**Clément Aubert**
Department of Computer Science
Appalachian State University
Boone, NC 28608
(828-)278-4620
(828-)262-2386
aubertc@appstate.edu
cs.appstate.edu/~aubertc/

**Research Statement**

My research focuses on giving tools to design better programming languages, to (i) ease the programmer's tasks, (ii) provide certifications, and (iii) design optimized compilers. To this end, I develop **semantics of programming languages**, that reveals the underlying mathematical structure of programs.

# Past Achievements

## Making the Mathematics Dynamic

In a first approximation, Mathematics objects are static (i.e., $2 + 3 = 5$ independently of any "execution") and Computer Science studies dynamic objects, programs that need to be evaluated to produce a result (i.e., $2 + 3$ *reduces* to 5). However, Mathematics can be made dynamics by considering as an object of study proofs and *rewriting of proofs*. Different programs can have the same input / output behavior, and, similarly, different proofs can prove the same statement under the same hypothesis; this analogy is made precise by the so-called "proof-as-program correspondence". Thanks to this isomorphism, abstract models of computation endowed with a precise cost model can be represented by proof systems. Starting from here, it is possible to impose limitations on the typing system, i.e., on the valid rules of the proof system, and to constrain the expressiveness of the corresponding class of programs.

Using a quantitative version of this fruitful correspondence between programs and proofs, I was able to prove, with Ph.D. students and recent Ph.D. laureates, that "balanced" logic programs are as expressive as 2-ways multi-head automata [3], and that further allowing unary logic program corresponded to the addition of a stack [2]. It is known that the former class of program captures Logarithmic Space, whereas the latter characterizes Polynomial Time, and this is where complexity theory kicks in. The limitations of those basics models of computation actually ensures that their corresponding proof systems will produce only program of such or such complexity. Thanks to Implicit Computational Complexity, you know *by design* that your programming language won't let you write programs that are going to need more than a given bound on space or time. By abstracting away implementation details, we managed to design high-level programming languages that are reliable and certified.

## Making the Programs Static

Semantics of programs also allows to "flatten" programs. Imagine a concurrent scenario (resources and data are shared) where programs can compute both forward and backward (i.e., we are in a reversible setting). Mathematics, and more precisely configuration structures, gives a tool to express all the possible scenario that could result from the interaction of multiple reversible programs. By writing down all the events that can occur and their connections when a program is confronted with a specific environment, you can decide whenever two programs have the same observational behavior: if the structures of the events provoked by a reversible program is the same as the structure given by another program, then we know that those two programs—no matter their actual content—are equivalent.

Actually, one can be finer than this: interesting relations on such structures determine a variety of quotients on programs [4, 1]. Whereas syntactical comparison of reversible and concurrent programs is nearly impossible and most of the time irrelevant, their semantics comparison gives instantly a reliable verdict. By tweaking what the environment can observe, one can model man-in-the-middle attacks, saturated channels, or add quantitative reasoning. By knowing whenever two part of programs are equivalent, and which events are going to occur, one can also cut dead branches in programs: this work leads to guarantees on the absence of deadlock and open the way to optimized compilation.

# Research Projects

The most theoretical aspects of my previous achievements reached an exciting stage of maturity. They can support grant proposals, be extended, move toward more practical realizations, but also be the starting point of thrilling students projects.

Each one of those lines support interesting and relevant extensions. For instance,

- We isolated conditions on logic programs and proof system that correspond to a particular complexity classes twice, and could illustrate the reproducibility and usability of our theory by isolating other complexity classes (Polynomial Space (**PSPACE**) seems reasonably accessible).
- Having a more fine-grained definition of the difference between environments and processes would provide more insights on equivalences on reversible and concurrent processes.

Those perspectives and the international networks that come with each one of them, could support timely and ambitious projects. For instance, the NSF 16565 solicitation seems accessible, since it targets young researchers willing to develop independent research and to integrate students. Regarding this aspect, each one of those lines could be turned into an object of study for motivated students:

- The correspondence between automata and logic programs offers a nice and clear set-up to understand what it means for a model of computation to simulate another. This would be a theoretical, but accessible, subject of research, that could lead to a publication in an international workshop.
- Basics knowledge in reversible and concurrent programming might soon become vital in the job market, and reversible imperative programming language such as Janus could be an interesting object of study, in complement of an initiation to multi-threading in Java for instance.

I am a passionate researcher that likes to share his enthusiasm and to work on multiple projects at the same time. I don't conceive my profession as being two-sided, and have always imagined my career as a blend of theoretical progresses and practical experiences, both shared with students.

# References


[1] **C. Aubert** and I. Cristescu. "Contextual equivalences in configuration structures and reversibility". In: *Journal of Logical and Algebraic Methods in Programming* 86.1 (2017), pp. 77–106. ISSN: 2352-2208. DOI: 10.1016/j.jlamp.2016.08.004.

[2] **C. Aubert**, M. Bagnol, and T. Seiller. "Unary Resolution: Characterizing Ptime". In: *Foundations of Software Science and Computation Structures (FOSSACS 2016)*. Vol. 9634. Lecture Notes in Computer Science. Springer, 2016, pp. 373–389. DOI: 10.1007/978-3-662-49630-5_22. Acceptance rate: 27.4%.

[3] **C. Aubert** and M. Bagnol. "Unification and Logarithmic Space". In: *Logical Methods in Computer Science, special issue of RTA/TLCA 2014* (2015).

[4] **C. Aubert** and I. Cristescu. "Reversible Barbed Congruence on Configuration Structures". In: *8th Interaction and Concurrency Experience (ICE 2015)*. Vol. 189. Electronic Proceedings in Theoretical Computer Science. 2015, pp. 68–95. DOI: 10.4204/EPTCS.189.7.