The Tyr Programming Language

Timm Felden

August 18, 2017

Abstract

This document defines Tyr, a research language for type-oriented programming. Type-oriented programming is a paradigm that extends on object-oriented programming. In type-oriented languages, types are first order values like integers and objects. An existing but primitive form of type orientation is the Java reflection API.

Acknowledgements

For Pony!

Contents

Part I

Core Language

1 Introduction

Type-oriented programming (TOP) is a paradigm that states that types are objects. In consequence, it is possible to perform calculations on types like any other calculation. As it is true for objects in object-oriented programming (OOP), types can be copied and may have mutable state. The mutable state of a type can be bounded by static knowledge in the same way as pointer can be restricted to point to objects of a certain type. As such, TOP implies OOP.

Tyr is a programming language created to explore this idea in practice. Tyr as a language is a descendant of C++ and Scala. In order to examine consequences of TOP for resource management, Tyr features manual memory management.

Type-level functions are descendants of C++ templates and Ada generics.

1.1 On OOP and TOP

TOP has no point without OOP for the following reason:

```
type A;
type B;

val x = new (if(phi) A else B)();
```

What could be the type of x if not $A \sqcap B$?

2 Syntax

The syntax of Tyr is inspired by Scala and SKilL.

2.1 Literals

2.2 Grammar

2.2.1 Top Level Structure

2.2.2 Members

2.2.3 Expressions

3 Semantics

The semantics of Tyr is loosely based on C++ and Scala.

3.1 Types of literal Values

To do (1)

The type of <Integer> is `int`, if no 'i' or no number behind the 'i' is supplied. If a number is supplied, the number will be used as argument for `Integer`(n). If the

$\langle \mathit{int} \rangle$ ::= ('0'-'9')+

$\langle \mathit{hex} \rangle$ ::= ('0'-'0'|'A'-'F'|'a'-'f')+

$\langle \mathit{Integer} \rangle$ ::= '-'? $\langle \mathit{int} \rangle$ ('i' $\langle \mathit{int} \rangle$?)?

$\langle \mathit{HexInteger} \rangle$ ::= `0x` $\langle \mathit{hex} \rangle$ ('i' $\langle \mathit{int} \rangle$?)?

$\langle \mathit{long} \rangle$ ::= '-'? $\langle \mathit{int} \rangle$ 'L'

$\langle \mathit{Float} \rangle$ ::= '-'? $\langle \mathit{int} \rangle$? '.' $\langle \mathit{int} \rangle$ (('e'|'E') '-'? $\langle \mathit{int} \rangle$)? ('f' $\langle \mathit{int} \rangle$?)?

$\langle \mathit{string} \rangle$ ::= '"' ~['"']* '"'

$\langle \mathit{Identifier} \rangle$ ::= '`' ~['`']+ '`'

Figure 1: Literals

$\langle \mathit{file} \rangle$ ::= ???

Figure 2: Literals

resulting Type has a named subtype in `tyr.lang`, the subtype will be chosen. Hence, `0i8` is a `byte` of value `0`. Also, `0`, `0i` and `0i32` are indistinguishable.

The type of <HexInteger> is UnsignedInteger(n), where n is the number supplied via 'i' defaulting to 32.

The type of <long> is `long`.

The type of <Float> is analogous to <Integer> except that it is based on `FloatingPoint`(n) and defaults to `double`. The type is `float` if a single `f` is supplied. This rule is designed to be compatible with common programming languages.

The type of <string> is `LiteralString`.

<Identifier> literals yield identifiers. An identifier is neither a type nor a value.

3.2 Unescaping of String and Identifier Literals

Tyr uses the same escaping mechanism as Java. Unescaping happens for <string> and <identifier> before further processing.

3.3 Access Paths introduced by local `with`

Any experession that yields a scope can be imported with a with. This import results in an access path to the names imported into the local scope. If that access has a side effect, the effect may be executed at every access to the scope. Function with an implicit `this` parameter start with an implicit `with this;` expression.

var/val: fields type var -> type field (in vtable)

defs: def -> virtual static def -> static type (ada non-poly pointer) type def -> type method

Typen: Any (top) void (<: Any) bool Integer int byte long UnsignedInteger FloatingPoint float double pointer

class Object <: pointer String <: Object IterableOnce <: String Iterable <: IterableOnce Option <: Iterable Seq <: Iterable Array <: Iterable

Part II
Compilation

modules, source paths, modules scopes, default scopes,

module naming convention: <organization>.<project> tyr.lang tyr.system tyr.collection skill.common

Part III
Libraries

4 IO

- Path (VFS) - File (cfile) - MappedFile (mmap) - Console

5 Collections

IterableOnce(T : Type) - static def for (p, b) - def foreach (f : LocalLambda[-> T])

Iterator <: IterableOnce - empty() - move() : bool - get() - for (p, b) = if(!empty) do

EquivalenceRelation(T : Type) - equals(T, T) : bool - hash(T) : int

MinimalEQR <: EquivalenceRelation - equals := == - hash := .to(Int)

Iterable <: IterableOnce

Seq <: Iterable

Array <: Seq

ArrayBuffer <: Seq

StringBuffer <: Seq(String)

List <: Seq

LinkedList <: List

Set <: Seq

HashSet(T : Type <: CT, Eq : Type(EquivalenceRelation) <: CT = MinimalEQR) <: Set

Map <: Iterable

HashMap(K : Type <: CT, V : Type <: CT, Eq : Type(EquivalenceRelation(K)) <: CT) <: Set

6 Threads

- Thread - ThreadPool - Semaphore - Mutex - Barrier

7 Native

-C method placement -C++ method placement?

Part IV

Appendix

To do...

☐ 1 (p. 3): sind die ints nicht in wahrheit LiteralInt?