Semantics of Computational Effects

A summary for my dad

Jack Liell-Cock

My research topic is in the area of *semantics of computational effects*, which is in the wider field of *denotational semantics*. This is an approach to formalising programming languages by modelling them with mathematical objects, called denotations. It originated from the work of Dana Scott and Christopher Strachey at Oxford in the 1970s, where they modelled computer programs from input x to output y as mathematical functions $f : x \to y$.

However, standard mathematical functions are not sufficient to model computer programs because mathematical functions are pure in the sense that, given the same input, they will always return the same output. On the other hand, computer systems have a wider range of capabilities. During the execution of a program, computers may throw exceptions, invoke nondeterminism, ask for user input, print something to the screen, or access and manipulate memory. These abilities may cause a program to produce a different result on the same inputs. Such operations are called *effectful* because they induce auxiliary effects on the system beyond the input and output of the computer program. The corresponding sources of impurity, such as memory access and nondeterminism, are called *computational effects*.

Modelling effectful programs via denotational semantics began in an ad hoc manner. For example, nondeterminism was given semantics using a powerdomain construction [10]. Then, in his seminal work on notions of computation [7, 8], Moggi gave a unified account for computational effects from a category-theoretic perspective, by considering the output of a computation to be the resulting value *along with* the effects that could happen before it is returned. Rather than considering a program as a function $f : x \to y$, it is considered a function $f : x \to T(y)$. Here, $T(y)$ is the set of all possible computations that could happen before the result of a pure value y. Moggi outlined the properties of T to generalise core programming paradigms such as compositionality. We call a T with such properties a *monad*.

I almost exclusively work with these things called *monads*. There are many flavours of and perspectives on these mathematical objects. One view takes an algebraic perspective by giving the effectful operations (such as a computer function mimicking a fair coin flip) a more primitive role [11, 9]. It aims to unify reasoning about effectful computation by describing them using algebra, so this field of research adopted the name *algebraic effects*. Programming language development is streamlined by the algebraic effect framework, which is witnessed by their adoption in a range of programming languages [1, 6, 2, 4, 12].

One current limitation with the semantics of computational effects is that it struggles to capture the dynamic creation of effectful operations. For example, it is easy to reason about a set of memory locations to which a program may read or write. However, in many computer programs, one can dynamically create new memory locations to store additional data, which complicates things. An extension of algebraic theories (and monads) that can handle such dynamically created effects was introduced by Sam Staton (my supervisor) as parameterised algebraic theories [15, 14]. Operations may create or use an abstract parameter type, often associated with a resource. In our example above, the parameter would be the memory locations.

This extension has been used to model more complex computational effects, like restriction [15], quantum information [13] and scoped effects [5].

Another area of research in the semantics of computational effects is *graded monads*. The idea behind grading is that it allows one to stratify and annotate effectful computations to reason more fine-grainedly about what effects are taking place. For example, we can use gradings to talk about exactly which memory locations were read from or written to, rather than just the occurrence of memory manipulation. Sam and I recently published a paper on graded monads at POPL 2025 [3]. It used this grading technique to integrate different types of uncertainty into a probabilistic programming language. In the resulting language, one could talk about both aleatory uncertainty (i.e. from inherent randomness like rolling a dice) and Knightian uncertainty (i.e. from a lack of information about the world, like the existence of aliens).

Overall, there are many ways in which the semantics of computational effects are beneficial in developing programming languages. Firstly, interfaces and modules are typically a compile-time choice, but once the landscape of possible implementations and interactions is understood, it is profitable to use monads to make this a more dynamic, run-time choice. Secondly, when particular implementations of an interface are of interest, it is worthwhile to examine their properties. The framework of algebraic effects focuses on the algebraic interface, abstracting away the intricacies of the host language. This allows us to answer questions like when two different call sequences to an API have the same effect. Hence, we may derive new, alternative ways to axiomatise effects. This is useful, for example, to verify the correctness of programs in safety-critical situations.

References

[1] A. Bauer and M. Pretnar. "Programming with algebraic effects and handlers". In: *Journal of Logical and Algebraic Methods in Programming*. Special Issue: The 23rd Nordic Workshop on Programming Theory (NWPT 2011) 84.1 (Jan. 2015), pp. 108–123. DOI: 10.1016/j.jlamp.2014.02.001.

[2] D. Leijen. "Koka: Programming with Row Polymorphic Effect Types". In: *Electronic proceedings in theoretical computer science* 153.MSFP (2014). Place: Ithaca Publisher: Cornell University Library, arXiv.org, pp. 100–126. DOI: 10.4204/EPTCS.153.8.

[3] J. Liell-Cock and S. Staton. "Compositional Imprecise Probability: A Solution from Graded Monads and Markov Categories". In: *Proc. ACM Program. Lang.* 9.POPL (Jan. 2025), 54:1596–54:1626. DOI: 10.1145/3704890.

[4] S. Lindley and J. Cheney. "Row-based effect types for database integration". In: *Proceedings of the 8th ACM SIGPLAN workshop on Types in language design and implementation*. TLDI '12. New York, NY, USA: Association for Computing Machinery, Jan. 2012, pp. 91–102. DOI: 10.1145/2103786.2103798.

[5] S. Lindley, C. Matache, S. Moss, S. Staton, N. Wu, and Z. Yang. *Scoped Effects as Parameterized Algebraic Theories*. arXiv:2402.03103 [cs, math]. Feb. 2024. DOI: 10.48550/arXiv.2402.03103.

[6] S. Lindley, C. McBride, and C. McLaughlin. "Do be do be do". In: *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*. POPL '17. New York, NY, USA: Association for Computing Machinery, Jan. 2017, pp. 500–514. DOI: 10.1145/3009837.3009897.

[7] E. Moggi. "Computational lambda-calculus and monads". In: *[1989] Proceedings. Fourth Annual Symposium on Logic in Computer Science*. Pacific Grove, CA, USA: IEEE Comput. Soc. Press, 1989, pp. 14–23. DOI: 10.1109/LICS.1989.39155.

[8] E. Moggi. "Notions of computation and monads". In: *Information and computation* 93.1 (1991). Place: San Diego, CA Publisher: Elsevier Inc, pp. 55–92. DOI: 10.1016/0890-5401(91)90052-4.

[9] G. Plotkin and J. Power. "Adequacy for Algebraic Effects". In: *Foundations of Software Science and Computation Structures*. Ed. by F. Honsell and M. Miculan. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2001, pp. 1–24. DOI: 10.1007/3-540-45315-6_1.

[10] G. D. Plotkin. "A Powerdomain Construction". In: *SIAM journal on computing* 5.3 (1976). Place: Philadelphia Publisher: Society for Industrial and Applied Mathematics, pp. 452–487. DOI: 10.1137/0205035.

[11] G. D. Plotkin and J. Power. "Notions of Computation Determine Monads". In: *Foundations of Software Science and Computation Structures*. Lecture Notes in Computer Science. ISSN: 0302-9743. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 342–356. DOI: 10.1007/3-540-45931-6_24.

[12] K. C. Sivaramakrishnan, S. Dolan, L. White, T. Kelly, S. Jaffer, and A. Madhavapeddy. "Retrofitting effect handlers onto OCaml". In: *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*. PLDI 2021. New York, NY, USA: Association for Computing Machinery, June 2021, pp. 206–221. DOI: 10.1145/3453483.3454039.

[13] S. Staton. "Algebraic Effects, Linearity, and Quantum Programming Languages". In: *ACM SIGPLAN Notices* 50.1 (Jan. 2015), pp. 395–406. DOI: 10.1145/2775051.2676999.

[14] S. Staton. "An Algebraic Presentation of Predicate Logic: Extended Abstract". In: *Foundations of Software Science and Computation Structures*. Lecture Notes in Computer Science. ISSN: 0302-9743. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 401–417. DOI: 10.1007/978-3-642-37075-5_26.

[15] S. Staton. "Instances of Computational Effects: An Algebraic Perspective". In: *Proceedings of the 2013 28th Annual ACM/IEEE Symposium on Logic in Computer Science*. LICS '13. USA: IEEE Computer Society, June 2013, p. 519.