Algebraic Effects and Types

as First-Class Features in the Fuzion Language

Fridtjof Siebert
Tokiwa Software GmbH

FOSDEM, 4. Feb 2023, Brussels
Who is this guy?

Fridtjof Siebert

Email: siebert@tokiwa.software
github: fridis
twitter: @fridi_s

‘90-‘94 AmigaOberon, AMOK PD
‘97 FEC Eiffel Sparc / Solaris
‘98-‘99 OSF: TurboJ Java Compiler
‘00-‘01 PhD on real-time GC
‘02-‘19 JamaicaVM real-time JVM based on CLASSSPATH / OpenJDK,

VeriFlux static analysis tool
‘20-... Fuzion
‘21-... Tokiwa Software

FOSDEM’23: Algebraic Effects and Types in Fuzion
Motivation: Fuzion Language

Many languages overloaded with concepts like classes, methods, interfaces, constructors, traits, records, structs, packages, values, ...

➡ Fuzion has one concept: a feature

Today’s compilers and tools are more powerful

➡ Tools make better decisions

Systems are safety-critical

➡ we need to ensure correctness
Fuzion Resources

Fuzion available

sources: github.com/tokiwa-software/fuzion
Fuzion Resources

Fuzion available

➡ sources: github.com/tokiwa-software/fuzion

➡ Website: flang.dev
  • tutorial
  • design
  • examples
  • ...
Backing Company

→ supports development of Fuzion
→ currently four employees
→ hiring
→ searching for funding
This Talk

Complementarity of Effects and Types

- Algebraic Effects for Fuzion
- Types as first-class features
- Types used to name Effects
Fuzion Effects

Fuzion Features are pure functions

- no mutation of data, no side-effects

Effects are used to model non-functional aspects

- state changes
- I/O
- thread communication
- exceptions
Algebraic Effects

Definition

- an algebraic effect is a set of operations
  - read, get_time, panic, log, ...
  - operations often model a non-functional effect
- operations may resume or abort
- an effect’s operations may be implemented by different handlers
- to execute code that uses an effect, a corresponding handler must be installed
Fuzion Effects

Static analysis verifies effects

→ Static analysis determines all effects
→ library code must list all effects
→ unexpected effects are a compile-time error
Fuzion Effects Example

Hello World:

```scala
hello_world ! io.out ⇒
  say "hello world!"

hello_world
```
Fuzion Effects Example

Hello World:

```fuzion
hello_world ! io.out ⇒
say "hello world!"

hello_world
```

> fz hw.fz
Fuzion Effects Example

Hello World:

```fz
hello_world ! io.out ⇒
say "hello world!"
```

```bash
> fz hw.fz
hello world!
> 
```
Fuzion Effects Example

Hello World:

```fz
hello_world ! io.out ⇒
  say "hello world!"

hello_world
```

> fz hw.fz
hello world!
> fz -effects hw.fz
Fuzion Effects Example

Hello World:

```fz
hello_world ! io.out ⇒
say "hello world!"
```

```
> fz hw.fz
hello world!
> fz -effects hw.fz
io.out
>
```
Fuzion Effects Example

Hello World:

```fuzion
hello_world ! io.out ⇒
say "hello world!"

my_handler : io.Can_Print is
  print(s Any) unit is
    io.err.print (($s).replace "!" "!!!11!")

io.out my_handler ()⇒hello_world
```

FOSDEM'23: Algebraic Effects and Types in Fuzion
Fuzion Effects Example

Hello World:

```fuzion
hello_world ! io.out ⇒
say "hello world!"

my_handler : io.Can_Print is
  print(s Any) unit is
    io.err.print (($s).replace "!" "!!!11!"

io.out my_handler ()→hello_world
```

> fz hw.fz
hello world!!!11!
>
Fuzion Effects Example

Hello World:

```plaintext
hello_world ! io.out ⇒
say "hello world!"

my_handler : io.Can_Print is
  print(s Any) unit is
    io.err.print (($s).replace "!" "!!!11!")

io.out my_handler () → hello_world
```

> fz hw.fz
hello world!!!11!
> fz -effects hw.fz
io.err
>
Types as First-Class Features
Types as First-Class Features

Generics in Java

\[
<T> \text{ void show\_number}(T a) \\
\{ \\
\quad \text{System.out.println}("a is " + a); \\
\}
\]
Types as First-Class Features

Type parameters in Fuzion

```fuzion
show_number(T type, a T) ⇒ say "a is $a"
```

Generics in Java

```java
<T> void show_number(T a) {
    System.out.println("a is " + a);
}
```
Types as First-Class Features

Type parameters in Fuzion

```haskell
show_number(T type,
a T) ⇒
say "a is $a"
```
Types as First-Class Features

Type parameters in Fuzion

```haskell
show_number(T type, a T) ⇒
  say "a is $a"

show_number i32 1234
show_number f64 3.14
```

FOSDEM’23: Algebraic Effects and Types in Fuzion
Types as First-Class Features

Type parameters in Fuzion

```fz
show_number(T type, a T) =>
say "a is $a"
```

```fz
show_number i32 1234
show_number f64 3.14
```
Types as First-Class Features

Type parameters in Fuzion

```fz
show_number(T type,
    a T) ⇒
    say "a is $a"

show_number i32 1234
show_number f64 3.14
```

> fz types.fz
a is 1234
a is 3.14
>

FOSDEM'23: Algebraic Effects and Types in Fuzion
Types as First-Class Features

Type inference

```fz
show_number(T type, a T) ⇒
say "a is $a"
```

```fz
show_number i32 1234
show_number f64 3.14
```

> fz types.fz
a is 1234
a is 3.14
>
Types as First-Class Features

Type inference

```haskell
show_number(T type, a T) =>
say "a is $a"
```

- show_number 1234
- show_number 3.14
Types as First-Class Features

Type constraints

```haskell
show_number(T type : numeric T, a T) ⇒
  say "a is $a"
```

```
show_number 1234
show_number 3.14
```
Types as First-Class Features

Type constraints

```haskell
show_number(T type : numeric T, 
a T) ⇒
say "a is $a, twice is \{a+a\}"
```

show_number 1234
show_number 3.14
Types as First-Class Features

Type constraints

```fz
show_number(T type : numeric T,
            a T) ⇒
        say "a is \$a, twice is \{a+a}\"

show_number 1234
show_number 3.14
```

```bash
> fz types.fz
a is 1234, twice is 2468
a is 3.14, twice is 6.28
> 
```
Types as First-Class Features

Type values

```haskell
show_number(T type : numeric T,
            a T) =>
    say "a is $a of type {T.name}"
```

show_number 1234
show_number 3.14
Types as First-Class Features

Type values

```fz
show_number(T type : numeric T, a T) ⇒
    say "a is $a of type \{T.name\}"
```

```bash
> fz types.fz
a is 1234 of type i32
a is 3.14 of type f64
>```

```fz
show_number 1234
show_number 3.14
```
Types as First-Class Features

Type with user defined features

\[
\text{sum}_\text{of}(T \text{ type : numeric } T, \\
\text{l list } T) \Rightarrow
\]
Types as First-Class Features

Type with user defined features

```plaintext
sum_of(T type : numeric T,
       l list T          ) =>
  l ? nil    =>
  | c Cons  =>
```
Types as First-Class Features

Type with user defined features

```
sum_of(T type : numeric T,
    l list T            ) ⇒
  l ? nil       ⇒
    | c Cons ⇒ c.head + sum_of c.tail
```
Types as First-Class Features

Type with user defined features

\[
\text{sum\_of}(T \text{ type} : \text{numeric } T, \\
\quad \text{l list } T) \Rightarrow \\
\quad \text{l } ? \text{ nil } \Rightarrow \\
\quad \mid \text{c Cons } \Rightarrow \text{c.head + sum\_of c.tail}
\]
Types as First-Class Features

Type with user defined features

```
sum_of(T type : numeric T,
       l list T ) ⇒
  l ? nil    ⇒ T.zero
  | c Cons ⇒ c.head + sum_of c.tail
```
Types as First-Class Features

Type with user defined features

```haskell
sum_of(T type : numeric T,
   l list T           ) ⇒
   l ? nil    ⇒ T.zero
   | c Cons ⇒ c.head + sum_of c.tail

numeric is
   type.zero numeric.this.type is abstract
```

FOSDEM'23: Algebraic Effects and Types in Fuzion
Types as First-Class Features

Type with user defined features

```plaintext
sum_of(T type : numeric T,
    l list T             ) ⇒
  l ? nil       ⇒ T.zero
  | c Cons ⇒ c.head + sum_of c.tail
```

numeric is

```plaintext
    type.zero numeric.this.type is abstract
i32 : numeric is
    type.zero i32 is 0
```
Types as First-Class Features

Type with user defined features

\[
\text{sum\_of}(T\ \text{type} : \text{numeric}\ T, \\\l\ \text{list}\ T) \Rightarrow \\
\l\ ?\ \text{nil} \quad \Rightarrow \ T.\text{zero} \\
|\ \text{c Cons} \Rightarrow \text{c.head} + \text{sum\_of}\ \text{c.tail}
\]
Types as First-Class Features

Type with user defined features

```haskell
sum_of(T type : numeric T,
       l list T           ) ⇒
  l ? nil       ⇒ T.zero
  | c Cons ⇒ c.head + sum_of c.tail
```

```haskell
say (sum_of [3.14159, 2.71828].as_list)
say (sum_of [1/3, 1/4].as_list)
say (sum_of f64 nil)
say (sum_of (fraction u8) nil)
```
Types as First-Class Features

Type with user defined features

```plaintext
sum_of(T type : numeric T,
    l list T           ) ⇒
    l ? nil    ⇒ T.zero
    | c Cons ⇒ c.head + sum_of c.tail

say (sum_of [3.14159, 2.71828].as_list)
say (sum_of [1/ 3, 1/ 4].as_list)
say (sum_of f64 nil)
say (sum_of (fraction u8) nil)
```

> fz types.fz
Types as First-Class Features

Type with user defined features

```hs
sum_of(T type : numeric T,  
   l list T           ) =>  
   l ? nil    => T.zero  
   | c Cons => c.head + sum_of c.tail
```

```hs
say (sum_of [3.14159, 2.71828].as_list)  
say (sum_of [1⁄ 3, 1⁄ 4].as_list)  
say (sum_of f64 nil)  
say (sum_of (fraction u8) nil)
```
Types as Named Effects

Example: Simple linked ring

- Creation of a linked ring requires **mutation**
- Any calculation using ring therefore uses **mutate** effect
- But feature may still be **pure** if mutation affects only temporary **local state**
Types as Named Effects
Types as Named Effects

Ring using global \texttt{mutate} effect
Types as Named Effects

Ring using global `mutate` effect

```plaintext
Ring(data String,
     old option Ring) ref is
last Ring := (old ? nil    ⇒ Ring.this
                  | r Ring ⇒ r.last  )
next := mut (old ? nil    ⇒ Ring.this
                 | r Ring ⇒ r       )
last.next ← Ring.this
```
Ring using global **mutate** effect

```haskell
Ring(data String,
    old option Ring) ref is

last Ring := (old ? nil    ⇒ Ring.this
           | r Ring ⇒ r.last  )

next := mut (old ? nil    ⇒ Ring.this
                | r Ring ⇒ r )

last.next ← Ring.this
```
Types as Named Effects

Ring using global **mutate** effect

```fuzion
Ring(data String,
    old option Ring) ref is
last Ring := (old ? nil    ⇒ Ring.this
          | r Ring ⇒ r.last   )
next := mut (old ? nil    ⇒ Ring.this
          | r Ring ⇒ r        )
last.next ← Ring.this

demo ⇒
  r := Ring "A" (Ring "B" (Ring "C" nil))
  for n := r, n.next.get; i in 1..10 do
    yak "{n.data} "
  demo
```

> fz demo.fz
Types as Named Effects

Ring using global `mutate` effect

```fuzion
Ring(data String,
    old option Ring) ref is
  last Ring := (old ? nil    ⇒ Ring.this
                  | r Ring ⇒ r.last   )
  next := mut (old ? nil    ⇒ Ring.this
                  | r Ring ⇒ r        )
  last.next ← Ring.this

demo ⇒
  r := Ring "A" (Ring "B" (Ring "C" nil))
  for n := r, n.next.get; i in 1..10 do
    yak "{n.data} "
demo
```

> fz demo.fz
A B C A B C A B C A
>

FOSDEM'23: Algebraic Effects and Types in Fuzion
Types as Named Effects

Ring using global `mutate` effect

```fuzion
Ring(data String, 
   old option Ring) ref is 
  last Ring := (old ? nil ⇒ Ring.this 
      | r Ring ⇒ r.last )
  next := mut (old ? nil ⇒ Ring.this 
      | r Ring ⇒ r )
  last.next ← Ring.this

demo ⇒ 
  r := Ring "A" (Ring "B" (Ring "C" nil))
  for n := r, n.next.get; i in 1..10 do 
    yak "{n.data} "
  demo
```

```
> fz demo.fz
A B C A B C A B C A
> fz -effects demo.fz
```
Types as Named Effects

Ring using global `mutate` effect

```fuzzion
Ring(data String,
    old option Ring) ref is
last  Ring := (old ? nil    ⇒ Ring.this
                  | r Ring ⇒ r.last     
)next := mut (old ? nil    ⇒ Ring.this
                  | r Ring ⇒ r           
)last.next ← Ring.this
```

demo ⇒
```
r := Ring "A" (Ring "B" (Ring "C" nil))
for n := r, n.next.get; i in 1..10 do
    yak "\{n.data\} "
demo
```

> fz demo.fz
A B C A B C A B C A
> fz -effects demo.fz
exit
io.err
io.out
mutate
panic
>
Types as Named Effects

Ring using **local mutability**

```plaintext
Ring(data String,
    old option Ring) ref is
last Ring := (old ? nil    ⇒ Ring.this
    | r Ring ⇒ r.last )
next := mut (old ? nil    ⇒ Ring.this
    | r Ring ⇒ r )
last.next ← Ring.this

demo ⇒
    r := Ring "A" (Ring "B" (Ring "C" nil))
    for n := r, n.next.get; i in 1..10 do
        yak "{n.data} "
demo
```

FOSDEM'23: Algebraic Effects and Types in Fuzion
Types as Named Effects

Ring using **local mutability**

```plaintext
Ring(
    data String,
    old option Ring ) ref is
last Ring := (old ? nil ⇒ Ring.this
    | r Ring ⇒ r.last )
next := mut (old ? nil ⇒ Ring.this
    | r Ring ⇒ r )
last.next ← Ring.this

demo ⇒
r := Ring "A" (Ring "B" (Ring "C" nil))
for n := r, n.next.get; i in 1..10 do
  yak "{n.data} "
demo
```
Types as Named Effects

Ring using **local mutability**

```haskell
Ring(M type : mutate,
    data String,
    old option Ring ) ref is
last Ring := (old ? nil ⇒ Ring.this
    | r Ring ⇒ r.last )
next := mut (old ? nil ⇒ Ring.this
    | r Ring ⇒ r )
last.next ← Ring.this

demo ⇒
r := Ring "A" (Ring "B" (Ring "C" nil))
for n := r, n.next.get; i in 1..10 do
    yak "\{n.data\}"
demo
```

FOSDEM'23: Algebraic Effects and Types in Fuzion

TOKIWA software
Types as Named Effects

Ring using **local mutability**

Ring(M type : mutate,
    data String,
    old option (Ring M)) ref is
last Ring M :=  (old ? nil    ⇒ Ring.this
    | r Ring ⇒ r.last  )
next :=       mut (old ? nil    ⇒ Ring.this
    | r Ring ⇒ r      )
last.next ← Ring.this

demo ⇒
    r := Ring "A" (Ring "B" (Ring    "C" nil))
    for n := r, n.next.get; i in 1..10 do
        yak "{n.data}" 
    demo
Types as Named Effects

Ring using **local mutability**

```plaintext
Ring(M type : mutate,
    data String,
    old option (Ring M)) ref is
  last Ring M := (old ? nil ⇒ Ring.this
                   | r Ring ⇒ r.last )
  next := mut (old ? nil ⇒ Ring.this
                 | r Ring ⇒ r )
  last.next ← Ring.this

demo ⇒
r := Ring "A" (Ring "B" (Ring "C" nil))
for n := r, n.next.get; i in 1..10 do
  yak "{n.data}" 

demo
```

---

**FOSDEM'23: Algebraic Effects and Types in Fuzion**

Types as Named Effects

Ring using **local mutability**

```plaintext
Ring(M type : mutate,
    data String,
    old option (Ring M)) ref is
last Ring M := (old ? nil ⇒ Ring.this
    | r Ring ⇒ r.last )
next := M.env.new (old ? nil ⇒ Ring.this
    | r Ring ⇒ r )
last.next ← Ring.this

demo ⇒
r := Ring "A" (Ring "B" (Ring "C" nil))
for n := r, n.next.get; i in 1..10 do
  yak "{n.data}" 
demo
```

FOSDEM'23: Algebraic Effects and Types in Fuzion
Types as Named Effects

Ring using **local mutability**

```plaintext
Ring(M type : mutate,
   data String,
   old option (Ring M)) ref is
last Ring M := (old ? nil  ⇒ Ring.this
                 | r Ring ⇒ r.last  )
next := M.env.new (old ? nil  ⇒ Ring.this
                   | r Ring ⇒ r     )
last.next ← Ring.this

mm : mutate is

demo ⇒
r := Ring "A" (Ring "B" (Ring "C" nil))
for n := r, n.next.get; i in 1..10 do
  yak "{n.data} "
demo
```

FOSDEM'23: Algebraic Effects and Types in Fuzion
Types as Named Effects

Ring using **local mutability**

```plaintext
Ring(M type : mutate,
    data String,
    old option (Ring M)) ref is
last Ring M := (old ? nil => Ring.this
    | r Ring => r.last)
next := M.env.new (old ? nil => Ring.this
    | r Ring => r)
last.next ← Ring.this

mm : mutate is
demo ⇒
r := Ring "A" (Ring "B" (Ring mm "C" nil))
for n := r, n.next.get; i in 1..10 do
    yak "{n.data}" 
demo
```
Types as Named Effects

Ring using **local mutability**

```plaintext
Ring(M type : mutate,
    data String,
    old option (Ring M)) ref is
last Ring M :=  (old ? nil ⇒ Ring.this
    | r Ring ⇒ r.last )
next := M.env.new (old ? nil ⇒ Ring.this
    | r Ring ⇒ r )
last.next ← Ring.this
mm : mutate is
demo ⇒
r := Ring "A" (Ring "B" (Ring mm "C" nil))
for n := r, n.next.get; i in 1..10 do
    yak "{n.data} "
mm.use () → demo
```

---

**TOKIWA**

software

---

**FOSDEM’23**: Algebraic Effects and Types in Fuzion
Types as Named Effects

Ring using **local mutability**

```fz
Ring(M type : mutate,
    data String,
    old option (Ring M)) ref is
last Ring M := (old ? nil ⇒ Ring.this
    | r Ring ⇒ r.last)
next := M.env.new (old ? nil ⇒ Ring.this
    | r Ring ⇒ r)
last.next ← Ring.this

mm : mutate is
demo ⇒
r := Ring "A" (Ring "B" (Ring mm "C" nil))
for n := r, n.next.get; i in 1..10 do
  yak "\{n.data\}" 
mm.use ()⇒demo
```

> fz demo.fz
Types as Named Effects

Ring using local mutability

```fsharp
Ring(M type : mutate,
data String,
old option (Ring M)) ref is
last Ring M := (old ? nil ⇒ Ring.this
                  | r Ring ⇒ r.last )
next := M.env.new (old ? nil ⇒ Ring.this
                    | r Ring ⇒ r )
last.next ← Ring.this

mm : mutate is
demo ⇒
r := Ring "A" (Ring "B" (Ring mm "C" nil))
for n := r, n.next.get; i in 1..10 do
  yak "{n.data} "
mm.use ()⇒demo
```

> fz demo.fz
A B C A B C A B C A
Types as Named Effects

Ring using **local mutability**

```plaintext
Ring(M type : mutate,  
   data String,  
   old option (Ring M)) ref is

last Ring M :=  (old ? nil  ⇒ Ring.this  
                  | r Ring  ⇒ r.last )  
next := M.env.new (old ? nil  ⇒ Ring.this  
                  | r Ring  ⇒ r )  

last.next ← Ring.this

mm : mutate is

demo ⇒
   r := Ring "A" (Ring "B" (Ring mm "C" nil))  
   for n := r, n.next.get; i in 1..10 do  
      yak "{n.data}"  
   mm.use ()⇒demo
```

---

```
> fz demo.fz
A B C A B C A B C A
> fz -effects demo.fz
```
Types as Named Effects

Ring using **local mutability**

```fz
Ring(M type : mutate,
    data String,
    old option (Ring M)) ref is
  last Ring M := (old ? nil  ⇒ Ring.this
                   | r Ring ⇒ r.last )
  next := M.env.new (old ? nil  ⇒ Ring.this
                      | r Ring ⇒ r         )
  last.next ← Ring.this

  mm : mutate is
  demo ⇒
    r := Ring "A" (Ring "B" (Ring mm "C" nil))
    for n := r, n.next.get; i in 1..10 do
      yak "{n.data} "
    mm.use ()⇒demo
```

```
> fz demo.fz
A B C A B C A B C A
> fz -effects demo.fz
exit
io.err
io.out
panic
```

FOSDEM'23: Algebraic Effects and Types in Fuzion
Fuzion: Status

Fuzion still under development

- language definition slowly getting more stable
- base library work in progress
- current implementation providing JVM and C backends
- Basic analysis tools available
Fuzion: Status

Fuzion still under development

- language definition slowly getting more stable
- base library work in progress
- current implementation providing JVM and C backends
- Basic analysis tools available
- Felix
Conclusion

Algebraic effects and Types as 1st class features

- complement one another surprisingly well
- effects encapsulate non-functional aspects
  - mutability
  - i/o
  - exceptions
- have a look, get involved!

@fuzion@types.pl
@FuzionLang
https://flang.dev
github.com/tokiwa-software/fuzion