Programming language techniques for proof assistants

Lecture 4
Variables as computational effects

Andrej Bauer
University of Ljubljana

International School on Logical Frameworks and Proof Systems Interoperability
Université Paris–Saclay, September 8–12, 2025

Overview

- Lecture 1: From declarative to algorithmic type theory
- Lecture 2: A monadic type checker
- Lecture 3: Holes and unification
- Lecture 4: Variables as computational effects
  - Algebraic operations and handlers
  - Variables as effects
  - A handler-based type checker

Lecture 4

Variables as computational effects

Before I forget…

We have postdoc positions in Ljubljana. Topics:

- Type theory
- AI and formalized math
- Constructive and synthetic mathematics
- Programming languages

Talk to me if you are interested.

A bird's-eye view of algebraic effects and handlers

- Algebraic theory: operations & equations.
- An n-ary operation $\mathsf{op} : C^n \to C$ on carrier C:
  - A constant $c \in C$ is a 0-ary operation.
  - Addition $+ : \mathbb{R}^2 \to \mathbb{R}$ is a 2-ary operation.
- And A-ary $\mathsf{op} : C^A \to C$:
  - $\int_0^1 : \mathbb{R}^{[0,1]} \to \mathbb{R}$ is an $[0, 1]$-ary operation on $\mathbb{R}$.
  - $\forall_A : \mathsf{Prop}^A \to A$ is an A-ary operation on Prop.
- P-parameterized operation: $\mathsf{op} : P \times C^n \to C$:
  - Scalar multiplication of vectors $\cdot : \mathbb{R} \times V^1 \to V$
  - Consing a list $\mathsf{cons} : P \times \mathsf{List}\, P \to \mathsf{List}\, P$
- A-ary P-parameterized operation on C:

  $$\mathsf{op} : P \times C^A \to C.$$

Example – state as an algebraic theory

- Fix a set of states S.
- Operations:
  - $\mathtt{get} : C^S \to C$
  - $\mathtt{put} : S \times C \to C$
- Equations:

$$
\begin{aligned}
\mathtt{get}(\lambda x.\,\mathtt{get}(\lambda y.\,\kappa)) &= \mathtt{get}(\lambda z.\,\kappa[z/x, z/y]) \\
\mathtt{get}(\lambda x.\,\mathtt{put}(x, \kappa)) &= \mathsf{get}(\lambda x.\,\kappa) \\
\mathsf{put}(x, \mathtt{get}(\lambda y.\,\kappa)) &= \mathsf{put}(x, \kappa[x/y]) \\
\mathtt{put}(x, \mathtt{put}(y, \kappa)) &= \mathsf{put}(y, \kappa)
\end{aligned}
$$

- Examples: I/O, exceptions, non-determinism, probabilistic computation, transactional memory, co-operative multi-threading, delimited continuations, ...
- Non-example: (undelimited) continuations

Algebraic handlers

- Math: a handler is a homomorphism from a free algebra.
- Algebraic handlers generalize exception handlers:
  - exception handler: intercept exception and do something,
  - algebraic handler: intercept *operation*, do something, possibly *resume* execution.

  "Algebraic operations are resumable exceptions."
- Programming tool:
  - modify effects: redirect output, log memory access, implement transactions, …
  - implement custom effects.

Example in OCaml 5: write-once state

```
type _ Effect.t +=
  | Get : unit -> int Effect.t
  | Put : int -> unit Effect.t

let get () = perform (Get ())
let put s = perform (Put s)

type mode = Initial | Modified
exception InvalidWrite

let with_state (s : int) (c : unit -> 'a) =
  let r = ref (Initial, s) in
  try
    c ()
  with
    | effect (Get ()), k -> continue k (snd !r)
    | effect (Put s), k ->
      (match !r with
       | (Initial, _) -> r := (Modified, s) ; continue k ()
       | (Modified, _) -> raise InvalidWrite)
```

Example in OCaml 5: write-once state (continued)

```
let eightyeight =
  with_state 42
    (fun () ->
      let a = get () in
      put (a + 4) ;
      a + get ())

 let problematic =
  with_state 42
    (fun () ->
      let a = get () in
      put (a + 4) ;
      if a * a > 666 then put 10 ;
      a + get ())
```

Variables as algebraic effects

- We remove the monad `Context.m` and write code in direct style.
- Algebraic operations for variables:

```
type _ Effect.t +=
 | LookupVar: TT.var -> (TT.tm option * TT.ty) Effect.t
 | LookupIdent: string -> TT.var option Effect.t
 | TopExtend: (string * TT.tm option * TT.ty) -> unit Effect.t
```

- Handlers for variables:
  - `Context.handle_context` – handle global variables at the top level
  - `Context.with_ident_var` – handle *one* local variable

A handler for global variables

Top level is wrapped by a handler for globally defined variables:

```
let handle_context c =
  let ctx = ref initial in
  try
    c ()
  with
   | effect (LookupVar v), k -> ...
   | effect (LookupIdent x), k -> ...
   | effect (TopExtend (x, def, ty)), k -> ...
```

This is just a standard state handler.

A handler for local variables

- Local variables are introduced by `let`, Π and λ.
- A handler for a single variable:

```
let with_ident_var x v ?def t c =
  try
    c ()
  with
  | effect (LookupIdent y), k when String.equal x y ->
     continue k (Some v)
  | effect (LookupVar w), k when Bindlib.eq_vars v w ->
     continue k (def, t)
```

- The handler intercepts operations pertaining *only* to `x` and `v`.
- Other variables are handled by outer handlers.
- Handlers are nested, as many as there are local variables.

Meta-variables as algebraic effects

- Algebraic operations for meta-variables:

```
type _ Effect.t +=
    | LookupMeta : TT.var -> (TT.tm option * TT.ty) Effect.t
    | FreshMeta_ : string * TT.ty_ -> TT.tm_ Effect.t
    | SetMeta_ : TT.var * TT.tm_ -> bool Effect.t
```

- `Context.handle_metas` – handles meta-variables:
    - the handler is wrapped around each top-level command,
    - consequently, each top-level command has its set of meta-variables
- Additionally, `Context.with_ident_var` intercepts `FreshMeta_` and `SetMeta_` to ensure correct interaction with the local variable.
- For details, see `lib/core/context.ml`.

Conclusion

- Monadic type checker:
  - A reader monad for variables
  - Monadic code helps reduce clutter
- Monadic type checker with holes:
  - A combination of reader and state monads
  - Monadic code does not help much with meta-variables
- Algebraic type checker:
  - Direct-style code
  - Cleaner implementation of meta-variables
  - Potential for new implementation techniques

Further reading

- Andrej Bauer: What is algebraic about algebraic effects and handlers? (lecture notes), arXiv:1807.05923, March 2019.
- Matija Pretnar: An Introduction to Algebraic Effects and Handlers (invited tutorial paper), Electronic Notes in Theoretical Computer Science, Volume 319, 21 December 2015, Pages 19–35.
- Bob Atkey: An Algebraic Approach to Typechecking and Elaboration, Scottish Programming Languages Seminar, February 2015.
- Conor McBride: An Effects-and-Handlers Implementation of Hindley-Milner Typechecking, 2024.