Tinygrace

A Simple, Safe, and Structurally Typed Language

Timothy Jones, James Noble

Victoria University of Wellington

tim,kjx@ecs.vuw.ac.nz

July 28, 2014

Motivation

Much of early object type theory was objects-first and structural

This is in contrast to modern 'mainstream' OO languages

- FJ (nominal classes) being the canonical example here

Grace is an objects-first, structurally typed language

Objects

```
object {
  method speak { print("My name is Miles") }
  print("Miles has been created")
}
```

Objects

```
def tails = object {
  method speak { print("My name is Miles") }
  print("Miles has been created")
}
```

Classes

```
class person(name) {
    method speak { print("My name is {name}") }
    print("{name} has been created")
}

person("Nathan").speak
```

Types

```
type Person = { speak → Done }

type PersonFactory = {
  person(name : String) → Person
}
```

Type matching

```
match (animal)
  case { p : Person → p.speak }
  case { d : Dog → d.bark }
  case { c : Cat → c.meow }
```

Inheritance

```
object {
  inherits person("Marie")
  self.speak
}
```

Tinygrace

A tiny subset of Grace

- Simple reduction semantics
- Static structural typing

Safer than Featherweight Java

- Replaces casting with type matching

Syntax

$$
\begin{array}{lll}
M & ::= & \textbf{method}\ S\ \{\, e \,\} & \textit{(Method)} \\
O & ::= & \textbf{object}\,\{\, I\ \overline{M} \,\} & \textit{(Object constructor)} \\
B & ::= & \textbf{match}(e)\ \overline{\textbf{case}\,\{\, x : \tau \to e \,\}} & \textit{(Match-Case branch)} \\
\tau & ::= & \mu X.\textbf{type}\,\{\, \overline{S} \,\} \mid X \mid (\tau \mid \tau) \mid (\tau\ \&\ \tau) & \textit{(Type)} \\
S & ::= & m(\overline{x : \tau}) \to \tau & \textit{(Method signature)} \\
e & ::= & x \mid e.m(\overline{e}) \mid O \mid B & \textit{(Expression)}
\end{array}
$$

Sugar

$$T \ ::= \ \textbf{type}\ X = \tau \mid \textbf{type}\ X = \{\, \overline{S} \,\} \qquad \textit{(Type alias)}$$

$$C \ ::= \ \textbf{class}\ S\ \{\, I\ \overline{M} \,\} \qquad \textit{(Class)}$$

$$I \ ::= \ \textbf{inherits}\ \mathrm{m}(\overline{e}) \mid \epsilon \qquad \textit{(Inherits clause)}$$

Assume a **type** Object = {}

Leave off parentheses when there are no parameters

Type Sugar

$$\textbf{type}\ X = \tau \ \rightsquigarrow\ e[\tau/X]$$
$$\textbf{type}\ X = \{\,\overline{S}\,\} \ \rightsquigarrow\ e[\mu X.\textbf{type}\{\,\overline{S}\,\}/X]$$

Class Sugar

$$\textsf{class}\ S\ \{\, I\ \overline{M}\,\}\ \rightsquigarrow\ \textsf{method}\ S\ \{\,\textsf{object}\,\{\, I\ \overline{M}\,\}\,\}$$

Inherits Sugar

$$\textsf{method}\ m(\overline{x : \tau_p}) \to \tau_r\ \{\, \textsf{object}\, \{\, \overline{B}\, \}\, \}$$

$$\textsf{object}\, \{\, \textsf{inherits}\, m(\overline{e_p})\ \overline{M}\, \}\ \rightsquigarrow\ \textsf{object}\, \{\, B[\overline{e_p/x}]\ \overline{M}\, \}$$

Programs

$$P \ ::= \ (\overline{T}, \text{e})$$

Subtyping: $\Sigma \vdash \tau_1 <: \tau_2$

$$\frac{\Sigma \vdash \tau <: \tau_1}{\Sigma \vdash \tau <: \tau_1 \mid \tau_2}\ (\text{S-}|\text{-L})$$

$$\frac{\Sigma \vdash \tau <: \tau_2}{\Sigma \vdash \tau <: \tau_1 \mid \tau_2}\ (\text{S-}|\text{-R})$$

$$\frac{\Sigma \vdash \tau_1 <: \tau \qquad \Sigma \vdash \tau_2 <: \tau}{\Sigma \vdash \tau_1 \mid \tau_2 <: \tau}\ (\text{S-}|)$$

Subtyping: $\Sigma \vdash \tau_1 <: \tau_2$

$$\frac{\Sigma \vdash \tau <: \tau_1 \qquad \Sigma \vdash \tau <: \tau_2}{\Sigma \vdash \tau <: \tau_1 \,\&\, \tau_2}\ (\text{S-}\&)$$

$$\frac{\Sigma \vdash \tau_1 <: \tau}{\Sigma \vdash \tau_1 \,\&\, \tau_2 <: \tau}\ (\text{S-}\&\text{-L})$$

$$\frac{\Sigma \vdash \tau_2 <: \tau}{\Sigma \vdash \tau_1 \,\&\, \tau_2 <: \tau}\ (\text{S-}\&\text{-R})$$

Typing: $\Gamma \vdash e : \tau$

$$\frac{\tau = \textbf{type}\,\{\,\overline{m(\overline{x : \tau_p}) \to \tau_r}\,\} \qquad \cdot \vdash \tau\ \checkmark \qquad \Gamma, \mathsf{self} : \tau, \overline{\overline{x : \tau_p} \vdash e : \tau_r}}{\Gamma \vdash \textbf{object}\,\{\,\overline{\textbf{method}\ m(\overline{x : \tau_p}) \to \tau_r\ \{\,e\,\}}\,\} : \tau}$$

Typing: $\Gamma \vdash e : \tau$

$$\frac{\Gamma \vdash e : \bigcup \overline{\tau_p} \qquad \Gamma, \overline{x : \tau_p \vdash e_c : \tau_c}}{\Gamma \vdash \mathsf{match}(e)\ \overline{\mathsf{case}\,\{\, x : \tau_p \to e_c \,\}} : \bigcup \overline{\tau_c}}\ (\text{T-Case})$$

$$\frac{\Gamma \vdash e : \tau_1 \qquad \cdot \vdash \tau_1 <: \tau_2}{\Gamma \vdash e : \tau_2}\ (\text{T-Sub})$$

Reduction: $e \longmapsto e'$

$$\frac{e_s \longmapsto e_s'}{e_s.m(\overline{e_p}) \longmapsto e_s'.m(\overline{e_p})}\ (\text{R-RECV})$$

$$\frac{e_p \longmapsto e_p'}{O_s.m(\overline{O_p}, e_p, \overline{e_p}) \longmapsto O_s.m(\overline{O_p}, e_p', \overline{e_p})}\ (\text{R-PRM})$$

$$\frac{\textbf{method}\ m(\overline{x : \tau_p}) \to \tau_r\ \{\, e_r \,\} \in O_s}{O_s.m(\overline{O_p}) \longmapsto [O_s/\mathsf{self}, \overline{O_p/x}]e_r}\ (\text{R-REQ})$$

Reduction: $e \longmapsto e'$

$$\frac{e \longmapsto e'}{\textbf{match}(e)\ \overline{\textbf{case}\,\{\, x : \tau \to e_c \,\}} \longmapsto \textbf{match}(e')\ \overline{\textbf{case}\,\{\, x : \tau \to e_c \,\}}}$$

$$\frac{O \in \tau}{\textbf{match}(O)\ \textbf{case}\,\{\, x : \tau \to e_c \,\}\ \overline{C} \longmapsto [O/x]e_c}$$

$$\frac{O \notin \tau}{\textbf{match}(O)\ \textbf{case}\,\{\, x : \tau \to e_c \,\}\ \overline{C} \longmapsto \textbf{match}(O)\ \overline{C}}$$

Instance check: $O \in \tau$

$$\frac{\cdot \vdash \mathsf{type}\,\{\,\overline{S}\,\} <: \tau}{\mathsf{object}\,\{\,\overline{\mathsf{method}\ S\ \{\,e\,\}}\,\} \in \tau}\ (\text{R-Inst})$$

Properties

- Progress
- Subject Substitution
- Variant Subtraction
- Subject Reduction
- Type Soundness

Featherweight Java

An FJ program has:

- A class table
- An expression

We also want to replicate:

- Constructors
- Fields
- (Safe) Casts

```
type Person = { bestFriend → Person }

object {




}.main
```

```
type Person = { bestFriend → Person }

object {
  class person(friend : Person) → Person {
    method bestFriend → Person { friend }
  }
  class narcissist → Person {
    inherits person(self)
  }



}.main
```

```
type Person = { bestFriend → Person }

object {
  class person(friend : Person) → Person {
    method bestFriend → Person { friend }
  }
  class narcissist → Person {
    inherits person(self)
  }
  method main → Person {
    person(narcissist).bestFriend
  }
}.main
```

```
type Person = {
  bestFriend → Dog
  speak → String
}
type Dog = {
  bestFriend → Person
  bark → String
}
method getDog(hasFriend : Person | Dog) → Dog {
  match (hasFriend)
    case { dog : Dog → dog }
    case { person : Person → person.bestFriend }
}
```

Casting

```
((Pair) new Pair(new Pair(new Object(),
new Object()), new Object()).fst).snd
```

Future Work

- Completeness of the type system
- Reification of types, pattern matching
- Gradual types, Unknown

gracelang.org

tim@ecs.vuw.ac.nz