



A type system for dynamic instances

Delft University of Technology

Albert ten Napel

September 5, 2019

Outline

- 1 Effects
- 2 Algebraic effects and handlers
- 3 Miro

Effects

- Used everywhere in programming
- Examples: mutable state, input/output, reading/writing to files/channels, randomness, ...
- Makes reasoning, testing and debugging difficult

Example

```
guesses = 0

// guess : () -> Int
def guess():
  global guesses
  n = input("give a number: ")
  guesses += 1
  if n == "42":
    print("you guessed correctly!")
  else:
    print("wrong number")
  return guesses
```

Algebraic effects and handlers

- Approach to programming with effects
- Composable: easily combine different effects
- Type safe

Algebraic effect interfaces

Example

```
effect State {  
  get  : () -> Int  
  put  : Int -> ()  
}  
  
effect IO {  
  input : String -> String  
  print : String -> ()  
}
```

Using algebraic effects

Example

```
guess : () -> Int!{State, IO}
guess () =
  n <- #input("give a number: ");
  x <- #get();
  #put(x + 1);
  if n == "42" then
    #print("you guessed correctly!")
  else:
    #print("wrong number");
  guesses <- #get();
  return guesses
```

Handling algebraic effects

Example

```
handleGuessIO : Int!{State}
handleGuessIO =
  handle( guess() ) {
    input msg k -> k "42"
    print msg k -> k ()
    return x -> return x
  }
```


Shortcoming

- We can only manipulate a single global state cell
- How to express multiple mutable references?
- How to express opening and reading/writing to files/channels?

Dynamic effect instances

Example

```
r1 <- new State;  
r2 <- new State;  
handle#r1 (  
  x <- r1#get();  
  r2#put (x + 1)  
) { ... }
```

Escaping instances

Example

```
escape ref =  
  return \() -> ref#get ()  
  
escaped =  
  ref <- new State;  
  fn <- handle#ref (escape ref) { ... };  
  return fn
```

The problem

- Dynamic instances allows us to express more effects
- Dynamic instances are not type-safe
- How to ensure that all operations will be handled?

- Language with algebraic effects and handlers
- With a restricted form of dynamic instances
- We can define mutable references, dynamic opening of files/channels, ...
- Type safe: ensures that all operations will be handled

Miro - Creating instances

Example

```
effect Config {  
  get : () -> Int  
}  
  
makeConfig : forall s. Int -> (Inst s Config)!{s}  
makeConfig [s] v =  
  new Config@s {  
    get () k -> k v  
    return x -> return x  
    finally x -> return x  
  } as x in return x
```

Miro - Using and handling instances

Example

```
useConfig : Int
useConfig =
  runscope(myscope ->
    -- c : Inst myscope Config
    c <- makeConfig [myscope] 42;
    x <- c#get();
    return x)
```

Miro - Runscope typing rule

T-RUNSCOPE

$$\frac{\Delta, s_{var}; \Gamma \vdash c : \tau ! r \quad s_{var} \notin \tau}{\Delta; \Gamma \vdash \text{runscope}(s_{var} \rightarrow c) : \tau ! (r \setminus \{s_{var}\})}$$

How does Miro make instances safe?

- Effect scopes: instances belong to a specific effect scope
- Instances always have a handler associated with them
- All instances of an effect scope are handled all at once
- Check that an effect scope does not escape its runscope

Miro - Type safety

Theorem 4 (Type safety).

if $(\cdot; \cdot \vdash c : \tau ! \emptyset)$ and $(c \rightsquigarrow^* c')$ then $\text{value}(c')$ or $(\exists c''. c' \rightsquigarrow c'')$

Lemma 6 (Preservation).

if $(\Delta; \Gamma \vdash c : \underline{\tau})$ and $(c \rightsquigarrow c')$ then $(\Delta; \Gamma \vdash c' : \underline{\tau})$

Miro - Type safety issue

$\text{runscope}(s \rightarrow \text{new State}@s \{h; \text{finally } f \rightarrow f \ 0\} \text{ as } r \text{ in return } (\lambda_r.r' \leftarrow \text{return } r; \text{return } 42))$
 \rightsquigarrow (S-RUNSCOPE)

$\text{runscope}^s(\text{new State}@s \{h; \text{finally } f \rightarrow f \ 0\} \text{ as } r \text{ in return } (\lambda_r.r' \leftarrow \text{return } r; \text{return } 42))$
 \rightsquigarrow (S-RUNSCOPENEW)

$\text{runscope}^s(f \leftarrow \text{runinst}_{s,\text{State}}^l\{h\}(\text{return } (\lambda_r.r' \leftarrow \text{return inst}(l); \text{return } 42)); f \ 0)$
 \rightsquigarrow (S-RUNSCOPECONG, S-SEQ, S-RUNINSTRETURN)

$\text{runscope}^s(f \leftarrow \text{return } (\lambda st.\text{return } (\lambda_r.r' \leftarrow \text{return inst}(l); \text{return } 42)); f \ 0)$

Conclusion

- Algebraic effects are great, but can be improved upon
- Miro - safely combine algebraic effects and dynamic instances
- Proving type safety is difficult

More in the thesis

- More examples
- Formalization of algebraic effects and handlers with static instances
- Formalization of Miro, type system and a small-step operational semantics
- Mechanization of algebraic effects and handlers with static instances in Coq (with type safety proofs)
- Prototype implementation of Miro in Haskell

Questions

Any questions?