The Verse Language: Types, Semantics, and Verification

Simon Peyton Jones

Lecture 1 - *June 29, 2026*

1 Overview

Verse is a declarative functional logic language that unifies types, expressions, and patterns. It is lenient, allowing functions to be called before arguments are evaluated, and it has an effect system rather than monads.

```
# bindings are immutable; double cannot be reassinged
double(x:int):int := x*2; double[6] # 12

# anonymous functions use =>
f := (x:int=>x+1); f[5] # 6
```

1.1 Choice

In Verse, an expression can contain zero or more values, using the choice operator. A choice expression isn't a tuple, but rather it allows a binding to range over a set of values during the evaluation of an expression.

```
v := (1 | 2 | 3); v * 2; v # 2 then 4 then 6

w := (1|2|3); x:=(4|5|6); (w,x) # (1,4) then (1, 5) then (1,6) then (2,4) then (2, 5)
    then (2,6) then (3,4) then (3, 5) then (3,6)

y := (z|2); z:=(3|4); (y,z) # (3, 3) then (4, 4) then (2, 3) then (2, 4)
```

Compiled By:
Anu Soneye, Bekah Sowards, Jack Phillips

1.2 Conditionals and Fail

Verse doesn't have booleans. Instead, it uses fail, which represents the lack of a value. An if-else evaluates to the if branch if its condition evaluates to one or more values. It evaluates to the else branch if the condition evaluates to fail.

```
if (1 | 2 | 3) {1} else {2} # 1

if (fail | fail | 2) {1} else {2} # 1

if (fail) {1} else {2} # 2
```

1.3 Arrays and Tuples

In Verse, tuples and arrays are the same thing. Arrays are functions with a finite domain. Accessing an array is equivalent to calling a function. An out-of-bounds array access evaluates to fail.

```
a := (1, 2, 3); a[0] # 1

b := (1, 2, 3, (1, 2, 3, (1, 2, 3, fail)))  # fail

c := ((1 | 2), 3, (4 | 5)) # (1, 3, 4) then (1, 3, 5) then (2, 3, 4) then (2, 3 5)
```

1.4 For

For takes two expressions. It iterates over the domain of the first expression and converts the range of the second expression to a tuple.

```
a := for {1 | 2 | 3} # (1, 2, 3)

b:=(1, 2, 3, 4, 5, 6); for (x : b) {x * x} # (1, 4, 9, 16, 25, 36)

for (x := (1 | 2 | 3 | fail | 5 | fail | 7)) {x * x} # (1, 4, 9, 25, 49)
```

1.5 Functions

Recall an array is just a function with a finite domain. It evaluates to fail if you attempt to access an out-of-bounds index. Likewise, a (closed) function evaluates to fail if you call it outside of its domain. You can also narrow a finite function by restricting its domain.

```
f(x:int):=x; f["some_string"]   # evaluates to fail because f's domain is int

as:=(8, 4, 7, 5); i:int; as[i]; i # 0 then 1 then 2 then 3

as:=(8, 4, 7, 5); i:int; i>1; as[i]; i # 2 then 3
```

1.6 Types

Types in Verse are partial functions that fail on values outside the domain of the type. Note that the function does not necessarily have to be the identity function, and Verse also supports polymorphism.

```
# natural numbers
nat(x:int)<closed><decides> := (x>=0)

# 1 or 2
oneOrTwo(x:=1|2)<closed> := x

# polymorphic swap
swap(t:type,x:t,y:t)<closed> := (y,x)
```