Problem statement

Albert ten Napel

1 Problem statement

In the real world programs are effectful. They have to interact with the outside world, manage state, throw exceptions and more. Purely functional programming is often said to be easier to reason about than imperative programming. This is partly because in purely functional programming effects are usually explicit. Many functional languages use monads to encapsulate effects. Monads make explicit what effects functions use and aid in reasoning. Unfortunately monads are hard to compose, requiring monad transformer stacks that are hard to understand and manage.
Algebraic effects and handlers is an alternative approach to effectful programming. Using algebraic effects effects one can easily implement effects such as non-determinism, mutable state, exceptions, backtracking, and cooperative multi-threading. These effects can be composed and ordered in any way without any extra effort required from the user.
Unfortunately not all effectful programs can be easily expressed using algebraic effects. Handlers will handle all operations of a certain effects making state threads using multiple mutable references impossible to define. These so-called dynamic effects can be implemented by adding dynamic instances to the system. In such a system we can dynamically create instances of effects, distinct from other instances. Handlers then only handle a specific instance, giving fine-grained control over effects.
There have been multiple type-and-effects systems proposed for algebraic effects and handler but unfortunately none of them account for dynamic instances. Programming with them can still result in runtime crashes because of unhandled operations.
We define a type-and-effect system for algebraic effects and handlers in the presence of dynamic instances. The system is sound with respect to progress and preservation and will ensure that a typed program with an empty effect

set will have no unhandled operations. We show that functional state threads can be expressed in this system. We also formalize the system in Coq and prove type soundness.

2 Contributions

- **We define a type-and-effect system for algebraic effects and handlers in the presence of dynamic instances.** The system is sound with respect to progress and preservation and will ensure that a typed program with an empty effect set will have no unhandled operations. Furthermore the system is a simple extension of a system with algebraic effects without instances.
- **We show how to implement state threads in this system.** Using our sytem we implement state threads similar to Haskell's ST-monad. We show that references cannot escape their scope and that we cannot use a variable from one state thread in another state thread.
- **We formally prove type-soundness of the system.** We prove that typed programs with empty effects sets will not have unhandled operations and so do not get stuck. We explain what techniques we used to implement the proofs.
- **We formalize the system in Coq.**

3 Milestones

- (1w) Formal system: syntax
- (1w) Formal system: type system
- (1w) Formal system: Semantics
- (1w) Coq: syntax, typing rules and semantics
- (1m) Coq: progress and preservation proofs
- (1w) Thesis chapter: Introduction (problem statement, proposed solution)

- (1m) Thesis chapter: Background (effects, algebraic effects and handlers, instances)
- (1m) Thesis chapter: Our system
- (2w) Thesis chapter: Examples (encapsulated effects, polymorphic heaps)
- (2w) Thesis chapter: Formalization
- (1w) Thesis chapter: Evaluation and discussion
- (2w) Thesis chapter: Related work
- (2w) Thesis chapter: Conclusion and future work (polymorphic operations, indexed effects)

4 Timeplan

Date	**Milestones**
12 November 2018	Syntax, part of background
26 November 2018	Type system, more of background
10 December 2018	Semantics, background done
7 January 2019	Coq syntax, typing rules, semantics, part of System chapter
21 January 2019	part of Coq proofs, course practical exercise
28 January 2019	Exam course
1 February 2019	course practical exercise deadline
11 February 2018	part of Coq proofs, part of System chapter
25 February 2019	Coq proofs done, System chapter done
11 March 2019	Examples done
25 March 2019	Formalization chapter done
8 April 2019	Related work done
22 April 2019	Introduction and Evaluation done
6 May 2019	Improvements, thesis done