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

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

```
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

```
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

```
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	Ψ	$::=$	$\mathtt{m} : \mathtt{N} \longrightarrow \mathtt{N}$
Hybrid Type	$\mathtt{N}$	$::=$	$\mathtt{C}\ \&\ \overline{\Psi}$
Member	$\mathtt{M}$	$::=$	$\mathtt{m}(\mathtt{x})\ \mathtt{e}$
Program Value	$\mathtt{v}$	$::=$	$\mathtt{new\ N}\ \{\overline{\mathtt{M}}\} \mid \mathtt{x}$
Expression	$\mathtt{e}$	$::=$	$\mathtt{v} \mid \mathtt{v.m(e)}$
Top-level classes	$\mathtt{P}$	$::=$	$\overline{\mathtt{class\ C} = \mathtt{N}\ \{\overline{\mathtt{M}}\}}$

Hybrid types

Purely structural type:

$$\frac{\vdash_{\mathtt{H}} \overline{\Psi}}{\vdash_{\mathtt{H}} \mathtt{Object}\&\overline{\Psi}} \quad \textit{(W-O)}$$

Class extended by (optional) structural part:

$$\frac{\vdash_{\mathtt{H}} (\mathtt{C}\,\&\,\overline{\Psi}) \Rightarrow \overline{\Psi'}; \ldots}{\vdash_{\mathtt{H}} \mathtt{C}\,\&\,\overline{\Psi}} \quad \textit{(W-C)}$$

Method signatures (elements of $\overline{\Psi}$):

$$\frac{\vdash_{\mathtt{H}} \mathtt{N}, \mathtt{N}'}{\vdash_{\mathtt{H}} \mathtt{m} : \mathtt{N} \longrightarrow \mathtt{N}'} \quad \textit{(W-M)}$$

Reduction

Formal argument can be reduced:

$$\frac{\mathrm{e} \longrightarrow_{\mathrm{P}} \mathrm{e}'}{\texttt{new N } \{\overline{\texttt{M}}\}\texttt{.m(e)} \longrightarrow_{\mathrm{P}} \texttt{new N } \{\overline{\texttt{M}}\}\texttt{.m(e}'\texttt{)}} \quad \textit{(R-C)}$$

Formal argument is in normal form, call method:

$$\frac{\mathtt{v}' = \texttt{new}\ldots \qquad \texttt{mbody}(\texttt{P}, \texttt{N } \{\overline{\texttt{M}}\}, \texttt{m}) = \texttt{m}(\texttt{x})\ \mathrm{e}}{\texttt{new N } \{\overline{\texttt{M}}\}\texttt{.m}(\mathtt{v}') \longrightarrow_{\mathrm{P}} \mathrm{e}[(\texttt{new N } \{\overline{\texttt{M}}\})/\texttt{this}, \mathtt{v}'/\texttt{x}]} \quad \textit{(R-I)}$$

Subtyping

Based on the hierarchy computation:

$$\frac{\overline{\Psi'} \subseteq \overline{\Psi} \quad \vdash_{\mathtt{H}} \mathtt{N} \Rightarrow \overline{\Psi}; \overline{\mathtt{N}}, \overline{\mathtt{N'}} \quad \vdash_{\mathtt{H}} \mathtt{N'} \Rightarrow \overline{\Psi'}; \overline{\mathtt{N'}}}{\vdash_{\mathtt{H}} \mathtt{N} <: \mathtt{N'}} \quad \textit{(S-N)}$$

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 $\mathrm{e} \longrightarrow_{\mathtt{P}} \mathrm{e}'$ and $\mathrm{e} : \mathtt{N}$, then $\exists \mathtt{N}'$, $\mathrm{e}' : \mathtt{N}' \wedge \mathtt{N}' <: \mathtt{N}$

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

Progress.

If $\mathrm{e} : \mathtt{N}$, then $\exists \overline{\mathtt{M}}, \mathrm{e} = \mathtt{new}\ \mathtt{N}\ \overline{\mathtt{M}}$ or $\exists \mathrm{e}'$, $\mathrm{e} \longrightarrow_{\mathtt{P}} \mathrm{e}'$

No subsumption

- Subsumption property:
  if $\Gamma \vdash_{\mathtt{H}} \mathtt{x} : \mathtt{N}$ and $\mathtt{N} <: \mathtt{N}'$, then $\Gamma \vdash_{\mathtt{H}} \mathtt{x} : \mathtt{N}'$
- 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 $\mathtt{N}' <: \mathtt{N}$:

```
o  = new Object { id(N o) : N = o }
t  = new N
t' = new N'
```

$\mathtt{o.id(t)} \longrightarrow_{\mathtt{P}} \mathtt{t} \quad : \mathtt{N}$

$\mathtt{o.id(t')} \longrightarrow_{\mathtt{P}} \mathtt{t'} \quad : \mathtt{N}'$

- 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*.
- P. Gerakios, A. Biboudis, and Y. Smaragdakis. Forsaking inheritance: Supercharged delegation in DelphJ. *OOPSLA '13*.
- S. S. Huang and Y. Smaragdakis. Morphing: Structurally shaping a class by reflecting on others. *ACM Transactions on Programming Languages and Systems*, 33(2):1–44, 2011.
- T. Jones, M. Homer, and J. Noble. Brand Objects for Nominal Typing. *ECOOP '15*.
- D. Malayeri and J. Aldrich. Integrating Nominal and Structural Subtyping. *ECOOP '08*.

Expression and method typing

Variables, `new` objects, method invocations:

$$\frac{\mathtt{x} \mapsto \mathtt{N} \in \Gamma \qquad \vdash_{\mathtt{H}} \mathtt{N}}{\Gamma \vdash_{\mathtt{H}} \mathtt{x} : \mathtt{N}} \quad \textit{(T-V)}$$

$$\frac{\mathtt{N} = \mathtt{C}\ \&\ \overline{\Psi} \qquad \vdash_{\mathtt{H}} \mathtt{N} \qquad (\Gamma \setminus \mathtt{this}), \mathtt{this} \mapsto \mathtt{N} \vdash_{\mathtt{H}} \overline{\Psi}\ \overline{\mathtt{M}}}{\Gamma \vdash_{\mathtt{H}} \mathtt{new}\ \ \mathtt{N}\ \{\overline{\mathtt{M}}\}\ :\ \mathtt{N}} \quad \textit{(T-N)}$$

$$\frac{\begin{array}{c}\Gamma \vdash_{\mathtt{H}} \mathtt{v}_1 : \mathtt{N}_1 \qquad \Gamma \vdash_{\mathtt{H}} \mathtt{e}_2 : \mathtt{N}_2 \qquad \vdash_{\mathtt{H}} \mathtt{N}_2 <: \mathtt{N}_3 \\ \vdash_{\mathtt{H}} \mathtt{N}_1 \Rightarrow \overline{\Psi'}; \ldots \qquad \overline{\Psi'}(\mathtt{m}) = \mathtt{N}_3 \longrightarrow \mathtt{N}_4\end{array}}{\Gamma \vdash_{\mathtt{H}} \mathtt{v}_1.\mathtt{m}(\mathtt{e}_2)\ :\ \mathtt{N}_4} \quad \textit{(T-I)}$$

Method definitions:

$$\frac{\Gamma, \mathtt{x} \mapsto \mathtt{N} \vdash_{\mathtt{H}} \mathtt{e} : \mathtt{N}'' \qquad \vdash_{\mathtt{H}} \mathtt{N}'' <: \mathtt{N}'}{\Gamma \vdash_{\mathtt{H}} \mathtt{m} : \mathtt{N} \longrightarrow \mathtt{N}'\ \ \mathtt{m}(\mathtt{x})\ \mathtt{e}} \quad \textit{(T-M)}$$

Hierarchy computation

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

$$\frac{\vdash_{\mathtt{H}} \overline{\Psi}}{\vdash_{\mathtt{H}} \mathtt{Object}\&\overline{\Psi} \Rightarrow \overline{\Psi}\ ;\ [\mathtt{Object}\&\overline{\Psi}]} \quad \textit{(H-O)}$$

- Involving classes:

$$\frac{\mathtt{H(C)} = \mathtt{N} \qquad \vdash_{\mathtt{H}} \mathtt{N} \Rightarrow \overline{\Psi'};\overline{\mathtt{N}} \qquad \vdash_{\mathtt{H}} \overline{\Psi} \qquad \mathtt{for\ all\ m} \in \mathrm{dom}(\overline{\Psi}) \cap \mathrm{dom}(\overline{\Psi'}) \quad \overline{\Psi}(\mathtt{m}) = \overline{\Psi'}(\mathtt{m})}{\vdash_{\mathtt{H}} \mathtt{C}\,\&\,\overline{\Psi} \Rightarrow \overline{\Psi} \cup \overline{\Psi'}\ ;\ \mathtt{C}\ \&\ \overline{\Psi},\overline{\mathtt{N}}} \quad \textit{(H-C)}$$

Method lookup

Look up method in structural part of object:

$$\frac{\mathtt{m} \in \mathsf{dom}(\overline{\mathtt{M}})}{\mathtt{mbody}(\mathtt{P}, \mathtt{N}\ \{\overline{\mathtt{M}}\}, \mathtt{m}) = \overline{\mathtt{M}}(\mathtt{m})} \quad \textit{(M-O)}$$

Look up method in the parent class:

$$\frac{\begin{array}{c}\mathtt{mbody}(\mathtt{P}, (\mathtt{N}\ \{\overline{\mathtt{M}}'\}), \mathtt{m}) = \mathtt{M} \\ \mathtt{m} \notin \mathsf{dom}(\overline{\mathtt{M}}) \qquad \mathtt{P}(\mathtt{C}) = \mathtt{N}\ \{\overline{\mathtt{M}}'\}\end{array}}{\mathtt{mbody}(\mathtt{P}, (\mathtt{C}\ \&\ \overline{\Psi})\ \{\overline{\mathtt{M}}\}, \mathtt{m}) = \mathtt{M}} \quad \textit{(M-C)}$$

Class definitions

$$\frac{\begin{array}{c}[\mathtt{this} \mapsto \mathtt{C}\ \&\ \bullet] \vdash_{\mathtt{H}} \overline{\Psi}\ \ \overline{\mathtt{M}} \\ \mathtt{H}(\mathtt{C}) = \mathtt{N} \qquad \mathtt{N} = \mathtt{C}'\ \&\ \overline{\Psi} \qquad \vdash_{\mathtt{H}} \mathtt{C}\ \&\ \bullet\end{array}}{\vdash_{\mathtt{H}} \mathtt{class}\ \mathtt{C} = \mathtt{N}\{\overline{\mathtt{M}}\}}\ \textit{(T-C)}$$