Foo
A Minimal Modern OO Calculus

Prodromos Gerakios    George Fourtounis
Yannis Smaragdakis

Department of Informatics
University of Athens

{pgerakios,gfour,smaragd}@di.uoa.gr
What

A **core** OO calculus
with *nominal* and (width) *structural subtyping*
Overview

Motivation

Semantics

Formal properties

Future directions
Why

- Well-known OO calculi (e.g., FJ) are non-minimal or only express one kind of subtyping
- We need a simple core calculus with flexibility
  - (painfully) minimal
  - study both nominal and structural subtyping
- Foo motivated by our own language modeling work
  - morphing [Huang and Smaragdakis, 2011, Gerakios et al., 2013]
Fundamentals

• Basic idea: *hybrid types* unify nominal and structural subtyping
• Very compact, tiny syntax, 15 rules for everything, non-essential features removed
• Mimics (modulo minor syntactic conventions) a tiny subset of Scala
  • our examples are executable code
Example: Extending a class

(new Employee
  { def extra() = println("add-on") }
).extra();
Example: Inheritance

Overriding a method:

class EnhancedEmployee extends Employee
{
    def extra() = println("more")
}
Example: Methods and formals

- Methods only accept one formal argument (plus the implicit this)
- But anonymous classes can see formals from their environment

```scala
class C
{
  def f(x : Integer) =
    new Object
    { def g(y : Integer) = x + y }
}
```
Example: Fields

- Fields are represented by dummy-argument methods that return the field value
- To set a field, we override its method

```java
class C {
  def field(d: Object) = 1
}
...
new C {
  def field(d: Object) = 42
}
```

- Informally, we use `obj.field` instead of `obj.field(new Object { })`
Example: Emulating multiple arguments

```scala
class Add {
  def apply(x : Integer,
             y : Integer) = x + y
}

(new Add).apply(5, 10)

becomes (Scala):

class Add {
  def x(): Integer
  def apply(y : Integer) = x() + y
}

(new Add { def x() = 5 }).apply(10)
```
Example: Structural subtyping

def fun(e : { def extra() }) = e.extra
...
fun(new Object
  { def extra() = println("subtyping") }
)
Syntax

Member type
\[ \Psi ::= m : N \rightarrow N \]

Hybrid Type
\[ N ::= C \& \bar{\Psi} \]

Member
\[ M ::= m(x) e \]

Program Value
\[ v ::= \text{new } N \{\bar{M}\} \mid x \]

Expression
\[ e ::= v \mid v.m(e) \]

Top-level classes
\[ P ::= \text{class } C = N \{\bar{M}\} \]
Hybrid types

Purely structural type:

\[ \vdash_H \Psi \]

\[ \vdash_H \text{Object} \& \Psi \]  \hspace{2cm} (W-O)

Class extended by (optional) structural part:

\[ \vdash_H (C \& \Psi) \Rightarrow \Psi'; \ldots \]

\[ \vdash_H C \& \Psi \]  \hspace{2cm} (W-C)

Method signatures (elements of \( \Psi \)):

\[ \vdash_H N, N' \]

\[ \vdash_H m : N \rightarrow N' \]  \hspace{2cm} (W-M)
Reduction

Formal argument can be reduced:

\[
\begin{align*}
    e \rightarrow_P e' \\
    \text{new } N \{\overline{M}\}.m(e) \rightarrow_P \text{new } N \{\overline{M}\}.m(e')
\end{align*}
\]  

(R-C)

Formal argument is in normal form, call method:

\[
\begin{align*}
    v' = \text{new } \ldots \quad mbody(P, N \{\overline{M}\}, m) = m(x) e \\
    \text{new } N \{\overline{M}\}.m(v') \rightarrow_P e[(\text{new } N \{\overline{M}\})/\text{this}, v'/x]
\end{align*}
\]  

(R-I)
Subtyping

Based on the hierarchy computation:

\[
\frac{
\Psi' \subseteq \Psi
}{
\vdash_H N \Rightarrow \Psi; \overline{N}, \overline{N'}
\frac{
\vdash_H N' \Rightarrow \Psi'; \overline{N'}
}{
\vdash_H N <: N'
}\]  \ (S-N)

Width subtyping (\(\subseteq\) relation)
Formal properties of Foo

- Correctness proof, being formalized in Coq
- No subsumption axiom
- Substitution lemma is special
Proof

Subject reduction, with narrowing.
If $e \rightarrow_p e'$ and $e : N$, then $\exists N', e' : N' \land N' < : N$

Foo does not admit the standard subject reduction theorem, like DOT [Amin et al., 2012]

Progress.
If $e : N$, then $\exists M, e = \text{new } N M$ or $\exists e', e \rightarrow_p e'$
Motivation

Semantics

Formal properties

Future directions

No subsumption

- Subsumption property:
  \[ \Gamma \vdash_{H} x : N \text{ and } N <: N', \text{ then } \Gamma \vdash_{H} x : N' \]
- Usually added as an axiom in the type system
- In Foo, **expressions have a single type**
- Substitutivity-of-subtypes-for-supertypes still captured by rules:
  - T-I “you can use a subtype for formal arguments”
  - T-M “you can use a subtype for method bodies”
Substitution lemma (I)

- Without subsumption, the familiar substitution lemma plays different role in the type safety proof
- Example, identity method, with $N' <: N$
  
  ```
  o = new Object { id(N o) : N = o }
  t = new N
  t' = new N'
  o.id(t) \rightarrow_P t : N
  o.id(t') \rightarrow_P t' : N'
  ```
- We cannot say that $t' : N$, so the substitution lemma does not hold for formals!
- A lemma still holds for substitution of this
Substitution lemma (II)

- Intuitively, lack of a substitution lemma for formals is not a problem.
- Values are passed/returned by rules $T-I/T-M$, which accept subtypes.
- Formally, our proof just uses the fact above directly, instead of going through a separate substitution lemma for formals.
Other core calculi

- **DOT**
  - combines nominal and structural subtypes
  - more features (path-dependent types), bigger calculus

- **Unity** [Malayeri and Aldrich, 2008]
  - structural subtyping with branding
  - similarity: internal vs. external methods
  - intersection types, depth subtyping, abstract
  - bigger calculus, e.g. 13 subtyping rules

- **Tinygrace**
  - almost as minimal as Foo, extra feature (casts)
  - structural subtyping, supports nominal subtyping if further extended with branding [Jones et al. 2015]
Future directions and applications

- We already have an extension of Foo with generics, to match FJ
- To be used in formalizing universal morphing (see our jUCM paper at MASPEGHI)
- Finish Coq proof (the usual culprit: binding representation)
Thank You!
References

N. Amin, A. Moors, and M. Odersky. Dependent Object Types: Towards a foundation for Scala’s type system. *FOOL ’12*.


Expression and method typing

Variables, new objects, method invocations:

\[
\frac{x \mapsto N \in \Gamma}{\Gamma \vdash_H x : N} \quad (T-V)
\]

\[
\frac{N = C \land \bar{\Psi} \vdash_H N \quad (\Gamma \setminus \text{this}), \text{this} \mapsto N \vdash_H \bar{\Psi} \ M}{\Gamma \vdash_H \text{new } N \{M\} : N} \quad (T-N)
\]

\[
\frac{\Gamma \vdash_H v_1 : N_1 \quad \Gamma \vdash_H e_2 : N_2 \quad \vdash_H N_2 <: N_3 \quad \vdash_H N_1 \Rightarrow \bar{\Psi}'; \ldots \quad \bar{\Psi}'(m) = N_3 \rightarrow N_4}{\Gamma \vdash_H v_1.m(e_2) : N_4} \quad (T-I)
\]

Method definitions:

\[
\frac{\Gamma, x \mapsto N \vdash_H e : N'' \quad \vdash_H N'' <: N'}{\Gamma \vdash_H m : N \rightarrow N' \quad m(x) \ e} \quad (T-M)
\]
Hierarchy computation

- Given a hybrid type $N$, extracts the pair $\Psi; \overline{N}$:
  $\Psi$: signatures for all methods that can be called on $N$
  $\overline{N}$: the “path” of parent classes towards Object

- Purely structural case:

  $\vdash_H \Psi \Rightarrow \overline{\Psi}; [\text{Object} & \overline{\Psi}]$  \hspace{1cm} (H-O)

- Involving classes:

  $H(C) = N \quad \vdash_H N \Rightarrow \overline{\Psi}; \overline{\overline{N}} \quad \vdash_H \overline{\Psi}$
  
  for all $m \in \text{dom}(\Psi) \cap \text{dom}(\overline{\Psi})$  \hspace{1cm} (H-C)

  $\Psi(m) = \overline{\Psi}(m)$

  $\vdash_H C & \overline{\Psi} \Rightarrow \overline{\Psi} \cup \overline{\overline{\Psi}} ; C & \overline{\Psi}, \overline{\overline{N}}$
Method lookup

Look up method in structural part of object:

\[ m \in \text{dom}(\overline{M}) \]
\[ \text{mbody}(P, N \{ \overline{M} \}, m) = \overline{M}(m) \]  \hspace{1cm} (M-O)

Look up method in the parent class:

\[ m \notin \text{dom}(\overline{M}) \quad P(C) = N \{ \overline{M}' \} \]
\[ \text{mbody}(P, (C \& \overline{Ψ}) \{ \overline{M} \}, m) = M \]  \hspace{1cm} (M-C)
Class definitions

\[
\begin{align*}
&\left[ \text{this} \mapsto C \land \bullet \right] \vdash_H \overline{\Psi} \overline{M} \\
&H(C) = N \quad N = C' \land \overline{\Psi} \quad \vdash_H C \land \bullet \\
&\vdash_H \text{class } C = N\{\overline{M}\}
\end{align*}
\]