Contextual Modal Types for Algebraic Effects and Handlers

Nikita Zyuzin and Aleksandar Nanevski

IMDEA Software Institute

Algebraic effects and handlers [2, 9] provide a modular and compositional description for computational effects. In this view, *only* a designated set of operations invokes side effects during evaluation of a term. Moreover, the use of such operations can be eliminated by a handler that provides definitions for these operations.

In this work, we propose that algebraic effects and handlers can be naturally typed using a variation of Contextual Modal Type Theory [5]. CMTT distinguishes between contextual modal variables $u :: A[\Delta]$, and normal variables $x : A$, having two separate contexts Θ and Γ for them respectively. The type $[\Delta]A$ classifies terms that depend on normal variables in the context Δ, but no other normal variables. The typing rules of CMTT are:

$$\square I\,\frac{\Theta;\Delta \vdash s : A}{\Theta;\Gamma \vdash \texttt{box}\ \Delta.\ s : [\Delta]A} \qquad \square E\,\frac{\Theta;\Gamma \vdash s : [\Delta]A \qquad \Theta, u :: A[\Delta];\Gamma \vdash t : B}{\Theta;\Gamma \vdash \texttt{let box}\ u\ \texttt{=}\ s\ \texttt{in}\ t : B}$$

$$\text{ctxhyp}\,\frac{(u :: A[\Delta]) \in \Theta \qquad \Theta;\Gamma \vdash \sigma : \Delta}{\Theta;\Gamma \vdash \texttt{handle}\ u\ \texttt{with}\ \sigma : A} \qquad \text{explsub}\,\frac{\Theta;\Gamma \vdash s_i : A_i \qquad i = \overline{1,n}}{\Theta;\Gamma \vdash \langle s_i/x_i, \dots\rangle : (x_i : A_i, \dots)}$$

The $\square I$ rule introduces contextual modality; the term under the box constructor may *only* use normal variables bound by the context Δ, but not by Γ. The rules $\square E$ and `ctxhyp` allow to use a boxed term, by binding it to a variable in the context ($\square E$), and then using this variable (`ctxhyp`). One can use contextual variables by applying an explicit substitution (`explsub`) that replaces all `box`-bound variables with appropriate terms.

We propose to view the context Δ as the algebraic theory of computations of type $[\Delta]A$. Thus the calculus immediately provides a type-and-effect system. Consider the program P that uses algebraic effects from theories $\texttt{St} \mathrel{\widehat{=}} \mathit{get} : \mathit{unit} \to \mathit{nat}, \mathit{put} : \mathit{nat} \to \mathit{unit}$ of state and $\texttt{Ex} \mathrel{\widehat{=}} \mathit{raise} : \mathit{unit} \to \bot$ of exceptions:

```
P ≙ box St, Ex. let x = get() in
                    if x = 42 then raise() else put(x + 1)
```

P has the type $[\texttt{St}, \texttt{Ex}]\texttt{unit}$, signifying that P can cause effects from `St` and `Ex`, but no others. To run P, we must provide an explicit substitution σ that defines all the operations from `St` and `Ex`, and then execute `let box u = P in handle u with` σ.

Explicit substitution is thus similar to handling of algebraic effects, which is why we write `handle` to denote applying it. Unfortunately, the similarity is not strong enough, as in CMTT we cannot define a handler neither for state nor for exceptions in this example. The problem arises because we want to use the operations *get*, *set* and *raise* as generic effects [8] and pass no continuation arguments that would allow state manipulation in substitution clauses.

In this paper we modify CMTT to adapt its notion of handling to programming with algebraic effects, preserving the type-and-effect discipline of contextual modal types. Specifically: (1) We use contextual modal types to denote algebraic theories of effectful computations; (2) We adopt the judgment for monadic computation $e \div A$ from [6], whose terms make the sequencing of effects explicit. Only terms of this judgment can be boxed; (3) When used left of $\vdash$ to declare variables, we generalize the judgment to $c \div A \Rightarrow B$. The generalized judgement denotes computations *hypothetical* in A (thus, it classifies effectful functions), and gives us suitable typing for the effect operators; (4) We extend the notion of applying explicit substitution in CMTT to handling of algebraic effects. Most important typing rules of our system are:

$$\text{cnthyp}\frac{(k \mathrel{\dot{\sim}} A \otimes B \rightsquigarrow C) \in \Gamma \quad \Theta; \Gamma \vdash s : A \quad \Theta; \Gamma \vdash t : B}{\Theta; \Gamma \vdash \texttt{throw}\ k\ s\ t \div C} \qquad \text{ophyp}\frac{\Theta; \Gamma, op \div A \Rightarrow B \vdash s : A}{\Theta; \Gamma, op \div A \Rightarrow B \vdash op\ s \div B}$$

$$\text{handler}\frac{\Theta; \Gamma, z : D, x : A_i, k \mathrel{\dot{\sim}} B_i \otimes D \rightsquigarrow C \vdash e_i \div C \qquad i = \overline{1, n}}{\Theta; \Gamma \vdash z : D.\{op_i(x, k) \Rightarrow e_i, \dots\} \div D.[op_i \div A_i \Rightarrow B_i, \dots] \triangleright C}$$

$$\text{ctxhyp}\frac{(u :: A[\Delta]) \in \Theta \quad \Theta; \Gamma \vdash h \div B.[\Delta] \triangleright C \quad \Theta; \Gamma \vdash t : B \quad \Theta; \Gamma, x : A, z' : B \vdash e \div C}{\Theta; \Gamma \vdash \texttt{handle}\ u\ \texttt{with}\ h\ \texttt{from}\ t\ \texttt{to}\ x.z'.e \div C}$$

In more detail, in $e \div A$, e is an effectful computation that runs and produces a value of type A. This judgement forces computations to be written as a sequence of `let` forms. The judgments $e \div A$ and $c \div A \Rightarrow B$ are related by the `effhyp` rule. Rule `conthyp` provides use for *continuation* variables also typed by a hypothetical judgement.

We use the rules $\Box I$ and $\Box E$ from CMTT, adapting them to the judgement for computations. The `handler` rule specifies handling of each operation $op_i \div A_i \Rightarrow B_i$, with corresponding terms e_i. We type check these terms in the extended context, where $z : D$ is a handler-bound variable shared by all operations, x is the operation's op_i parameter, and $k \mathrel{\dot{\sim}} B_i \otimes D \rightsquigarrow C$ is the continuation for op_i. The continuation takes op_i's output and a new value for the shared variable, returning a value of the return type of the handler.

Finally, the `ctxhyp` rule types handling of computations, and subsumes the old rule from CMTT. Here, handler h serves as the explicit substitution: h substitutes all the free variables from the algebra Δ. `from` t specifies initial value t for the handler-bound variable. `to` $x.z'.e$ binds the final value and shared variable after handling to respectively x and z' in e.

With our new typing rules, we can handle the program P, now setting `St` $\widehat{=}$ $get \div unit \Rightarrow nat, put \div nat \Rightarrow unit$ and `Ex` $\widehat{=}$ $raise \div unit \Rightarrow \bot$:

```
let box u = P in handle u with
  s: nat. { get(v, k) => throw k s s,
            put(v, k) => throw k v (),
            raise(v, k) => ret ((), s) }
from 0 to x. s'. ret (x, s')
```

This handler has the type $nat.[get \div unit \Rightarrow nat, put \div nat \Rightarrow unit, raise \div unit \Rightarrow \bot] \triangleright unit \times nat$. The handler-bound variable s preserves the state of the program between different invocations of get and put: the body of an operation receives current state in the context and specifies the resulting state when calling continuation. `from` 0 sets the initial state to 0 for the handled computation. Finally, we return the resulting value and state as a pair.

We are currently working on proving type soundness of the proposed calculus. In future, we plan to scale to dependent types, as context with dependent types will allow us to concisely specify algebraic theories *with equations between algebraic operations*. As CMTT has already been shown to support dependent types [5], we expect that our extension to algebraic effects will support them as well. Dependent types will also facilitate verification of programs with generic effects, and we plan to explore potential connections with separation logic. These will provide a different verification perspective, compared to other systems with algebraic effects and dependent types, e.g. [1]. We also plan to explore abstraction over handlers and contexts in the contextual types [4], to obtain abstraction over algebraic theories as an alternative to row polymorphism [3]. Additionally, our use of contextual types gives another perspective on scoping for algebraic effects and handlers [7, 10].

References

[1] Danel Ahman. Handling fibred algebraic effects. *Proc. ACM Program. Lang.*, 2(POPL), December 2017.

[2] Andrej Bauer. What is algebraic about algebraic effects and handlers? *arXiv preprint arXiv:1807.05923*, 2018.

[3] Dariusz Biernacki, Maciej Piróg, Piotr Polesiuk, and Filip Sieczkowski. Abstracting algebraic effects. *Proceedings of the ACM on Programming Languages*, 3(POPL):6, 2019.

[4] Andrew Cave and Brigitte Pientka. First-class substitutions in contextual type theory. In *Logical Frameworks & Meta-languages: Theory & Practice (LFMTP)*, pages 15–24, 2013.

[5] Aleksandar Nanevski, Frank Pfenning, and Brigitte Pientka. Contextual modal type theory. *ACM Transactions on Computational Logic (TOCL)*, 9(3):23, 2008.

[6] Frank Pfenning and Rowan Davies. A judgmental reconstruction of modal logic. *Mathematical structures in computer science*, 11(4):511–540, 2001.

[7] Maciej Piróg, Tom Schrijvers, Nicolas Wu, and Mauro Jaskelioff. Syntax and semantics for operations with scopes. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 809–818, 2018.

[8] Gordon Plotkin and John Power. Algebraic operations and generic effects. *Applied categorical structures*, 11(1):69–94, 2003.

[9] Matija Pretnar and Gordon D Plotkin. Handling algebraic effects. *Logical Methods in Computer Science*, 9, 2013.

[10] Nicolas Wu, Tom Schrijvers, and Ralf Hinze. Effect handlers in scope. In *Proceedings of the 2014 ACM SIGPLAN Symposium on Haskell*, pages 1–12, 2014.