Research Statement: Specifications for Information Security and Systematic Program Analysis Derivations

Mounir Assaf

January 1, 2017

I am a young researcher trying to make impact on science, at my scale and within the scope of my abilities and means. I pursue ideal goals, and rely on pragmatic tools.

I use programming languages, logic and mathematical reasoning in order to revolutionise program analysis for software security. I rely on systematic design techniques, in order to devise modern and novel foundations capable of supporting my long-term goals: designing practical tools for end-to-end assurance of security requirements.

1 Systematic Design Techniques

I advocate systematic design techniques in my research to solve applied problems and provide foundational and reusable advances. Beyond intrinsic beauty, they allow me to tackle challenges and solve problems by completely separating creative processes from aspects involving "crunching formulas" or "exhausting greek letters".

My research involves leveraging my expertise of abstract interpretation and analytic combinatorics theories to design program analyses for security. Both theories require a specification, whose design is the creative part. With that specification, textbook calculations take over for systematic problem solving by relying on decades of well-documented insights.

Pedagogy is another perk of separating creativity and calculus; a good specification that is well explained is often sufficient to comprehend both the problem and its solution, without having to expose all details of calculations at first.

Sometimes, stepping into less explored corners of these textbook approaches requires deep understanding, dedication and risk-taking so that, maybe, "the powerful play goes on, and you may contribute a verse" – Walt Whitman. For abstract interpretation, I am contributing a verse. I hope to do the same for analytic combinatorics. Meanwhile, I will take great care in advocating its use in verification of asymptotic security properties [Assaf, 2015, Chapter 6].

1.1 Abstract Interpretation

Abstract interpretation is a theory for approximating the semantics of programs [Cousot and Cousot, 1977]. Specifications for abstract interpretation include a model of program computations and a mathematical object, called ***Galois connection***, linking the computation model to a property of interest, e.g., the range of program variables. The beauty of abstract interpretation is leveraging mathematical properties of Galois connections to systematically transpose complex and incomputable models of program executions into models that are simpler and tractable; this has been used as a foundation to design static analyses among others; an impressive list of topics covered by abstract interpretation appears at `https://www.di.ens.fr/~cousot/AI/`.

My first contribution to abstract interpretation is a contribution to the aforementioned list, extending the range of topics covered by abstract interpretation to monitoring for information security. Assaf and Naumann [2016] provides a foundation for using abstract interpretation to design information flow monitors (see Section Information Security for more information).

My second contribution to abstract interpretation is putting to use a computation model, at the level of sets of sets, to derive static analyses for security. Assaf [2015, Chapter 5] and Assaf et al. [2016, 2017] are all dedicated to working out the idea – at different stages of maturity – that specifications of some abstract interpreters should not be at the usual level of sets, but at the level of sets of sets. This is crucial to specify some program analyses for security.

Future work. My work on monitors by abstract interpretation have not yet benefited from the more general computation model, at the level of set of sets. This incremental result, to which we hint at, in the conclusion of Assaf et al. [2017], will have deeper impact on enforceable security requirements.

So far, my work focuses on using ***collecting semantics***, i.e. computation models, that are described by denotational semantics in order to specify and derive analyses for security. Figuring out the first derivation using collecting semantics – at the level of sets of sets – that is described by small-step semantics will have far-reaching impact (small-step semantics defines the meaning of programs through atomic discrete operations, in contrast to denotational semantics that describes program computations as mathematical functions). My preliminary investigations are promising; chaotic small-step iterations for describing computation models at the level of sets of sets are possible, and their abstraction requires ascertaining some invariants to be explored further. This work will eventually open up new possibilities for trading-off performance and precision by designing complex fixpoint iteration strategies. In addition, since most scalable static analysers implement small-step iterations strategies, this work will help understand the targeted changes that are needed to implement these security analyses as *modular standalone abstract domains* – avoiding invasive changes whenever a new security analysis is plugged in.

1.2 Analytic Combinatorics

Analytic combinatorics is a theory for predicting quantitative properties of large combinatorial structures [Flajolet and Sedgewick, 2009]. Specifications for analytic combinatorics include descriptions of how to construct combinatorial structures, as well as discrete functions computing the properties of interest. In a similar way to abstract interpretation, one focuses on describing how to construct combinatorial objects, e.g., words composed of two letters ‘a’ and ‘b’. Analytic combinatorics beautifully transposes this specification into quantitative and asymptotic information about combinatorial objects, by relying on mathematical tools from algebra (e.g, formal power series) and analysis (e.g., analysis of functions over the complex plane).

Analytic combinatorics has applications in many fields, e.g., compilation and analysis of algorithms. To the best of my knowledge, my dissertation [Assaf, 2015, Chapter 7] is the first to apply this theory to the field of quantitative program analysis of information flow (see section on Information Flow for more information).

In a nutshell, the observations attackers can make during a program’s execution determines various quantitative information flow metrics. This work relies on abstract interpretation to compute a specification of attackers’ observations. Analytic combinatorics takes over, in order to use this specification of attackers’ observations and *systematically* find quantitative information about the leakage of sensitive data.

Future work. My work relying on analytic combinatorics remains unpublished. It builds on work that is just now being published, and should find its way to a paper describing a tightly integrated vision [Assaf, 2015, Chapters 5, 6 and 7]: from designing security requirements, to leveraging abstract interpretation and analytic combinatorics theories for the systematic design of enforcement techniques.

Future work on improving these techniques pertains to Section Information Flow.

Another promising area to explore would be the automatic computation of analytic combinatorics specifications, by relying on abstract interpretation. A tight integration of these two theories, similarly to what is done in my dissertation, would greatly impact automated program analysis, e.g. for estimating computational complexity, or usage of resources as diverse as time, space, memory or threads. Such analyses will also find many applications in security. Beyond information flow security, one such application may provide an alternative abstract interpretation-based analysis for estimating the complexity of intrusion detection signatures [Goubault-Larrecq and Lachance, 2016].

1.3 More Perspectives

Both abstract interpretation and analytic combinatorics theories may benefit greatly from mechanisation, in order to anticipate the need for more assurance: not only do we need to prove that programs are correct wrt. formal requirements, we also need to ensure that the tools used to obtain these proofs are correct.

In abstract interpretation, ongoing efforts are exploring different techniques [Jourdan et al., 2015, Cachera and Pichardie, 2010, Darais and Horn, 2016, Blazy et al., 2016]. Adapting these efforts to abstract interpretations at the level of sets of sets may target provable and certifiable guarantees of security requirements.

Analytic combinatorics tools may also benefit from such a mechanisation effort, which may first aim at providing user-friendly libraries by taking inspiration and building on work by Boldo et al. [2015]. Eventually, this work would lead to certifiable proofs of quantitative properties of programs.

My expertise in both fields, and my grit, will empower me in this research endeavour.

2 Information Security

Programs are buggy. More often than not, software bugs lead to attack vectors that can be exploited by malicious entities for financial, political, or strategic advantage... Fortunately, many bugs can be detected by tools that are nowadays evolving from niche markets – embedded software in aeronautics, nuclear, or transportations – to general purpose software, e.g., open-source software. These bugs are the ones related to properties such as safety, runtime errors, buffer overflows, use-after-free or functional in-correctness. In sharp contrast, tools for detecting leakage of sensitive information or data corruption – confidentiality and integrity violations – are not up for the challenge. Not yet, at least! I do hope to change that.

Tools designed to enforce information security lack well-defined foundations that rely on the semantics of programs. They are not precise enough, since they rely mostly on techniques coined in seminal works in the field [Denning, 1976, Denning and Denning, 1977]. Most importantly, possibilities for improving these tools are not well understood. In fact, naive attempts to improve their precision by standard program analysis techniques break their correction – i.e. some information leaks are not detected anymore[1]. The problem lies in the design of enforcement mechanisms for ***information flow control*** – the field aiming at tracking how information propagates in programs to ensure they comply with a security policy.

Often, the design of these mechanisms relies on trial and error guided by mere intuitions, instead of relying on principled approaches that would guide and enrich these intuitions. I, myself, have designed enforcement mechanisms for information flow control by relying on mere intuition in my early research career. I would define information flow control mechanisms, then would attempt to prove they enforce a security policy; uncovering problems in the proof would lead me to backtrack and redefine the mechanisms before attempting another proof of correctness. Convinced that we ought to rely on much more principled and systematic techniques, I switched gears to

[1] I explain an instance of this problem in my defense slides: `http://massaf.net/files/PhdDefense_Assaf.pdf`

develop expertise and understanding of the framework of calculational abstract interpretation; this theory provides a foundation for the systematic design of correct-by-construction program analysis frameworks. Despite existing work relying on abstract interpretation for information flow control, these works forego the use of the calculational approach relying on Galois connections, missing the opportunity to harness the systematic and principled design approach of the theory. The main insight that is missing in previous work relying on abstract interpretation for information flow control, is a computation model of programs at the level of sets of sets. This computation model at the level of sets of sets captures ***hyperproperties*** [Clarkson and Schneider, 2010] – a formal characterisation of security requirements, in contrast to standard computation models at the level of sets – aimed at ***properties***. While previous work [Cousot and Cousot, 1994] points out the existence of this computation model at the level of sets of sets, to the best of my knowledge my work is the first to put it to use as a means to specify and derive analyses for information flow control. This is a major contribution to my application domain – information flow control. Yet, pushing further this idea will have impact on the theory of abstract interpretation as well – see Section on Abstract Interpretation.

My recent research [Assaf, 2015, Assaf et al., 2016, Assaf and Naumann, 2016, Assaf et al., 2017] focuses on harnessing the systematic design techniques offered by the calculational framework of abstract interpretation, to derive correct-by construction analyses for information security. This "deep conceptual work" (in the words of a reviewer) provides analyses for information flow with a semantics foundation. It shows off the benefits of a principled design technique, by providing novel insights explaining the inner workings of information flow policies and their enforcement mechanisms. As evidence of these benefits: I uncover various cases where existing analyses techniques are not as precise as they could be [Assaf, 2015, Assaf et al., 2016, Assaf and Naumann, 2016, Assaf et al., 2017]; I derive simple yet novel enforcement techniques that are within reach of the existing body of knowledge, but never explicitly pointed out before [Assaf et al., 2016, Assaf and Naumann, 2016]; I derive more intricate enforcement mechanisms, combining standard program analysis techniques from abstract interpretation, in order to improve precision [Assaf, 2015, Assaf and Naumann, 2016, Assaf et al., 2017]. This work also points out that the versatile specification tools of abstract interpretation, namely collecting semantics and Galois connections, can be of equal importance for the specification of security requirements, aka hyperproperties [Clarkson and Schneider, 2010].

Future work will build on this expertise and understanding, to streamline specifications of security requirements by relying on abstract interpretation specifications in the form of expressive semantics for hierarchies of computation models and Galois connections. These specifications will open up the door for an exciting journey, tapping into the comprehensive and systematic framework of calculational abstract interpretation to improve state-of-the-art analyses by developing novel enforcement techniques, scaling to richer language constructs and trading-off precision and performance.

2.1 Formal Requirements for Security

Hyperproperties are by now an established theory that sharply characterises security requirements. They serve to formalise the intuition appearing in the literature that security requirements are requirements relating and constraining several program executions, in contrast to standard properties that constrain each individual program execution – this is partly why a computation model at the level of sets of sets is needed to specify program analyses for security in abstract interpretation, instead of the standard model at the level of sets.

Assaf et al. [2017] points out that hyperproperties can be formalised in terms of more general computation models forming a hierarchy of semantics, and should not be limited to infinite sequences of states; this remark is folklore in abstract interpretation literature, but deserves to be advertised and disseminated in the security community, alongside the role of Galois connections and collecting semantics in specifying security requirements. This should benefit the specification

of formal security requirements by leveraging versatile and well-understood tools borrowed from abstract interpretation. In addition, this should also formalise the notion that information flow policies themselves can be organised into a hierarchy: formal security requirements against strong attacker models (attackers who make more observations about a system) implies security against weaker attacker models (attackers who make less observations about a system).

Future Work. Showcasing the benefits of the versatile tools offered by abstract interpretation, I intend to revisit in the near term some well-known definitions of security requirements in the information flow control literature with a particular emphasis on Galois connections and collecting semantics. Typically, nondeterminism remains a subject of debate in information flow literature; leveraging standard ways abstract interpretation deals with nondeterminism will lead to a simple yet interesting definition of security requirements that ought to be compared with existing qualitative and quantitative definitions. In addition, this work should also pave the road for scaling to richer languages features by extending existing definitions of security and accounting for additional attacker-observable vectors, e.g., interactive inputs, memory allocation or low level details such as persistent memory in stack and registers and some side channels.

2.2 Static Analysis & Monitoring

Since the seminal work of Denning in the seventies, the field of information flow control have achieved widespread interest, with many deep contributions pushing further our understanding, techniques and reasoning tools. One pitfall remained: most analyses techniques in information flow control depend too much on syntax and lack means of reasoning on the semantics of programs. Many research endeavours try to reconcile information flow control mechanisms and semantics reasoning by relying on abstract interpretation – a theory for approximating program semantics. Yet, for a long time, the first expert comment I would get after mentioning "information flow control" and "abstract interpretation" is "instrumentation", as a way to signify that prior work falls short of this endeavour and comfort my own dissatisfaction with prior literature on the subject. My most recent work meets these expectations [Assaf and Naumann, 2016, Assaf et al., 2017].

Halfway through my Phd, I also developed interest in quantitative information flow in order to estimate the amount of sensitive leakage a program may leak – in contrast to qualitative information flow which aims at simply deciding whether a program leaks sensitive information. There, the need for more semantics reasoning in security analyses was even more flagrant. Most quantitative information flow analysis techniques relying on approximation resort to analyses initially developed to ensure safety; this enables them to incorporate some semantics reasoning, at the cost of either great imprecision, or not supporting a large set of program features. The initial design of the cardinality analysis [Assaf, 2015, Assaf et al., 2016, Chapter 5] for quantitative information flow targets improving precision for programs that allow attackers to control some of the inputs. While these early works [Assaf, 2015, Assaf et al., 2016] introduce the quantitative analysis in a restricted setting for the two-point information flow lattice – program inputs are labelled as either public or confidential, they both put under the spotlight the express intent that the analysis should be at least as precise as state-of-the-art qualitative analyses. In abstract interpretation, being at least as precise means that state-of-the-art qualitative analyses can be derived as an abstraction of the quantitative cardinality analysis: Bingo! a formalisation of the cardinality analysis entirely within the calculational framework of abstract interpretation leads to a formalisation and a rationalisation of classical qualitative information flow analyses, relying only on the semantics of programs and no instrumentation. Even in the restricted setting of the two-point lattice, this is one step forward beyond existing work [Assaf et al., 2013b,a]. Insights developed while working out these ideas eventually lead to the rationalisation and improvement of information flow monitors [Assaf and Naumann, 2016] for the two-point information flow

lattice as well as the rationalisation and improvement of information flow static analyses for the general case with arbitrary information flow lattices [Assaf et al., 2017].

Future Work. A few words may characterise my research agenda: *understand, rationalise and improve program analyses for information security – within the systematic framework of calculational abstract interpretation.*

My work on information flow monitors have yet to benefit from the recent and more general framework that I use for the calculational derivation of static analyses for information flow control. Ongoing work rationalises monitors for general security lattices. From there on, future work in both static and dynamic information flow techniques will continue in lock-step. I will aim at improving automated existing program analysis techniques and designing novel ones to support richer language constructs, more precision, fine-tuned performances as well as more intricate security policies. Setting this work into the framework of calculational abstract interpretation will help making a difference and bringing information flow control analyses to a level of maturity on a par with other program analysis fields. In the near term, my focus will be on designing partitioning strategies for information flow analyses to improve precision. This will help in extending the supported language constructs to nondeterminism, interactive input-output behaviours, and more with a far better level of precision than existing work and building on my previous work relying on analytic combinatorics to specify attackers' observations. In addition, I will investigate iteration strategies for fixpoint computation of hyperproperty abstractions to fine-tune the trade-off between precision and performance; this is dependent on ongoing foundational theoretical work addressing abstract interpretation for general hyperproperties, including additional information flow security requirements which my work have not focused on yet. In addition, my work on static and dynamic analyses can greatly benefit from decentralised and hardware-assisted information flow tracking [Dalton et al., 2007, Abdul Wahab et al., 2016, de Amorim et al., 2016].

Last but not least, this work will support and actually be guided by my long-term objectives of addressing end-to-end information security in compilation toolchains by automated analyses.

2.3 More Perspectives

Modern architectures and toolchains for compilation are structured as pipelines of several compilation passes bridging the gap between high level languages and machine code: several passes throughout compilation, with each pass targeted at a specialised optimisation or code transformation from an intermediate representation to another. *This pipeline could be leveraged to ensure automated end-to-end assurance of formal requirements, from source level code to machine code.* First, to put the stress on automated analyses for security, they should be put to the test by ensuring that a security analysis that certifies a program in some intermediate language also certifies optimised versions of this program in the same intermediate language. This is an ideal and probably unattainable goal. However this ideal goal will serve as an experiment to guide further developments and ensure analyses for security move away from syntactic reasoning to include **mostly semantics reasoning**. Second, code transformations in the compilation pipeline that introduce additional attacker-observable vectors – those translating from one intermediate representation to another – should serve as checkpoints to plug in additional security analyses in order to ensure the low level observable vectors they introduce do not break security guarantees proven for higher level versions of the program. These two principles are dependent on tools for specifying security requirements and pushing security analyses to perform reasoning on program semantics far beyond the state-of-the-art; These tools are within reach in the framework of calculational abstract interpretation for hyperproperties that I have been developing and gaining expertise in, in order to specify and systematically derive security analyses.

The research project I am advocating is an original one thanks to novel multidisciplinary

approaches it puts forward and exploits. My experience as a young and mature independent researcher will prove valuable in offsetting the risks of this research endeavour; typically, this can be evidenced by the great passion and laser-focus I tackled my research with, in order to address eagerly sought-after and challenging problems in few, but solid and high-quality advances.

Publications

M. Assaf. *From Qualitative to Quantitative Program Analysis : Permissive Enforcement of Secure Information Flow*. PhD thesis, Université de Rennes 1, May 2015. URL `https://hal.inria.fr/tel-01184857`. *Best 2015 Phd award by the French Research Group on Programming Languages and Software Engineering GDR GPL.*

M. Assaf and D. Naumann. Calculational design of information flow monitors. In *IEEE Computer Security Foundations Symposium*, pages 210–224, 2016.

M. Assaf, J. Signoles, F. Tronel, and E. Totel. Moniteur hybride de flux d'information pour un langage supportant des pointeurs. In *SARSSI - 8ème Conférence sur la Sécurité des Architectures Réseaux et des Systèmes d'Information*, 2013a. URL `http://hal.inria.fr/hal-00909293`. *Best student paper award.*

M. Assaf, J. Signoles, F. Tronel, and É. Totel. Program transformation for non-interference verification on programs with pointers. In *Security and Privacy Protection in Information Processing Systems*, pages 231–244. 2013b. *Best student paper award.*

M. Assaf, J. Signoles, É. Totel, and F. Tronel. The cardinal abstraction for quantitative information flow. In *Workshop on Foundations of Computer Security (FCS)*, June 2016. `https://hal.inria.fr/hal-01334604`.

M. Assaf, D. Naumann, J. Signoles, É. Totel, and F. Tronel. Hypercollecting semantics and its application to static analysis of information flow. In *ACM Symposium on Principles of Programming Languages*, Jan. 2017. URL `https://arxiv.org/abs/1608.01654`.

References

M. Abdul Wahab, P. Cotret, M. Nasr Allah, G. Hiet, V. Lapotre, and G. Gogniat. Towards a hardware-assisted information flow tracking ecosystem for arm processors. In *26th International Conference on Field-Programmable Logic and Applications (FPL 2016)*, Aug. 2016. URL `https://hal.archives-ouvertes.fr/hal-01337579`.

S. Blazy, V. Laporte, and D. Pichardie. An abstract memory functor for verified c static analyzers. In *International Conference on Functional Programming*, page 14, 2016.

S. Boldo, C. Lelay, and G. Melquiond. Coquelicot: A user-friendly library of real analysis for Coq. *Mathematics in Computer Science*, 9(1):41–62, 2015.

D. Cachera and D. Pichardie. A certified denotational abstract interpreter. In *Interactive Theorem Proving (ITP)*, pages 9–24, 2010.

M. R. Clarkson and F. B. Schneider. Hyperproperties. *Journal of Computer Security*, 18(6): 1157–1210, 2010.

P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *ACM Symposium on Principles of Programming Languages*, pages 238–252, 1977.

P. Cousot and R. Cousot. Higher-order abstract interpretation (and application to comportment analysis generalizing strictness, termination, projection and per analysis of functional languages). In *International Conference on Computer Languages (ICCL)*, pages 95–112, 1994.

M. Dalton, H. Kannan, and C. Kozyrakis. Raksha: A flexible information flow architecture for software security. In *Proceedings of the 34th Annual International Symposium on Computer Architecture ISCA'07*, pages 482–493, 2007.

D. Darais and D. V. Horn. Constructive galois connections: Taming the galois connection framework for mechanized metatheory. In *International Conference on Functional Programming (ICFP)*, 2016.

A. A. de Amorim, N. Collins, A. DeHon, D. Demange, C. Hritcu, D. Pichardie, B. C. Pierce, R. Pollack, and A. Tolmach. A verified information-flow architecture. *Journal of Computer Security*, 24(6):689–734, 2016.

D. E. Denning. A lattice model of secure information flow. *Communications of ACM*, 19(5): 236–243, 1976.

D. E. R. Denning and P. J. Denning. Certification of programs for secure information flow. *Communications of ACM*, 20(7):504–513, 1977.

P. Flajolet and R. Sedgewick. *Analytic combinatorics*. Cambridge University Press, 2009.

J. Goubault-Larrecq and J.-P. Lachance. On the complexity of monitoring orchids signatures. In *International Conference on Runtime Verification*, pages 169–184, 2016.

J.-H. Jourdan, V. Laporte, S. Blazy, X. Leroy, and D. Pichardie. A formally-verified C static analyzer. In *ACM Symposium on Principles of Programming Languages*, pages 247–259, 2015.