



# A type system for dynamic instances

Delft University of Technology

Albert ten Napel

September 4, 2019

# Outline

- 1 Effects
- 2 Algebraic effects and handlers
- 3 Miro
- 4 Syntax
- 5 Type safety
- 6 Conclusion
- 7 Questions

## 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 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 : (List Int)!{State}
handleGuessIO =
  handle( guess() ) {
    input msg k -> (k "13") ++ (k "42")
    print msg k -> k ()
    return x -> return [x]
  }
```

# Multiple state cells

## Example

```
effect State1 {  
  get1 : () -> Int  
  put1 : Int -> ()  
}
```

```
effect State2 {  
  get2 : () -> Int  
  put2 : Int -> ()  
}
```

# 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
```

# 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 -> 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 - Syntax

$$\begin{aligned}s &::= s_{var} \mid s_{loc} \\ \tau &::= \text{Inst } s \ \varepsilon \mid \tau \rightarrow \underline{\tau} \mid \forall s_{var}. \underline{\tau} \\ \underline{\tau} &::= \tau ! r \\ \nu &::= x, y, z, k \mid \text{inst}(l) \mid \lambda x. c \mid \Lambda s_{var}. c \\ c &::= \text{return } \nu \mid \nu \ \nu \mid x \leftarrow c; c \mid \nu \# \text{op}(\nu) \mid \nu [s] \\ &\quad \mid \text{new } \varepsilon @ s \{h; \text{finally } x \rightarrow c\} \text{ as } x \text{ in } c \\ &\quad \mid \text{runscope}(s_{var} \rightarrow c) \\ &\quad \mid \text{runscope}^{s_{loc}}(c) \\ &\quad \mid \text{runinst}_{s_{loc}, \varepsilon}^l \{h\}(c) \\ h &::= \text{op } x \ k \rightarrow c; h \mid \text{return } x \rightarrow c\end{aligned}$$

# Miro - Runscope typing rule

T-RUNSCOPE

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

# 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

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

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

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

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

# Conclusion

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



# Questions

Any questions?