

A new polymorphic type system for Miking

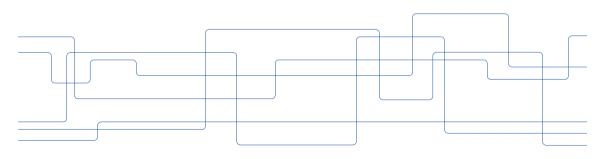
Anders Ågren Thuné (athune@kth.se)

digital futures

Miking Workshop 2023

2023-11-23

Type system key contributors (alphabetical order): David Broman, Elias Castegren, Viktor Palmkvist, Anders Ågren Thuné





- Background
- Limitations of the current type checker
- ► The new features
- Implementation
- Future challenges

- ▶ Miking was originally developed without a type checker.
- ► Can you spot the bug?

```
sem eval env =
| TmLet t ->
eval
   (insert t.ident (eval t.body) env)
   t.inexpr
```

```
let x = 5 in addi x = 1
```

- ▶ Miking was originally developed without a type checker.
- Can you spot the bug?

```
sem eval env =
| TmLet t ->
eval
   (insert t.ident (eval t.body) env)
   t.inexpr
```

```
let x = 5 in addi x = 1
```

Without type check: runtime error

- ▶ Miking was originally developed without a type checker.
- ► Can you spot the bug?

```
sem eval env =
| TmLet t ->
eval
   (insert t.ident (eval env t.body) env)
   t.inexpr
```

```
let x = 5 in addi x = 1
```

▶ Without type check: runtime error



▶ We want to catch the "obvious" errors *before* runtime.

sem eval : Env -> Expr -> Value

▶ We want to catch the "obvious" errors *before* runtime.

Type Checking

▶ We want to catch the "obvious" errors *before* runtime.

```
sem eval : Env -> Expr -> Value
```

▶ Development started in September 2021 and type checking was enabled for mi universally in June 2022. Based on FreezeML ¹ for first-class polymorphism.

¹Emrich et al., PLDI '20



Limitations of the Type Checker

► Is this code okay?

```
type Expr
con TmInt : Int -> Expr
let getInt = lam x.
  match x with TmInt i then i
  else never
getInt (TmInt 0)
```

Limitations of the Type Checker

► Is this code okay?

```
type Expr
con TmInt : Int -> Expr
let getInt = lam x.
  match x with TmInt i then i
  else never
getInt (TmInt 0)
```

What about now?

```
con TmString : String -> Expr
getInt (TmString "hello")
```



▶ We annotate the type with a set of constructors

```
getInt : Expr{TmInt} -> Int
```

We annotate the type with a set of constructors

```
getInt : Expr{TmInt} -> Int
```

Now we can differentiate different versions of a type



What if any constructors are okay?

```
type Foo
con F1 : Int -> Foo
con F2 : Foo -> Foo

let f = lam x.
  match x with F2 _ then x
  else F1 0
```

▶ What if any constructors are okay?

```
type Foo
con F1 : Int -> Foo
con F2 : Foo -> Foo

let f = lam x.
  match x with F2 _ then x
  else F1 0
```

► We use polymorphism to express *open types*

```
f : Foo{a} -> Foo{a}
    where a :: {F1,F2}
```

A new formalization of Miking's core system

```
\begin{array}{lll} \tau & ::= & \alpha \mid \tau_1 \rightarrow \tau_2 \mid \forall \alpha :: \kappa. \ \tau \mid \tau_1 \times \tau_2 \mid \tau_1 \leadsto^{\overline{p}} \tau_2 \mid \tau. T \mid \delta & \text{(Types)} \\ \kappa & ::= & \star \mid \delta & \text{(Kinds)} \\ \delta & ::= & \langle T_1 : \overline{\mathsf{K}}_1, \dots, T_n : \overline{\mathsf{K}}_n \rangle & \text{(Constructor types)} \\ e & ::= & x \mid \lambda x : \tau. \ e \mid e_1 \ e_2 \mid \Lambda \alpha :: \kappa. \ e \mid e[\tau] & \text{(Expressions)} \\ & \mid & \text{fix} \lambda^{\tau_1 \rightarrow \tau_2} \ f \ x. \ e \mid (e_1, e_2) \mid \pi_1 \ e \mid \pi_2 \ e \\ & \mid & \mathsf{K} \ [\tau] \ e \mid & \text{match } e \ \text{with } p \ \text{then } e_1 \ \text{else } e_2 \mid \text{never}^{\tau} \\ & \mid & \text{type } T \ \text{in } e \mid & \text{con } \mathsf{K} : (X :: \delta). \ \tau \rightarrow T \ \text{in } e \\ & \mid & \text{sem}^{\tau} \ \{p \rightarrow e\} \mid e_1 \oplus e_2 \mid e_1 \bullet e_2 \end{array}
```

Extension with constructor types and a new exhaustiveness checker

```
sem typeCheckExpr env =
| TmNever t ->
match matchesPossible env with None () then
   TmNever {t with ty = newpolyvar env.currentLvl t.info}
else ...
errorSingle [t.info] msg
```

Implementation in Miking

Extension with constructor types and a new exhaustiveness checker

```
sem typeCheckExpr env =
| TmNever t ->
match matchesPossible env with None () then
   TmNever {t with ty = newpolyvar env.currentLvl t.info}
else ...
errorSingle [t.info] msg
```

Language fragments for extensible and reusable code



▶ Better error messages and editor support.



- Better error messages and editor support.
- Expanded features (GADTs, gradual typing, ...) and formalization.



- Better error messages and editor support.
- Expanded features (GADTs, gradual typing, ...) and formalization.
- ► Error messages at the level of MLang.



- ► The ML-style type checker offers easy and expressive typing for MExpr. You can try it out today!
- Constructor types and exhaustiveness checker coming soonTM.
- Outlook
 - Better programmer support.
 - Extended features.