants with Industry

8.1. Bilateral Contracts with Industry

- Renault, 2014-2016, 45 000 euros. The purpose of this contract is to develop solutions for running a mix of real-time and best-effort applications on a small embedded multicore architecture. Our goal is to optimize the usage of the processor resource. The PhD of Antoine Blin is supported by a CIFRE fellowship with Renault.
- Orange Labs, 2016-2017, 60 000 euros. The purpose of this contract is to apply the techniques developed in the context of the PhD of Antoine Blin to the domain of Software Defined Networks where network functions are run using virtual machines on commodity multicore machines.
- Thales Research, 2016-2018, 45 000 euros. The purpose of this contract is to enable the usage of multicore architectures in avionics systems. More precisely, our goal is to develop optimizations for a software TDMA hypervisor developed by Thales that provides full time-isolation of tasks. The PhD of Cédric Courtaud is supported by a CIFRE fellowship with Thales Research.
- OSADL, 2016-2017, development of the Prequel patch query language, 20 000 euros. OSADL is an organization headquartered in Germany that promotes and supports the use of open source software in the automation and machine industry. The project is in the context of the OSADL project SIL2LinuxMP bringing together various companies in automotive and embedded sytems with the goal of developing methodologies for certifying the basic components of a GNU/Linux-based RTOS.

9. Partnerships and Cooperations

9.1. Regional Initiatives

- City of Paris, 2016-2019, 100 000 euros. As part of the "Émergence - young team" program the city of Paris is supporting part of our work on domain-specific languages.

9.2. National Initiatives

9.2.1. ANR

ITrans - awarded in 2016, duration 2017 - 2020

Members: LIP6 (Whisper), David Lo (Singapore Management University)

Coordinator: Julia Lawall

Whisper members: Julia Lawall, Gilles Muller

Funding: ANR PRCI, 287,820 euros.

Objectives:

Large, real-world software must continually change, to keep up with evolving requirements, fix bugs, and improve performance, maintainability, and security. This rate of change can pose difficulties for clients, whose code cannot always evolve at the same rate. This project will target the problems of *forward porting*, where one software component has to catch up to a code base with which it needs to interact, and *back porting*, in which it is desired to use a more modern component in a context where it is necessary to continue to use a legacy code base, focusing on the context of Linux device drivers. In this project, we will take a *history-guided source-code transformation-based* approach, which automatically traverses the history of the changes made to a software system, to find where changes in the code to be ported are required, gathers examples of the required changes, and generates change rules to incrementally back port or forward port the code. Our approach will be a success if it is able to automatically back and forward port a large number of drivers for the Linux operating system to various earlier and later versions of the Linux kernel with high accuracy while requiring minimal developer effort. This objective is not achievable by existing techniques.

Chronos network, Time and Events in Computer Science, Control Theory, Signal Processing, Computer Music, and Computational Neurosciences and Biology - (2015 - 2016)

Coordinator: Gerard Berry

Whisper member: Gilles Muller

Funding: ANR 2014, Défi “Société de l’information et de la communication”.

The Chronos interdisciplinary network aims at placing in close contact and cooperation researchers of a variety of scientific fields: computer science, control theory, signal processing, computer music, neurosciences, and computational biology. The scientific object of study will be the understanding, modeling, and handling of time- and event-based computation across the fields.

Chronos will work by organizing a regular global seminar on subjects ranging from open questions to concrete solutions in the research fields, workshops gathering subsets of the Chronos researchers to address specific issues more deeply, a final public symposium presenting the main contributions and results, and an associated compendium.

9.2.2. Multicore Inria Project Lab

The Multicore IPL is an Inria initiative, led by Gilles Muller, whose goal is to develop techniques for deploying parallel programs on heterogeneous multicore machines while preserving scalability and performance. The IPL brings together researchers from the ALF, Algorille, CAMUS, Compsys, DALI, REGAL, Runtime and Whisper Inria Teams. These connections provide access to a diversity of expertise on open source development and parallel computing, respectively. In this context, Gilles Muller and Julia Lawall are working with Jens Gustedt and Mariem Saeid of Inria Lorraine on developing a domain-specific language that eases programming with the ordered read-write lock (ORWL) execution model. The goal of this work is to provide a single execution model for parallel programs and to allow them to be deployed on multicore machines with varying architectures.

9.3. International Initiatives

9.3.1. Inria International Partners

9.3.1.1. Informal International Partners

We collaborate with David Lo and Lingxiao Jiang of Singapore Management University, who are experts in software mining, clone detection, and information retrieval techniques. Our work with Lo and/or Jiang has led to 7 joint publications since 2013 [66], [77], [81], [82], [83], [86], [84], at conferences including ASE and ICSME.

9.4. International Research Visitors

9.4.1. Visits of International Scientists

9.4.1.1. Internships

Natacha Crooks, PhD student at the University of Austin, Texas, spent three months in Whisper from May to August working on Ipanema.

Derek Palinski, undergraduate at Oberlin College, USA, spent January and June to August working on understanding of device driver evolution, including the evaluation of Prequel.

Vatika Harlalka, undergraduate at the International Institute of Information Technology - Hyderabad, India, spent May to July working on strategies for improving the performance of multicore real-time systems.

Denis Merigoux, final-year student from Ecole Polytechnique, spent March to August working on inference of Coccinelle semantic patches from examples.

Roman Delgado, undergraduate at UPMC, spent June to August working with Pierre-Évariste Dagand on implementing dependent induction in type theory.

Swaraj Dash, undergraduate at Cambridge University, spent August to September working with Pierre-Évariste Dagand on the derivative of indexed datatypes.

Redha Gouicem, Master 2 at UPMC, spent March to August working on memory access control for multicore real-time systems.

Axelle Piot, Master 2 at ENS, spent March to July working on Ipanema.

10. Dissemination

10.1. Promoting Scientific Activities

10.1.1. Scientific Events Selection

10.1.1.1. Member of the Conference Program Committees

- Pierre-Évariste Dagand: TyDe 2016 PC.
- Julia Lawall: FLOPS 2016 PC, PLDI 2016 EPC, LCTES 2016 PC, OOPSLA 2016 PC, ASE 2016 ERP, Middleware 2016 PC, GPCE 2016 PC, SLE 2016 PC, ICSME 2016 ERA.
- Gilles Muller: EDCC 2016 PC, OPODIS 2016 PC, DSN 2016 PC, ASPLOS 2016 PC.

10.1.2. Journal

10.1.2.1. Member of the Editorial Boards

- Julia Lawall: Editorial board of Science of Computer Programing (2008 - present).

10.1.2.2. Reviewer - Reviewing Activities

- Pierre-Évariste Dagand: Journal of Logical and Algebraic Methods in Programming (journal), Journal of Functional Programming (journal), Type-driven Development (workshop)
- Julia Lawall: Automated Software Engineering (journal).
- Gilles Muller: IEEE Transactions on Computers, Operating Systems Review.

10.1.3. Invited Talks

- Julia Lawall: PPL workshop (Japan) 2016, Linux Security Summit 2016, SPLASH 2016 Programming Languages Mentoring Workshop, IFIP WG 2.4 (Software Implementation Technology).

10.1.4. Research Administration

- Pierre-Évariste Dagand: Member of the steering committee for the Colloquium d'Informatique de L'UPMC Sorbonne Universités
- Julia Lawall: IFIP TC secretary (2012 - present).

  Hiring committees: Inria Paris (CR2, 2016), Bordeaux (MdC, 2016), CNAM (MdC, 2016)
- Gilles Muller: EuroSys steering committee (2013-2016), elected member of WG 10.4 (Dependability), representative of Inria in Sorbonne University's advisory committee for research, member of the project committee board of the Inria Paris Center, member of the Paris committee for allocating post-docs, PhD stipends and sabbaticals.
- Bertil Folliot: Elected member of the IFIP WG10.3 working group (Concurrent systems)

10.2. Teaching - Supervision - Juries

10.2.1. Teaching

- Licence: Pierre-Évariste Dagand, Distributed cooperating objects, L3, UPMC, France
- Professional Licence: Bertil Folliot, Programmation C, L2, UPMC, France
- Professional Licence: Bertil Folliot, Lab projects, L2, UPMC, France

10.2.2. Supervision

- PhD in progress : Antoine Blin, CIFRE Renault, Vers une utilisation efficace des multi-coeurs dans des systèmes embarqués à criticités multiples, 30 Janvier 2017, Gilles Muller, Julien Sopéna (Regal)
- PhD in progress : Mariem Saeid, Jens Gustedt (Camus), soutenance en 2017, Gilles Muller.
- PhD in progress : Cédric Courtaud, CIFRE Thalès, 2016-2018, Gilles Muller, Julien Sopéna (Regal).
- PhD in progress : Redha Gouicem, 2016-2018, Gilles Muller, Julien Sopéna (Regal).

10.2.3. Juries

- Pierre-Évariste Dagand: member of the Jury of Mahsa Najafzadeh (UPMC)
- Julia Lawall: PhD reporter for Pierre Wilke (Rennes), Alan Charpentier (Bordeaux), Guido Lena Cota (Milan, defense in 2017), Krishna Narasimhan (Frankfurt, defense in 2017). PhD jury for Ripon Saha (UT Austin).
- Gilles Muller: President of the PhD thesis of T. Tigori (U. of Nantes), Member of the Jury of V. Trigonakis (EPFL, Switzerland), Reporter of the PhD of A. Walker (U. of New South Wales, Australia).

10.3. Popularization

10.3.1. Talks

- Julia Lawall: Coccinelle: invited talk, Linux Security Summit, 2016.
- Julia Lawall: Coccinelle BoF: Linuxcon Europe 2016.

- Julia Lawall: Outreachy intern panel, Linuxcon Europe 2016.
- Julia Lawall: Prequel, 2016 OSADL networking day.

11. Bibliography

Major publications by the team in recent years

[1] J. BRUNEL, D. DOLIGEZ, R. R. HANSEN, J. L. LAWALL, G. MULLER. *A foundation for flow-based program matching using temporal logic and model checking*, in "POPL", Savannah, GA, USA, ACM, January 2009, pp. 114–126

[2] L. BURGY, L. RÉVEILLÈRE, J. L. LAWALL, G. MULLER. *Zebu: A Language-Based Approach for Network Protocol Message Processing*, in "IEEE Trans. Software Eng.", 2011, vol. 37, n^{o} 4, pp. 575-591

[3] P.-É. DAGAND, A. BAUMANN, T. ROSCOE. *Filet-o-Fish: practical and dependable domain-specific languages for OS development*, in "Programming Languages and Operating Systems (PLOS)", 2009, pp. 51–55

[4] A. KENNEDY, N. BENTON, J. B. JENSEN, P.-É. DAGAND. *Coq: The World's Best Macro Assembler?*, in "PPDP", Madrid, Spain, ACM, 2013, pp. 13–24

[5] G. MULLER, C. CONSEL, R. MARLET, L. P. BARRETO, F. MÉRILLON, L. RÉVEILLÈRE. *Towards Robust OSes for Appliances: A New Approach Based on Domain-specific Languages*, in "Proceedings of the 9th Workshop on ACM SIGOPS European Workshop: Beyond the PC: New Challenges for the Operating System", Kolding, Denmark, 2000, pp. 19–24

[6] G. MULLER, J. L. LAWALL, H. DUCHESNE. *A Framework for Simplifying the Development of Kernel Schedulers: Design and Performance Evaluation*, in "HASE - High Assurance Systems Engineering Conference", Heidelberg, Germany, IEEE, October 2005, pp. 56–65

[7] F. MÉRILLON, L. RÉVEILLÈRE, C. CONSEL, R. MARLET, G. MULLER. *Devil: An IDL for hardware programming*, in "Proceedings of the Fourth Symposium on Operating Systems Design and Implementation (OSDI)", San Diego, California, USENIX Association, October 2000, pp. 17–30

[8] Y. PADIOLEAU, J. L. LAWALL, R. R. HANSEN, G. MULLER. *Documenting and Automating Collateral Evolutions in Linux Device Drivers*, in "EuroSys", Glasgow, Scotland, March 2008, pp. 247–260

[9] N. PALIX, G. THOMAS, S. SAHA, C. CALVÈS, J. L. LAWALL, G. MULLER. *Faults in Linux 2.6*, in "ACM Transactions on Computer Systems", June 2014, vol. 32, n^{o} 2, pp. 4:1–4:40

Publications of the year

Articles in International Peer-Reviewed Journals

[10] J.-P. LOZI, F. DAVID, G. THOMAS, J. LAWALL, G. MULLER. *Fast and Portable Locking for Multicore Architectures*, in "ACM Transactions on Computer Systems", January 2016 [*DOI :* 10.1145/2845079], https://hal.inria.fr/hal-01252167

International Conferences with Proceedings

[11] A. BLIN, C. COURTAUD, J. SOPENA, J. LAWALL, G. MULLER. *Maximizing Parallelism without Exploding Deadlines in a Mixed Criticality Embedded System*, in "28th EUROMICRO Conference on Real-Time Systems (ECRTS'16)", Toulouse, France, July 2016, https://hal.inria.fr/hal-01346979

[12] A. BLIN, C. COURTAUD, J. SOPENA, J. LAWALL, G. MULLER. *Understanding the Memory Consumption of the MiBench Embedded Benchmark*, in "Netys", Marakech, Morocco, May 2016, https://hal.inria.fr/hal-01349421

[13] A. BLOT, P.-É. DAGAND, J. LAWALL. *From Sets to Bits in Coq*, in "FLOPS 2016", Kochi, Japan, March 2016, https://hal.archives-ouvertes.fr/hal-01251943

[14] P.-E. DAGAND, N. TABAREAU, É. TANTER. *Partial Type Equivalences for Verified Dependent Interoperability*, in "ICFP 2016 - 21st ACM SIGPLAN International Conference on Functional Programming", Nara, Japan, September 2016, pp. 298-310 [*DOI :* 10.1145/2951913.2951933], https://hal.inria.fr/hal-01328012

[15] K. NARASIMHAN, C. REICHENBACH, J. LAWALL. *Interactive Data Representation Migration: Exploiting Program Dependence to Aid Program Transformation*, in "PEPM 2017 Workshop on Partial Evaluation and Program Manipulation", Paris, France, January 2017, https://hal.inria.fr/hal-01408266

[16] M. SAIED, J. GUSTEDT, G. MULLER. *Automatic Code Generation for Iterative Multi-dimensional Stencil Computations*, in "High Performance Computing, Data, and Analitics", Hydarabat, India, A. BENOÎT (editor), IEEE, December 2016, https://hal.inria.fr/hal-01337093

[17] F. THUNG, D. X. B. LE, D. LO, J. LAWALL. *Recommending Code Changes for Automatic Backporting of Linux Device Drivers*, in "32nd IEEE International Conference on Software Maintenance and Evolution (ICSME)", Raleigh, North Carolina, United States, IEEE, October 2016, https://hal.inria.fr/hal-01355859

National Conferences with Proceedings

[18] T. BOURKE, P.-E. DAGAND, M. POUZET, L. RIEG. *Vérification de la génération modulaire du code impératif pour Lustre*, in "JFLA 2017 - Vingt-huitième Journées Francophones des Langages Applicatifs", Gourettes, France, January 2017, https://hal.inria.fr/hal-01403830

Research Reports

[19] A. BLIN, C. COURTAUD, J. SOPENA, J. LAWALL, G. MULLER. *Maximizing Parallelism without Exploding Deadlines in a Mixed Criticality Embedded System*, Inria, February 2016, n^{o} RR-8838, https://hal.inria.fr/hal-01268078

[20] J. LAWALL, Q. LAMBERT, G. MULLER. *Prequel: A Patch-Like Query Language for Commit History Search*, Inria Paris, June 2016, n^{o} RR-8918, https://hal.inria.fr/hal-01330861

References in notes

[21] T. BALL, E. BOUNIMOVA, B. COOK, V. LEVIN, J. LICHTENBERG, C. MCGARVEY, B. ONDRUSEK, S. K. RAJAMANI, A. USTUNER. *Thorough Static Analysis of Device Drivers*, in "EuroSys", 2006, pp. 73–85

[22] A. BAUMANN, P. BARHAM, P.-É. DAGAND, T. HARRIS, R. ISAACS, S. PETER, T. ROSCOE, A. SCHÜPBACH, A. SINGHANIA. *The multikernel: A new OS architecture for scalable multicore systems*, in "SOSP", 2009, pp. 29–44

[23] T. F. BISSYANDÉ, L. RÉVEILLÈRE, J. L. LAWALL, Y.-D. BROMBERG, G. MULLER. *Implementing an embedded compiler using program transformation rules*, in "Software: Practice and Experience", 2013

[24] T. F. BISSYANDÉ, L. RÉVEILLÈRE, J. LAWALL, Y.-D. BROMBERG, G. MULLER. *Implementing an Embedded Compiler using Program Transformation Rules*, in "Software: Practice and Experience", February 2015, vol. 45, n^{o} 2, pp. 177-196, https://hal.archives-ouvertes.fr/hal-00844536

[25] T. F. BISSYANDÉ, L. RÉVEILLÈRE, J. LAWALL, G. MULLER. *Ahead of Time Static Analysis for Automatic Generation of Debugging Interfaces to the Linux Kernel*, in "Automated Software Engineering", May 2014, pp. 1-39 [*DOI :* 10.1007/s10515-014-0152-4], https://hal.archives-ouvertes.fr/hal-00992283

[26] A. P. BLACK, S. DUCASSE, O. NIERSTRASZ, D. POLLET. *Pharo by Example*, Square Bracket Associates, 2010

[27] E. BRADY, K. HAMMOND. *Resource-Safe Systems Programming with Embedded Domain Specific Languages*, in "14th International Symposium on Practical Aspects of Declarative Languages (PADL)", LNCS, Springer, 2012, vol. 7149, pp. 242–257

[28] T. BRAIBANT, D. POUS. *An Efficient Coq Tactic for Deciding Kleene Algebras*, in "1st International Conference on Interactive Theorem Proving (ITP)", LNCS, Springer, 2010, vol. 6172, pp. 163–178

[29] C. CADAR, D. DUNBAR, D. R. ENGLER. *KLEE: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Programs*, in "OSDI", 2008, pp. 209–224

[30] P. CASPI, N. HALBWACHS, D. PILAUD, J. PLAICE. *Lustre: a declarative language for programming synchronous systems*, in "14th ACM Symposium on Principles of Programming Languages", ACM, 1987

[31] V. CHIPOUNOV, G. CANDEA. *Reverse Engineering of Binary Device Drivers with RevNIC*, in "EuroSys", 2010, pp. 167–180

[32] A. CHLIPALA. *The Bedrock Structured Programming System: Combining Generative Metaprogramming and Hoare Logic in an Extensible Program Verifier*, in "ICFP", 2013, pp. 391–402

[33] L. A. CLARKE. *A system to generate test data and symbolically execute programs*, in "IEEE Transactions on Software Engineering", 1976, vol. 2, n^{o} 3, pp. 215–222

[34] E. CLARKE, O. GRUMBERG, S. JHA, Y. LU, H. VEITH. *Counterexample-guided abstraction refinement for symbolic model checking*, in "J. ACM", 2003, vol. 50, n^{o} 5, pp. 752–794

[35] COQ DEVELOPMENT TEAM. *The Coq proof assistant reference manual*, 2015, http://coq.inria.fr

[36] P. COUSOT, R. COUSOT. *Abstract Interpretation: Past, Present and Future*, in "CSL-LICS", 2014, pp. 2:1–2:10

[37] P.-É. DAGAND. *Reusability and Dependent Types*, University of Strathclyde, 2013

[38] F. DAVID. *Continuous and Efficient Lock Profiling for Java on Multicore Architectures*, Université Pierre et Marie Curie - Paris VI, July 2015, https://hal.inria.fr/tel-01263203

[39] I. DILLIG, T. DILLIG, A. AIKEN. *Sound, complete and scalable path-sensitive analysis*, in "PLDI", June 2008, pp. 270–280

[40] D. R. ENGLER, B. CHELF, A. CHOU, S. HALLEM. *Checking System Rules Using System-Specific, Programmer-Written Compiler Extensions*, in "OSDI", 2000, pp. 1–16

[41] D. R. ENGLER, D. Y. CHEN, A. CHOU, B. CHELF. *Bugs as Deviant Behavior: A General Approach to Inferring Errors in Systems Code*, in "SOSP", 2001, pp. 57–72

[42] A. GOLDBERG, D. ROBSON. *Smalltalk-80: The Language and Its Implementation*, Addison-Wesley, 1983

[43] G. GONTHIER, A. MAHBOUBI, E. TASSI. *A Small Scale Reflection Extension for the Coq system*, Inria Saclay Ile de France, 2015, n^{o} RR-6455

[44] L. GU, A. VAYNBERG, B. FORD, Z. SHAO, D. COSTANZO. *CertiKOS: A Certified Kernel for Secure Cloud Computing*, in "Proceedings of the Second Asia-Pacific Workshop on Systems (APSys)", 2011, pp. 3:1–3:5

[45] L. GUO, J. L. LAWALL, G. MULLER. *Oops! Where did that code snippet come from?*, in "11th Working Conference on Mining Software Repositories, MSR", Hyderabad, India, ACM, May 2014, pp. 52–61

[46] A. ISRAELI, D. G. FEITELSON. *The Linux kernel as a case study in software evolution*, in "Journal of Systems and Software", 2010, vol. 83, n^{o} 3, pp. 485–501

[47] A. KADAV, M. M. SWIFT. *Understanding modern device drivers*, in "ASPLOS", 2012, pp. 87–98

[48] G. A. KILDALL. *A Unified Approach to Global Program Optimization*, in "POPL", 1973, pp. 194–206

[49] G. KLEIN, K. ELPHINSTONE, G. HEISER, J. ANDRONICK, D. COCK, P. DERRIN, D. ELKADUWE, K. ENGELHARDT, R. KOLANSKI, M. NORRISH, T. SEWELL, H. TUCH, S. WINWOOD. *seL4: formal verification of an OS kernel*, in "SOSP", 2009, pp. 207–220

[50] J. L. LAWALL, J. BRUNEL, N. PALIX, R. R. HANSEN, H. STUART, G. MULLER. *WYSIWIB: Exploiting fine-grained program structure in a scriptable API-usage protocol-finding process*, in "Software, Practice Experience", 2013, vol. 43, n^{o} 1, pp. 67–92

[51] J. L. LAWALL, B. LAURIE, R. R. HANSEN, N. PALIX, G. MULLER. *Finding Error Handling Bugs in OpenSSL using Coccinelle*, in "Proceeding of the 8th European Dependable Computing Conference (EDCC)", Valencia, Spain, April 2010, pp. 191–196

[52] J. L. LAWALL, D. LO. *An automated approach for finding variable-constant pairing bugs*, in "25th IEEE/ACM International Conference on Automated Software Engineering", Antwerp, Belgium, September 2010, pp. 103–112

[53] C. LE GOUES, W. WEIMER. *Specification Mining with Few False Positives*, in "TACAS", York, UK, Lecture Notes in Computer Science, March 2009, vol. 5505, pp. 292–306

[54] X. LEROY. *Formal verification of a realistic compiler*, in "Communications of the ACM", 2009, vol. 52, n^{o} 7, pp. 107–115

[55] Z. LI, S. LU, S. MYAGMAR, Y. ZHOU. *CP-Miner: A Tool for Finding Copy-paste and Related Bugs in Operating System Code*, in "OSDI", 2004, pp. 289–302

[56] Z. LI, Y. ZHOU. *PR-Miner: automatically extracting implicit programming rules and detecting violations in large software code*, in "Proceedings of the 10th European Software Engineering Conference", 2005, pp. 306–315

[57] D. LO, S. KHOO. *SMArTIC: towards building an accurate, robust and scalable specification miner*, in "FSE", 2006, pp. 265–275

[58] J.-P. LOZI, F. DAVID, G. THOMAS, J. L. LAWALL, G. MULLER. *Remote Core Locking: migrating critical-section execution to improve the performance of multithreaded applications*, in "USENIX Annual Technical Conference", Boston, MA, USA, June 2012, pp. 65–76

[59] J. LOZI, B. LEPERS, J. R. FUNSTON, F. GAUD, V. QUÉMA, A. FEDOROVA. *The Linux scheduler: a decade of wasted cores*, in "Proceedings of the Eleventh European Conference on Computer Systems, EuroSys 2016, London, United Kingdom, April 18-21, 2016", C. CADAR, P. PIETZUCH, K. KEETON, R. RODRIGUES (editors), ACM, 2016, pp. 1:1–1:16, http://doi.acm.org/10.1145/2901318.2901326

[60] J.-P. LOZI. *Towards more scalable mutual exclusion for multicore architectures*, Université Pierre et Marie Curie - Paris VI, July 2014, https://tel.archives-ouvertes.fr/tel-01067244

[61] S. LU, S. PARK, Y. ZHOU. *Finding Atomicity-Violation Bugs through Unserializable Interleaving Testing*, in "IEEE Transactions on Software Engineering", 2012, vol. 38, n^{o} 4, pp. 844–860

[62] M. MERNIK, J. HEERING, A. M. SLOANE. *When and How to Develop Domain-specific Languages*, in "ACM Comput. Surv.", December 2005, vol. 37, n^{o} 4, pp. 316–344, http://dx.doi.org/10.1145/1118890.1118892

[63] G. MORRISETT, G. TAN, J. TASSAROTTI, J.-B. TRISTAN, E. GAN. *RockSalt: better, faster, stronger SFI for the x86*, in "PLDI", 2012, pp. 395-404

[64] M. ODERSKY, T. ROMPF. *Unifying functional and object-oriented programming with Scala*, in "Commun. ACM", 2014, vol. 57, n^{o} 4, pp. 76–86

[65] M. C. OLESEN, R. R. HANSEN, J. L. LAWALL, N. PALIX. *Coccinelle: Tool support for automated CERT C Secure Coding Standard certification*, in "Science of Computer Programming", October 2014, vol. 91, n^{o} B, pp. 141–160, https://hal.inria.fr/hal-01096185

[66] K. PAVNEET SINGH, F. THUNG, D. LO, J. LAWALL. *An Empirical Study on the Adequacy of Testing in Open Source Projects*, in "21st Asia-Pacific Software Engineering Conference", Jeju, South Korea, December 2014, https://hal.inria.fr/hal-01096132

[67] T. REPS, T. BALL, M. DAS, J. LARUS. *The Use of Program Profiling for Software Maintenance with Applications to the Year 2000 Problem*, in "ESEC/FSE", 1997, pp. 432–449

[68] L. R. RODRIGUEZ, J. LAWALL. *Increasing Automation in the Backporting of Linux Drivers Using Coccinelle*, in "11th European Dependable Computing Conference - Dependability in Practice", Paris, France, 11th European Dependable Computing Conference - Dependability in Practice, November 2015, https://hal.inria.fr/hal-01213912

[69] C. RUBIO-GONZÁLEZ, H. S. GUNAWI, B. LIBLIT, R. H. ARPACI-DUSSEAU, A. C. ARPACI-DUSSEAU. *Error propagation analysis for file systems*, in "PLDI", Dublin, Ireland, ACM, June 2009, pp. 270–280

[70] L. RYZHYK, P. CHUBB, I. KUZ, E. LE SUEUR, G. HEISER. *Automatic device driver synthesis with Termite*, in "SOSP", 2009, pp. 73–86

[71] L. RYZHYK, A. WALKER, J. KEYS, A. LEGG, A. RAGHUNATH, M. STUMM, M. VIJ. *User-Guided Device Driver Synthesis*, in "OSDI", 2014, pp. 661–676

[72] R. K. SAHA, J. L. LAWALL, S. KHURSHID, D. E. PERRY. *On the Effectiveness of Information Retrieval Based Bug Localization for C Programs*, in "ICSME 2014 - 30th International Conference on Software Maintenance and Evolution", Victoria, Canada, IEEE, September 2014, pp. 161-170 [*DOI :* 10.1109/ICSME.2014.38], https://hal.inria.fr/hal-01086082

[73] R. SAHA, J. L. LAWALL, S. KHURSHID, D. E. PERRY. *On the Effectiveness of Information Retrieval based Bug Localization for C Programs*, in "International Conference on Software Maintenance and Evolution (ICSME)", Victoria, BC, Canada, September 2014

[74] S. SAHA, J.-P. LOZI, G. THOMAS, J. LAWALL, G. MULLER. *Hector: Detecting resource-release omission faults in error-handling code for systems software*, in "DSN 2013 - 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)", Budapest, Hungary, IEEE Computer Society, June 2013, pp. 1-12 [*DOI :* 10.1109/DSN.2013.6575307], https://hal.inria.fr/hal-00918079

[75] S. SAHA. *Improving the Quality of Error-Handling Code in Systems Software using Function-Local Information*, Université Pierre et Marie Curie - Paris VI, March 2013, https://tel.archives-ouvertes.fr/tel-00937807

[76] D. A. SCHMIDT. *Data Flow Analysis is Model Checking of Abstract Interpretations*, in "POPL", 1998, pp. 38–48

[77] P. SENNA, L. RÉVEILLÈRE, L. JIANG, D. LO, J. LAWALL, G. MULLER. *Understanding the genetic makeup of Linux device drivers*, in "PLOS'13 - 7th Workshop on Programming Languages and Operating Systems", Nemacolin Woodlands Resort, Pennsylvania, United States, ACM, November 2013 [*DOI :* 10.1145/2525528.2525536], https://hal.inria.fr/hal-00927070

[78] P. SENNA TSCHUDIN, J. LAWALL, G. MULLER. *3L: Learning Linux Logging*, in "BElgian-NEtherlands software eVOLution seminar (BENEVOL 2015)", Lille, France, December 2015, https://hal.inria.fr/hal-01239980

[79] M. SHAPIRO. *Purpose-built languages*, in "Commun. ACM", 2009, vol. 52, n^{o} 4, pp. 36–41

[80] R. TARTLER, D. LOHMANN, J. SINCERO, W. SCHRÖDER-PREIKSCHAT. *Feature consistency in compile-time-configurable system software: facing the Linux 10,000 feature problem*, in "EuroSys", 2011, pp. 47–60

[81] F. THUNG, D. LO, J. L. LAWALL. *Automated library recommendation*, in "WCRE 2013 - 20th Working Conference on Reverse Engineering", Koblenz, Germany, R. LÄMMEL, R. OLIVETO, R. ROBBES (editors), IEEE, October 2013, pp. 182-191 [*DOI :* 10.1109/WCRE.2013.6671293], https://hal.inria.fr/hal-00918076

[82] F. THUNG, S. WANG, D. LO, J. LAWALL. *Automatic recommendation of API methods from feature requests*, in "ASE 2013 - 28th IEEE/ACM International Conference on Automated Software Engineering", Palo Alto, California, United States, E. DENNEY, T. BULTAN, A. ZELLER (editors), IEEE, November 2013, https://hal.inria.fr/hal-00918828

[83] Y. TIAN, D. LO, J. LAWALL. *Automated construction of a software-specific word similarity database*, in "2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering, CSMR-WCRE", Antwerp, Belgium, IEEE, February 2014, pp. 44-53, https://hal.inria.fr/hal-01086077

[84] Y. TIAN, D. LO, J. LAWALL. *SEWordSim: software-specific word similarity database*, ACM, May 2014, pp. 568-571, ICSE Companion 2014 - Companion Proceedings of the 36th International Conference on Software Engineering, Poster [*DOI :* 10.1145/2591062.2591071], https://hal.inria.fr/hal-01086079

[85] W. WANG, M. GODFREY. *A Study of Cloning in the Linux SCSI Drivers*, in "Source Code Analysis and Manipulation (SCAM)", IEEE, 2011

[86] S. WANG, D. LO, J. LAWALL. *Compositional Vector Space Models for Improved Bug Localization*, in "30th International Conference on Software Maintenance and Evolution", Victoria, Canada, IEEE, September 2014, pp. 171-180, https://hal.inria.fr/hal-01086084

[87] J. YANG, C. HAWBLITZEL. *Safe to the Last Instruction: Automated Verification of a Type-safe Operating System*, in "PLDI", 2010, pp. 99–110