The Challenges of Running the Fuzion Language on OpenJDK

Mapping a Functional Language to efficient Java Bytecode

Fridtjof Siebert
Tokiwa Software GmbH

FOSDEM 2024, 3. Feb 2024, Brussels
Who is this guy?

Fridtjof Siebert

Email: siebert@tokiwa.software
github: fridis
Twitter etc. @fridi_s
@fridi@mastodon.social
@fridis.bsky.social
fridi_si@instagram

‘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 2024: Running Fuzion on the OpenJDK
Who is this guy?

Fridtjof Siebert

Email: siebert@tokiwa.software
github: fridis
Twitter etc. @fridi_s @fridi@mastodon.social @fridis.bsky.social fridi_si@instagram

‘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 2024: Running Fuzion on the OpenJDK
The Challenges of Running the Fuzion Language on OpenJDK

overview

→ Fuzion quick intro
→ Tagged union types
→ Product types with value semantics
→ Type parameters
→ Multiple Inheritance
→ Classfile verifier
The Challenges of Running the Fuzion Language on OpenJDK

overview

» Fuzion quick intro
» Tagged union types
» Product types with value semantics
» Type parameters
» Multiple Inheritance
» Classfile verifier
Motivation: Fuzion Language

- One concept: a feature
- Systems are safety-critical
- Tools make developer’s life easier
- Fuzion is
  - statically typed
  - polymorphic: union types, parametric types, inheritance
  - pure using effects
Fuzion Toolchain

Front end

Middle end

Analyzer

JVM backend

C backend

Interpreter

.FZ

.FUM

.FAPP

.FUIR

.JVM

.CLASS

.C

FOSDEM 2024: Running Fuzion on the OpenJDK
Fuzion Toolchain

- Front end
- Middle end
- Analyzer

- .fz
- .fum
- .fapp
- .fuir

- JVM backend
- .class

- C backend
- .c

- Interpreter

FOSDEM 2024: Running Fuzion on the OpenJDK
The Challenges of Running the Fuzion Language on OpenJDK

overview

- Fuzion quick intro
- Tagged union types
- Product types with value semantics
- Type parameters
- Multiple Inheritance
- Classfile verifier
Tagged Union Types: General
Tagged Union Types: General

oven(setting off | degC | degF) is
Tagged Union Types: General

off.

oven(setting off | degC | degF) is
Tagged Union Types: General

off.
degC(v i32).

oven(setting off | degC | degF) is
Tagged Union Types: General

off.
degC(v i32).
degF(v f64).
oven(setting off | degC | degF) is
Tagged Union Types: General

off.
degC(v i32).
degF(v f64).
oven(setting off | degC | degF) is
match setting
Tagged Union Types: General

off.  
degC(v i32).  
degF(v f64).  
oven(setting off | degC | degF) is
  match setting
    off ⇒ switch_off
Tagged Union Types: General

off.
degC(v i32).
degF(v f64).
oven(setting off | degC | degF) is
  match setting
    off ⇒ switch_off
    tc degC ⇒ heat_to tc
Tagged Union Types: General

```plaintext
do.
degC(v i32).
degF(v f64).
oven(setting off | degC | degF) is
  match setting
    off    => switch_off
    tc degC => heat_to tc
    tf degF => heat_to (degC ((tf.v-32)*5/9).as_i)
```
Tagged Union Types: General

off.
degC(v i32).
degF(v f64).
oven(setting off | degC | degF) is
  match setting
    off => switch_off
    tc degC => heat_to tc
    tf degF => heat_to (degC ((tf.v-32)*5/9).as_i32)
Tagged Union Types: General

off.
degC(v i32).
degF(v f64).
oven(setting off | degC | degF) is
  match setting
    off ⇒ switch_off
    tc degC ⇒ heat_to tc
    tf degF ⇒ heat_to (degC ((tf.v-32)*5/9).as_i32)

in Java, setting will be turned into three fields
Tagged Union Types: General

off.
degC(v i32).
degF(v f64).

oven(setting_off | degC | degF) is
  match setting
    off   => switch_off
    tc degC => heat_to tc
    tf degF => heat_to (degC ((tf.v-32)*5/9).as_i32)

in Java, setting will be turned into three fields

int setting_tag; // 0, 1 or 2 for off / degC / degF
Tagged Union Types: General

```plaintext
tagged
off.
degC(v i32).
degF(v f64).
oven(setting off | degC | degF) is
  match setting
    off  ⇒ switch_off
    tc degC ⇒ heat_to tc
    tf degF ⇒ heat_to (degC ((tf.v-32)*5/9).as_i32)
```

in Java, setting will be turned into three fields

```java
int setting_tag; // 0, 1 or 2 for off / degC / degF
```
Tagged Union Types: General

off.
degC(v i32).
degF(v f64).
oven(setting off | degC | degF) is
  match setting
    off  ⇒ switch_off
    tc degC ⇒ heat_to tc
    tf degF ⇒ heat_to (degC ((tf.v-32)*5/9).as_i32)

in Java, setting will be turned into three fields

int setting_tag; // 0, 1 or 2 for off / degC / degF
int setting_degC_v;
Tagged Union Types: General

off.
degC(v i32).
degF(v f64).
oven(setting off | degC | degF) is
    match setting
        off  ⇒ switch_off
        tc degC ⇒ heat_to tc
        tf degF ⇒ heat_to (degC ((tf.v-32)*5/9).as_i32)

in Java, setting will be turned into three fields

    int setting_tag;       // 0, 1 or 2 for off / degC / degF
    int setting_degC_v;
    double setting_degF_v;
Tagged Union Types: Nullable

Temperature ref is
as_celsius i32 ⇒ abstract
oven(setting off | degC | degF) is
match setting
  off ⇒ switch_off
  tc degC ⇒ heat_to tc
  tf degF ⇒ heat_to (degC ((tf.v-32)*5/9).as_i32)
Tagged Union Types: Nullable

off.
Temperature ref is
   as_celsius i32 ⇒ abstract
oven(setting off | Temperature) is
   match setting
      off ⇒ switch_off
      tc degC ⇒ heat_to tc
      tf degF ⇒ heat_to (degC ((tf.v-32)*5/9).as_i32)
Tagged Union Types: Nullable

off.

Temperature ref is
as_celsius i32 ⇒ abstract
oven(setting off | Temperature) is
match setting
    off ⇒ switch_off
    t Temperature ⇒ heat_to t.as_celsius
Tagged Union Types: Nullable

```plaintext
def off.
  Temperature ref is
  as_celsius i32 ⇒ abstract
  oven(setting off | Temperature) is
  match setting
    off ⇒ switch_off
    t Temperature ⇒ heat_to t.as_celsius
```

In Java, this could be two fields...

```plaintext
int setting_tag;  // 0 for off, 1 for Temperature
Temperature setting_temperature;
```
Tagged Union Types: Nullable

```java
off.
Temperature ref is
  as_celsius i32 => abstract
oven(setting off | Temperature) is
match setting
  off => switch_off
  t Temperature => heat_to t.as_celsius
```

in Java, ...or a reference that might be null

```java
Temperature setting_ref; // null for off, Temperature otherwise
```
Tagged Union Types: Ref-like

off.
clean.
Temperature ref is ...
Error ref is ...
oven(setting off | clean | Temperature | Error) is ...
Tagged Union Types: Ref-like

off.
clean.
Temperature ref is
...
Error ref is
...
oven(setting off | clean | Temperature | Error) is
...

in Java, this could be three fields

```java
int setting_tag; // 0/1/2/3 for off/clean/Temperature/Error
Temperature setting_temperature;
Error setting_error;
```
Tagged Union Types: Ref-like

off.
clean.
Temperature ref is
...
Error ref is
...
oven(setting off | clean | Temperature | Error) is
...

in Java, this could be three fields

```java
int setting_tag; // 0/1/2/3 for off/clean/Temperature/Error
Temperature setting_temperature;
Error setting_error;
```
Tagged Union Types: Ref-like

off.
clean.
Temperature ref is
...
Error ref is
...
oven(setting off | clean | Temperature | Error) is
...

in Java, this could be two fields

```java
int setting_tag;
Object setting_ref;
// 0/1/2/3 for off/clean/Temperature/Error
// Temperature or Error
```
Tagged Union Types: Ref-like

off.
clean.
Temperature ref is
...
Error ref is
...
oven(setting off | clean | Temperature | Error) is
...

in Java, this could be two fields

```java
int setting_tag; // 0/1/2/3 for off/clean/Temperature/Error
Object setting_ref; // Temperature or Error
```
Tagged Union Types: Ref-like

off.
clean.
Temperature ref is

...  
Error ref is

...  
oven(setting off | clean | Temperature | Error) is

...  
in Java, this could be one field

Object setting_ref;
  // off_G, clean_G, Temperature or Error
static Object off_G = new Integer(1);
static Object clean_G = new Integer(2);
Tagged Union Types: Int-like

on.
off.
clean.
err.
oven(setting off | clean | on | err) is
...

Tagged Union Types: Int-like

```java
on.
off.
clean.
err.
oven(setting off | clean | on | err) is
...
```

in Java, this could be one field

```java
int setting_tag; // 0/1/2/3 for on/off/clean/err
```
Tagged Union Types: bool-like

on.
off.
oven(setting on | off) is ...

in Java, this could be one field

```java
int setting; // 0/1 for on/off
```
Tagged Union Types: bool-like

in Java, this could be one field

```java
int setting;  // 0/1 for on/off
```
Tagged Union Types: bool-like

on.
off.
oven(setting on | off) is ...

in Java, this could be one field

```java
boolean setting; // true/false for on/off
```
Tagged Union Types: bool-like

on.
off.
oven(setting on | off) is ...

in Java, this could be one field

```java
boolean setting;  // true/false for on/off
```
Tagged Union Types: unit-like

in Java, this could be one field

```java
boolean setting;  // true/false for on/off
```
Tagged Union Types: unit-like

```java
void.
off.
oven(setting void | off) is
...
```

in Java, this could be zero fields

```java
// nothing, oven is always off
```
Tagged Union Types: void-like

```java
void. # never used!
off. # never used!
oven(setting void | off) is ...
```

in Java, this could be zero fields

```java
// nothing, oven is always off
```

FOSDEM 2024: Running Fuzion on the OpenJDK
Tagged Union Types: void-like

```java
void. # never used!
void. # never used!
oven(setting void | void) is
...
```

in Java, this could be zero fields

```java
// nothing, oven is always off
// no code for oven, it cannot be called!
```
The Challenges of Running the Fuzion Language on OpenJDK

overview

- Fuzion quick intro ✓
- Tagged union types ✓
- Product types with value semantics
- Type parameters
- Multiple Inheritance
- Classfile verifier
Product Type defined as feature

point (x, y i32).
Product Types: As several values

point (x, y i32).

draw (p point) ! graphics ⇒
  graphics.env.draw_point p.x p.y
Product Types: As several values

point (x, y i32).

draw (p point) ! graphics ⇒
    graphics.env.draw_point p.x p.y

p1 := point 3 4
draw p1
Product Types: As several values

```
point (x, y i32).
draw (p point) ! graphics ⇒
  graphics.env.draw_point p.x p.y
p1 := point 3 4
draw p1
```
Product Types: As several values

point (x, y i32).

draw (p point) ! graphics ⇒
  graphics.env.draw_point p.x p.y

p1 := point 3 4
draw p1

in Java, we need two variables or fields

void draw(int p_x, int p_y) { ... }

FOSDEM 2024: Running Fuzion on the OpenJDK
Product Types: As several values

point \( (x, y \ i32) \).

draw \((p\ point)\) ! graphics \Rightarrow 
  graphics.env.draw_point p.x p.y

\[ p1 := \text{point 3 4} \]
draw p1

in Java, we need two variables or fields

\[
\text{void draw(int p_x, int p_y) \{ ... \}}
\]
Product Types: As several values

point (x, y i32).

draw (p point) ! graphics ⇒
  graphics.env.draw_point p.x p.y

p1 := point 3 4
draw p1

in Java, we need two variables or fields

void draw(int p_x, int p_y) {
  int p1_x = 3;
  int p1_y = 4;
}
Product Types: As several values

point (x, y i32).

draw (p point) ! graphics ⇒
  graphics.env.draw_point p.x p.y

p1 := point 3 4
draw p1

in Java, we need two variables or fields

void draw(int p_x, int p_y) { ... }
int p1_x = 3;
int p1_y = 4;
Product Types: As several values

point (x, y i32).

draw (p point) ! graphics ⇒
  graphics.env.draw_point p.x p.y

p1 := point 3 4
draw p1

in Java, we need two variables or fields

void draw(int p_x, int p_y) { ... }
int p1_x = 3;
int p1_y = 4;
draw(p1_x, p1_y);
Returning Product Type Values

point \( (x, y \ i32) \) is
  shear \( (k \ i32) \Rightarrow point \ x+k*y \ y \)

\[
p1 := \text{point} \ 3 \ 4 \\
p2 := p1.shear \ 2
\]
Returning Product Type Values

\[
\text{point (x, y i32) is }
\]
\[
\text{shear (k i32) } \Rightarrow \text{ point x+k*}y \\
\]
\[
p1 := \text{ point 3 4} \\
p2 := \text{ p1.shear 2}
\]

in Java, what can we do?
Returning Product Type Values

in Java, what can we do?
Returning Product Type Values

in Java, what can we do?

→ inline the call, so return is assignment to local variable
Returning Product Type Values

in Java, what can we do?

- **inline** the call, so return is assignment to local variable
- **alloc and return** new container for two *ints*
## Returning Product Type Values

in Java, what can we do?

- **inline** the call, so return is assignment to local variable
- **alloc and return** new container for two `ints`
- **using caller-allocated container** and call-by-ref

<table>
<thead>
<tr>
<th>Method</th>
<th>BenchMark 1</th>
<th>BenchMark 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>inline call, return assignment to local variable</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>alloc and return new container for two <code>ints</code></td>
<td>107%</td>
<td>90%</td>
</tr>
<tr>
<td>using caller-allocated container and call-by-ref</td>
<td>106%</td>
<td>99%</td>
</tr>
<tr>
<td>add static fields</td>
<td>80%</td>
<td>95%</td>
</tr>
<tr>
<td>add result fields as <code>ThreadLocal</code> values</td>
<td>6%</td>
<td></td>
</tr>
<tr>
<td>add result fields to current thread instance</td>
<td>n/a</td>
<td>95%</td>
</tr>
<tr>
<td>using global pre-allocated container and call-by-ref</td>
<td>79%</td>
<td>99%</td>
</tr>
</tbody>
</table>
Returning Product Type Values

in Java, what can we do?

➡ inline the call, so return is assignment to local variable

➡ alloc and return new container for two \texttt{int}s

➡ using \texttt{caller-allocated container} and call-by-ref

➡ add \texttt{static fields} for results
Returning Product Type Values

in Java, what can we do?

- inline the call, so return is assignment to local variable
- alloc and return new container for two \texttt{int}s
- using caller-allocated container and call-by-ref
- add static fields for results
- add result fields as \texttt{ThreadLocal} values
Returning Product Type Values

in Java, what can we do?

- **inline** the call, so return is assignment to local variable
- **alloc and return** new container for two `ints`
- using **caller_allocated container** and call-by-ref
- add **static fields** for results
- add result fields as **ThreadLocal values**
- add result fields to **current thread instance**
## Returning Product Type Values

in Java, what can we do?  

<table>
<thead>
<tr>
<th>Approach</th>
<th>JMH</th>
</tr>
</thead>
<tbody>
<tr>
<td>inline the call, so return is assignment to local variable</td>
<td>100%</td>
</tr>
<tr>
<td>alloc and return new container for two <code>int</code>s</td>
<td>107%</td>
</tr>
<tr>
<td>using caller-allocated container and call-by-ref</td>
<td>106%</td>
</tr>
<tr>
<td>add static fields for results</td>
<td>80%</td>
</tr>
<tr>
<td>add result fields as <code>ThreadLocal</code> values</td>
<td>6%</td>
</tr>
<tr>
<td>add result fields to current thread instance</td>
<td>n/a</td>
</tr>
</tbody>
</table>
Returning Product Type Values

in Java, what can we do?

- **inline** the call, so return is assignment to local variable
  - JMH: 100%
  - ad-hoc: 100%

- alloc and return new container for two `int`
  - JMH: 107%
  - ad-hoc: 90%

- using caller-allocated container and call-by-ref
  - JMH: 106%
  - ad-hoc: 99%

- add **static** fields for results
  - JMH: 80%
  - ad-hoc: 95%

- add result fields as `ThreadLocal` values
  - JMH: 6%
  - ad-hoc: 5%

- add result fields to current thread instance
  - JMH: n/a
  - ad-hoc: 95%
Returning Product Type Values

in Java, what can we do?

➡ inline the call, so return is assignment to local variable
  JMH: 100%  ad-hoc: 100%

➡ alloc and return new container for two \texttt{ints}
  JMH: 107%  ad-hoc: 90%

➡ using caller-allocated container and call-by-ref
  JMH: 106%  ad-hoc: 99%

➡ add static fields for results
  JMH: 80%  ad-hoc: 95%

➡ add result fields as \texttt{ThreadLocal} values
  JMH: 6%  ad-hoc: 5%

➡ add result fields to current thread instance
  JMH: n/a  ad-hoc: 95%
## Returning Product Type Values

in Java, what can we do?

<table>
<thead>
<tr>
<th>Option</th>
<th>JMH</th>
<th>ad-hoc</th>
</tr>
</thead>
<tbody>
<tr>
<td>➡ inline the call, so return is assignment to local variable</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>➡ alloc and return new container for two \texttt{int}s</td>
<td>107%</td>
<td>90%</td>
</tr>
<tr>
<td>➡ using caller-allocated container and call-by-ref</td>
<td>106%</td>
<td>99%</td>
</tr>
<tr>
<td>➡ add static fields for results</td>
<td>80%</td>
<td>95%</td>
</tr>
<tr>
<td>➡ add result fields as \texttt{ThreadLocal} values</td>
<td>6%</td>
<td>5%</td>
</tr>
<tr>
<td>➡ add result fields to current thread instance</td>
<td>n/a</td>
<td>95%</td>
</tr>
</tbody>
</table>
## Returning Product Type Values

in Java, what can we do?

<table>
<thead>
<tr>
<th>Method</th>
<th>JMH</th>
<th>ad-hoc</th>
</tr>
</thead>
<tbody>
<tr>
<td>inline the call, so return is assignment to local variable</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>alloc and return new container for two <code>ints</code></td>
<td>107%</td>
<td>90%</td>
</tr>
<tr>
<td>using caller-allocated container and call-by-ref</td>
<td>106%</td>
<td>99%</td>
</tr>
<tr>
<td>add static fields for results</td>
<td>80%</td>
<td>95%</td>
</tr>
<tr>
<td>add result fields as <code>ThreadLocal</code> values</td>
<td>6%</td>
<td>5%</td>
</tr>
<tr>
<td>add result fields to current thread instance</td>
<td>n/a</td>
<td>95%</td>
</tr>
</tbody>
</table>
Returning Product Type Values

Project Valhalla

→ Q-types is just what we need!

→ best with VM guarantees for no heap allocation.
The Challenges of Running the Fuzion Language on OpenJDK

Overview

- Fuzion quick intro ✓
- Tagged union types ✓
- Product types with value semantics ✓
- Type parameters
- Multiple Inheritance
- Classfile verifier
Type Parameters

Example

\[
\text{mean (a, b, c T : numeric) } \Rightarrow \\
(a + b + c) / T.\text{from_u32 } 3
\]
Type Parameters

Example

mean (a, b, c T : numeric) ⇒
(a + b + c) / T.from_u32 3

can be called with T being i32, f64, etc.

say (mean 3 4 5)
say (mean 3.14 2.71 1.41)
Type Parameters

Example

mean (a, b, c T : numeric) ⇒
(a + b + c) / T.from_u32 3

can be called with T being i32, f64, etc.

say (mean 3 4 5)
say (mean 3.14 2.71 1.41)

Java uses type erasure for generics
Type Parameters

Example

\[
\text{mean} \ (a, \ b, \ c \ T : \ \text{numeric}) \Rightarrow
\ (a + b + c) \ / \ T.\text{from}u32 \ 3
\]

can be called with \( T \) being \textit{i32}, \textit{f64}, etc.

say (mean 3 4 5)

say (mean 3.14 2.71 1.41)

Java uses type erasure for generics
Fuzion uses monomorphization!
Type Parameters

Example

```java
mean (a, b, c T : numeric) ⇒
(a + b + c) / T.from_u32 3
```

JVM backend will create several Java versions for \texttt{i32}, \texttt{f64}, etc.:

```java
mean_i32(int a, int b, int c) { return (a+b+c)/3 ; }
mean_f32(double a, double b, double c) { return (a+b+c)/3.0; }
...```
The Challenges of Running the Fuzion Language on OpenJDK

overview

- Fuzion quick intro ✓
- Tagged union types ✓
- Product types with value semantics ✓
- Type parameters ✓
- Multiple Inheritance
- Classfile verifier
Multiple Inheritance

Options for dynamic binding

- table-/lookupswitch and invokestatic
- invokedynamic
- invokeinterface
Multiple Inheritance

Options for dynamic binding

- table-/lookupswitch and invokestatic
- invokedynamic
- invokeinterface

Fuzion JVM backend

- invokestatic in case unique target type
- invokeinterface otherwise
The Challenges of Running the Fuzion Language on OpenJDK

overview

➡️ Fuzion quick intro ✓
➡️ Tagged union types ✓
➡️ Product types with value semantics ✓
➡️ Type parameters ✓
➡️ Multiple Inheritance ✓
➡️ Classfile verifier

FOSDEM 2024: Running Fuzion on the OpenJDK
Classfile Verifier

what should I say?

➡ helped a lot

➡ made JVM backend much easier than C backend
The Challenges of Running the Fuzion Language on OpenJDK

overview

➤ Fuzion quick intro ✓
➤ Tagged union types ✓
➤ Product types with value semantics ✓
➤ Type parameters ✓
➤ Multiple Inheritance ✓
➤ Classfile verifier ✓
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 & Shadow
Thank you. Any questions?

Please follow and stay informed

➡️ https://github.com/tokiwa-software/fuzion
➡️ https://fuzion-lang.dev
➡️ @FuzionLang
➡️ @Fuzion@types.pl