Fuzion Language Update

The marathon run from a language prototype to a full implementation and toolchain.
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,

‘20-... Fuzion
‘21-... Tokiwa Software
Motivation

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 Summary

Fuzion

- uses the **feature** as its main concept
- is **statically typed**
- has **inheritance** and **redefinition**
- uses **value types** and **dynamic (ref) types**
- encourages **immutability**
- offloads tasks and decisions from developers to **tools**
Fuzion Logo
Backining Company

TOKIWA
software

- supports development of Fuzion
- currently three employees
- hiring
- searching for funding
Using White Space
Using White Space

Avoiding bloat caused by

;  {  }  (  ) ,

and

➡️ use whitespace instead
➡️ increase clarity and readability
➡️ avoid errors
# Using White Space: Semicolons

Flat line feeds: same indentation

<table>
<thead>
<tr>
<th>stmtnt1</th>
<th>stmtnt2</th>
<th>stmtnt3</th>
</tr>
</thead>
<tbody>
<tr>
<td>stmtnt1;</td>
<td>stmtnt2;</td>
<td>stmtnt3;</td>
</tr>
</tbody>
</table>
Using White Space: Semicolons

Flat line feeds: same indentation

<table>
<thead>
<tr>
<th>stmnt1</th>
<th>stmnt2</th>
<th>stmnt3</th>
</tr>
</thead>
</table>

Equivalent to

| stmnt1; | stmnt2; | stmnt3; |

Statement separation in single line using ;

| stmnt1; stmnt2; stmnt3 |
Using White Space: Blocks

Indenting line feeds ⇔ {

if cond
    stmt1
else
    stmt2

Equivalent to

if cond {
    stmt1
} else {
    stmt2
}
Using White Space: Blocks

Indenting line feeds \(\Leftrightarrow\) 

\[
\begin{align*}
  \text{if cond} \\
  \quad \text{stmt1} \\
  \text{else} \\
  \quad \text{stmt2}
\end{align*}
\]

Blocks in a single line:

\[
\begin{align*}
  \text{if cond then stmt1 else stmt2}
\end{align*}
\]

Equivalent to

\[
\begin{align*}
  \text{if cond} \{ \\
  \quad \text{stmt1} \\
  \} \text{ else } \{ \\
  \quad \text{stmt2} \\
  \}
\end{align*}
\]
Using White Space: Calls

Calls do not require ( , )

Equivalent to

f a b c

f(a, b, c)
Using White Space: Calls

Calls do not require ( , )

```
f a b c
```

Equivalent to

```
f(a, b, c)
```

nesting

```
f (g x y) (h z) c
```

```
f(g(x, y), h(z), c)
```
## Using White Space: Calls & Tuples

<table>
<thead>
<tr>
<th>Calls with two arguments</th>
<th>Equivalent to</th>
</tr>
</thead>
<tbody>
<tr>
<td>f a b</td>
<td>f(a, b)</td>
</tr>
</tbody>
</table>
# Using White Space: Calls & Tuples

<table>
<thead>
<tr>
<th>Calls with two arguments</th>
<th>Equivalent to</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>f a b</code></td>
<td><code>f(a, b)</code></td>
</tr>
</tbody>
</table>

Call with one tuple as argument

| `g (a, b)`                  | `g((a, b))`   |
Using White Space: Calls & Tuples

Calls with two arguments

\[ f \ a \ b \]

Equivalent to

\[ f(a, b) \]

Call with one tuple as argument

\[ g(\ a, \ b) \]

Equivalent to

\[ g((a, b)) \]
Using White Space: Arrays

Accessing array `a` at indices `i, j`

`a[i, j]`
Using White Space: Arrays

Accessing array \( a \) at indices \( i, j \)

\[ a[i, j] \]

Creating array with elements \( x, y, z \)

\[ [x, y, z] \]
Using White Space: Arrays

Accessing array \( a \) at indices \( i, j \)

\[ a[i, j] \]

Creating array with elements \( x, y, z \)

\[ [x, y, z] \]

Passing this array as argument in call to \( f \)

\[ f [x, y, z] \]
Using White Space: Arrays

Accessing array $a$ at indices $i, j$

$\textcolor{purple}{a[i, j]}$

Creating array with elements $x, y, z$

$[x, y, z]$

Passing this array as argument in call to $f$

$f [x, y, z]$
Using White Space: Operators

Using prefix operator on argument

\[ f -x \]
Using White Space: Operators

Using prefix operator on argument          Equivalent to

\[ f \ -x \]          \[ f\ (-x) \]
## Using White Space: Operators

<table>
<thead>
<tr>
<th>Using prefix operator on argument</th>
<th>Equivalent to</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>f -x</code></td>
<td><code>f(-x)</code></td>
</tr>
</tbody>
</table>

while

<table>
<thead>
<tr>
<th><code>f - x</code></th>
<th><code>f-x</code></th>
</tr>
</thead>
</table>
Using White Space: Operators

Using prefix operator on argument

\[ f \ - \ x \]

while

\[ f \ - \ x \]

and

\[ f \ - \ x \]

Equivalent to

\[ f \ (\ - \ x ) \]

\[ f \ - \ x \]

\[ (f \ - \ ) \ x \]
### Using White Space: Operators

**Using infix operators in calls**

- `f a-b`

**Equivalent to**

- `f (a-b)`
Using White Space: Operators

Using infix operators in calls

<table>
<thead>
<tr>
<th>f a-b</th>
<th>Equivalent to</th>
</tr>
</thead>
<tbody>
<tr>
<td>f (a-b)</td>
<td></td>
</tr>
</tbody>
</table>

while

<table>
<thead>
<tr>
<th>f a -b</th>
<th>Equivalent to</th>
</tr>
</thead>
<tbody>
<tr>
<td>f (a) (-b)</td>
<td></td>
</tr>
</tbody>
</table>
Using White Space: Operators

Using infix operators in calls

\[ f \ a - b \]

while

\[ f \ a \ - b \]

and

\[ f \ a \ - b \]

Equivalent to

\[ f (a - b) \]

\[ f (a) (-b) \]

\[ (f a) - b \]
Using White Space: In Java

White space has important semantics in other languages as well, e.g., Java code

```
if (cc) x(); elsewhere();
```

is very different to

```
if (cc) x(); else where();
```

Semantic white space is basically a matter of practice and habit.
Type Inference

Avoid the need for explicit types.
Type Inference

Avoid the need for explicit types.

Occurs at assignments:

\[ x := \text{expr} \quad \text{explicit assignment} \]

\[ f(a \ T) => g(a) \quad \text{implicit assignment to feature result} \]

\[ h \ 3.14e3 \quad \text{assignment of value to feature argument} \]

Types propagates in both direction: forward and backward!
Type Inference: Feature Results

Expression type propagated forward:

\[ v := 123 \quad \text{type } \texttt{i32}, \text{ default for integer literal} \]
\[ s := \text{"hello"} \quad \text{type } \texttt{string} \]
\[ h(v \cdot \text{u8}) \Rightarrow v \gg 4 \quad \text{type } \texttt{u8} \]
\[ \text{origin} \Rightarrow \text{point } 0 \ 0 \quad \text{type } \texttt{point<i32>} \]
Type Inference: Type Arguments

Type of actual arguments in call propagated to call’s type arguments:

\[
s^2 := \text{square}<\text{i32}>\ 123 \quad \text{using explicit type parameter}
\]

\[
s^2 := \text{square}\ 123 \quad \text{using type inference}
\]
Type Inference: Lambdas

Backward propagation: Lambda type defined by what it is assigned to:

\[
\begin{align*}
\text{si}(f \ (\text{string, i32}) \rightarrow \text{string}) & \Rightarrow \text{say} \ (f \ "hello" \ 3) \\
\text{ii}(f \ (\text{i32, i32}) \rightarrow \text{i32}) & \Rightarrow \text{say} \ (f \ 10 \ 3) \\
\text{si} \ (s, i \rightarrow s \ast i) & \Rightarrow \text{lambda type} \ (\text{string, i32}) \rightarrow \text{string} \\
\text{ii} \ (s, i \rightarrow s \ast i) & \Rightarrow \text{lambda type} \ (\text{i32, i32}) \rightarrow \text{i32}
\end{align*}
\]
Type Inference: Numeric Literals

Backward propagation: Target type defines type of literal
Type Inference: Numeric Literals

Backward propagation: Target type defines type of literal

\[ u \ u_{16} := 54321 \]

literal of type \( u_{16} \)
Type Inference: Numeric Literals

Backward propagation: Target type defines type of literal

\[ u \ u16 \ := \ 54321 \] literal of type \( u16 \)

\[ v \ u16 \ := \ 54.321E3 \] literal of type \( u16 \)
Type Inference: Numeric Literals

Backward propagation: Target type defines type of literal

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>u</td>
<td>\texttt{u16} := 54321</td>
<td>literal of type \texttt{u16}</td>
</tr>
<tr>
<td>v</td>
<td>\texttt{u16} := 54.321E3</td>
<td>literal of type \texttt{u16}</td>
</tr>
<tr>
<td>w</td>
<td>:= \texttt{u16} 54321</td>
<td>literal of type \texttt{u16}</td>
</tr>
</tbody>
</table>
Type Inference: Numeric Literals

Backward propagation: Target type defines type of literal

\[
\begin{align*}
  u & \colon \text{u16} := 54321 & \text{literal of type } & \text{u16} \\
  v & \colon \text{u16} := 54.321E3 & \text{literal of type } & \text{u16} \\
  w & := \text{u16} 54321 & \text{literal of type } & \text{u16} \\
  x & \colon \text{f64} := 54321 & \text{literal of type } & \text{f64}
\end{align*}
\]
Type Inference: Numeric Literals

Backward propagation: Target type defines type of literal

u  u16 := 54321  literal of type u16
v  u16 := 54.321E3  literal of type u16
w := u16 54321  literal of type u16
x  f64 := 54321  literal of type f64
y  f64 := 54.321E3  literal of type f64
## Type Inference: Numeric Literals

Backward propagation: Target type defines type of literal

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Value</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>u</td>
<td>u16</td>
<td>54321</td>
<td>u16</td>
</tr>
<tr>
<td>v</td>
<td>u16</td>
<td>54.321E3</td>
<td>u16</td>
</tr>
<tr>
<td>w</td>
<td>u16</td>
<td>54321</td>
<td>u16</td>
</tr>
<tr>
<td>x</td>
<td>f64</td>
<td>54321</td>
<td>f64</td>
</tr>
<tr>
<td>y</td>
<td>f64</td>
<td>54.321E3</td>
<td>f64</td>
</tr>
<tr>
<td>z</td>
<td>f64</td>
<td>54321</td>
<td>f64</td>
</tr>
</tbody>
</table>
Fuzion Toolchain: Planned Design
Fuzion Toolchain: Planned Design
Fuzion Toolchain: Planned Design

.FZ

.FZ

.FZ

Front end
Fuzion Toolchain: Planned Design

Front end

.FZ

.FUZ

.FUM

.FUM
Fuzion Toolchain: Planned Design
Fuzion Toolchain: Planned Design

Front end

Middle end

Optimiser

.fz

.fum

.fuir

.jar

.elf

Back end

Back end

.TrimSpace

.TrimSpace

.TrimSpace

.TrimSpace

.TrimSpace

.TrimSpace

.TrimSpace

.TrimSpace
Fuzion Toolchain: Changes 2021

Front end

Middle end

Interpreter

Optimizer

.fz

.fzm

.fum

.fuir

.fapp

.Back end

.elf
Fuzion Toolchain: Changes 2021

- .fz
- .fz
- .fz
- .fz

Front end

Middle end

Optimiser

Interpreters

C back end

elf
Fuzion Toolchain: State Jan’22

- Frontend: .fz
- Backend: .fz
- Intermediate: .fum
- Optimizer: .fap
- Interpreter: .fuir
- C Backend: .fuir
- ELF: .elf
Fuzion Module Files

Binary file format
Fuzion Module Files

Fuzion Language Update

Fuzion Module Files

Binary file format

fmap() ped to memory,

references by indices, not names

contains features, types, expressions, source code

features are routine, field, intrinsic, abstract or intrinsic

types are feature types or type parameters

10 kinds of expressions: call, match, const, assign, ...
Fuzion Module Files

Binary file format

- fmap() ped to memory
- references using offsets, not names
- contains features, types, expressions, source code
- features are routine, field, intrinsic, abstract or choice
- types are feature types or type parameters
- 10 kinds of expressions: call, match, const, assign, pop, ...
Fuzion Module Files

Impact:

- faster compilation
- module-based correctness analysis
- define feature visibility
- container for exchange of library code
Other Tools: FZJava

Create Fuzion interface to Java module

```
java.lang.System.out.println "Hello Java !"
```
Other Tools: Language Server

Support for IDEs and editors (vim, emacs)

- completion
- signature help
- documentation
- ...

FOSDEM'22: Fuzion Language Update
Other Tools: FuzionDoc tool

Extract documentation from Fuzion source code
Fuzion: Next Steps

Development Plan

➡ intermediate files: .fum, .fapp, .fuir
➡ simple analysis tools: field init, immutability
➡ C back-end: GC, floats, etc.
   • interfacing C library code
➡ Standard Library
➡ Modeling I/O, thread communication and immutability
   • using automatic monadic lifting?
Conclusion

We are running a Marathon here

➡️ we have not made the first 10km yet

➡️ we need
  • to grow our team
  • get developer feedback
  • secure long-term funding

➡️ please get involved!

http://flang.dev
siebert@tokiwa.software
github.com/tokiwa-software/fuzion
Conclusion

We are running a Marathon here

- we have not made the first 10km yet
- we need
  - to grow our team
  - get developer feedback
  - secure long-term funding
- please get involved!

http://flang.dev
siebert@tokiwa.software
github.com/tokiwa-software/fuzion