The project was a success. My success criterion was to implement a syntax-level interpreter for Eff, to design a low-level byte code for Eff, to implement a compiler which can transform Eff programs into a linearised representation and finally to implement a virtual machine which is capable of interpreting the designed byte code. Furthermore, the above components had to be reasonably efficient when compared to existing solutions.

The work described above was carried out and all components were successfully implemented. Both my CEK interpreter and my SHADE VM outperformed Eff on various benchmarks and the performance of SHADE VM was often comparable to that of Multicore OCaml.

Travel Diary This dissertation is actually a story about a journey which starts in the theoretical valleys of Mathematics, leads through the more practical (but still quite theoretical) areas of Computer Science and finally ends in the realms of practical Systems Programming.

This work was partly motivated by the desire to explore how algebraic effect handlers could be used in systems programming. The use of algebraic effect handlers is appealing as they provide a convenient abstraction for the handling of side effects functionally. However, efficient runtime systems often work with low-level bytecodes rather than with syntax-level representations of programs. Designing appropriate bytecodes and ways to realise algebraic effect handlers closer to the hardware is necessary if we are ever going to use them in the real world. The existence of Multicore OCaml and SHADE VM proves that this is possible. To show a real world systems programming application I exhibited a simple concurrent system (a toy webserver) which was executed in the runtime system I designed.

1 Further work

Optimisations The fact that there are algebraic theories behind algebraic effects and their handlers allows us to prove certain compiler transformations correct by equational reasoning. This gives us the opportunity to take a set of effect equations (for instance, the equation $\texttt{assign}(x, y); \texttt{assign}(x, z) = \texttt{assign}(x, z)$ would show that the first `assign` can be optimised away) and generate new optimisations (perhaps entirely new ones no-one has thought about yet or ones that are dependent on the semantics of some effects).

Type and effect systems [**?**] allow for even more optimisations and the opportunity to parallelise independent parts of programs. As hardware is getting increasingly parallel such optimisations can help us to get the most out of our multicore machines.

Open source The project is packaged up as an OPAM package and I intend to make it available on Github for other programming language enthusiasts.

Part II project My implementation was concerned with only the core features of Eff. There would be countless ways to improve the compiler or the virtual machine. The rules of SHADE VM are simple enough that a byte code interpreter for SHADEcode should not be too hard to implement in a low-level language such as C. However, this would have to involve designing a custom garbage collector for SHADE which might prove interesting.

Masters or PhD As the size of continuations is not too big it might make sense to think about these control operators in the context of distributed systems too. For instance, imagine the following situation where the communication overhead in the network might be reduced.

Alice and Bob are communicating. Alice sends a message M_1 to Bob and Bob replies with M_2. Based on the contents of M_2 Alice performs some kind of computation c and replies with M_3. If c can be realised as a continuation resumed with M_2 as its argument and given that shadows/continuations can be serialised using a convenient byte code representation, would it make sense to *package up a decision as a continuation* and attach it to the original message M_1? This would eliminate the need to exchange M_2 and M_3 over the network as Bob could resume the continuation using M_1 and obtain the result *locally*.

2 Self-reflection

If I were to start the project again I would try to be less ambitious. I vastly underestimated the intellectual challenge behind continuations, handlers and control operators.

At the beginning of the project I got a bit carried away with syntactic sugar and trying to implement a lot of features of the language properly. This proved to be counterproductive and I had to abandon these efforts. If I had to do this again I would make the development even more iterative than it was: I would design a smaller minimal viable product first, build a very minimal but working (and tested!) pipeline and then iterate more on improvements. This was my strategy when developing, but I could do it better the next time.

Closing thoughts Furthermore, I think there is a lack of elementary material concerning this topic on the web. I am hoping that my Preparation chapter and this dissertation as a whole will in some ways help popularise algebraic effects and their handlers and that readers of this document will find them less cryptic than I first did.