# A type system for dynamic instances

Delft University of Technology

Albert ten Napel

September 4, 2019

# Outline

# Effects

## 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)
```