Princeton Talk on Objects

Luca Cardelli

Digital Equipment Corporation, Systems Research Center

CONTENTS

Syntax

Records

$A,B ::=$		type
	$[l_i \upsilon_i : B_i \ ^{i \in 1..n}]$	record type ($\upsilon_i \in \{^-, ^o, ^+\}$, l_i distinct)
$a,b ::=$		term
	$[l_i = b_i \ ^{i \in 1..n}]$	record (l_i distinct)
	$a.l$	field selection
	$a.l{:=}b$	field update

Simple Objects

$A,B ::=$		type
	$[l_i \upsilon_i : B_i \ ^{i \in 1..n}]$	object type ($\upsilon_i \in \{^-, ^o, ^+\}$, l_i distinct)
$a,b ::=$		term
	$[l_i = \varsigma(x_i{:}A) b_i \ ^{i \in 1..n}]$	object (l_i distinct)
	$a.l$	method invocation
	$a.l \Leftarrow \varsigma(x{:}A) b$	method override
	$clone(a)$	cloning

Object with Self

$A,B ::=$		type
	$Obj(X)[l_i \upsilon_i : B_i\{X\} \ ^{i \in 1..n}]$	object type ($\upsilon_i \in \{^-, ^o, ^+\}$, l_i distinct)
$a,b ::=$		term
	$obj(X{=}A)[l_i = \varsigma(x_i{:}X) b_i \ ^{i \in 1..n}]$	object (l_i distinct)
	$a.l$	method invocation
	$a.l \Leftarrow \varsigma(Y{<:}A, x{:}Y) b$	method override
	$clone(a)$	cloning

Ambient Syntax

A,B ::=	...	type	
	X		type variable
	Top		maximum type
	$\forall(X{<:}A)B$		bounded universal type
a,b ::=	...	term	
	x		variable
	$\lambda(X{<:}A)b$		type abstraction
	b(A)		type application
	let x:A=a in b		call-by-value let

Field Notation: [...l=b...] stands for let y:?=b in [...l=ς(x)y...] with y∉b

a.l:=b stands for let y:?=b in a.l⇐ς(x)y with y∉b

Functions: are definable

Type Rules

Judgments

$E \vdash \diamond$	well-formed environment judgment
$E \vdash A$	type judgment
$E \vdash A <: B$	subtyping judgment
$E \vdash \upsilon_1 A <: \upsilon_2 B$	component subtyping judgment
$E \vdash a : A$	value typing judgment

Ambient Type Theory

(Env ø)
$$\frac{}{\emptyset \vdash \diamond}$$

(Env x)
$$\frac{E \vdash A \qquad x \notin dom(E)}{E,x{:}A \vdash \diamond}$$

(Env X)
$$\frac{E \vdash A \qquad X \notin dom(E)}{E,X{<:}A \vdash \diamond}$$

(Type X)
$$\frac{E',X{<:}A,E'' \vdash \diamond}{E',X{<:}A,E'' \vdash X}$$

(Type Top)
$$\frac{E \vdash \diamond}{E \vdash Top}$$

(Type ∀)
$$\frac{E,X{<:}A \vdash B}{E \vdash \forall(X{<:}A)B}$$

(Sub Refl)
$$\frac{E \vdash A}{E \vdash A <: A}$$

(Sub Trans)
$$\frac{E \vdash A <: B \qquad E \vdash B <: C}{E \vdash A <: C}$$

(Sub X)
$$\frac{E',X{<:}A,E'' \vdash \diamond}{E',X{<:}A,E'' \vdash X <: A}$$

(Sub Top)
$$\frac{E \vdash A}{E \vdash A <: Top}$$

(Sub ∀)
$$\frac{E \vdash A' <: A \qquad E,X{<:}A' \vdash B <: B'}{E \vdash \forall(X{<:}A)B <: \forall(X{<:}A')B'}$$

(Val Subsumption)
$$\frac{E \vdash a : A \qquad E \vdash A <: B}{E \vdash a : B}$$

(Val x)
$$\frac{E',x{:}A,E'' \vdash \diamond}{E',x{:}A,E'' \vdash x{:}A}$$

(Val Fun2)
$$\frac{E,X{<:}A \vdash b : B}{E \vdash \lambda()b : \forall(X{<:}A)B}$$

(Val Appl2)
$$\frac{E \vdash a : \forall(X{<:}A)B\{X\} \qquad E \vdash A' <: A}{E \vdash a() : B\{A'\}}$$

(Val Let)
$$\frac{E \vdash a : A \qquad E,x{:}A \vdash b : B}{E \vdash \text{let } x{:}A{=}a \text{ in } b : B}$$

Records

(Type Record) (l_i distinct)

$$\frac{E \vdash B_i \qquad \forall i \in 1..n}{E \vdash [l_i \upsilon_i : B_i \ ^{i \in 1..n}]}$$

(Sub Record) (l_i distinct)

$$\frac{E \vdash \upsilon_i B_i <: \upsilon_i' B_i' \qquad \forall i \in 1..n}{E \vdash [l_i \upsilon_i : B_i \ ^{i \in 1..n+m}] <: [l_i \upsilon_i' : B_i' \ ^{i \in 1..n}]}$$

(Sub Invariant)

$$\frac{E \vdash B}{E \vdash {}^{o} B <: {}^{o} B}$$

(Sub Covariant)

$$\frac{E \vdash B <: B' \qquad \upsilon \in \{^{o}, ^{+}\}}{E \vdash \upsilon B <: {}^{+} B'}$$

(Sub Contravariant)

$$\frac{E \vdash B' <: B \qquad \upsilon \in \{^{o}, ^{-}\}}{E \vdash \upsilon B <: {}^{-} B'}$$

(Val Record) (l_i distinct) (where $A \equiv [l_i \upsilon_i : B_i \ ^{i \in 1..n}]$)

$$\frac{E \vdash b_i : B_i \qquad \forall i \in 1..n}{E \vdash [l_i = b_i \ ^{i \in 1..n}] : A}$$

(Val Record Select)

$$\frac{E \vdash a : [l_i \upsilon_i : B_i \ ^{i \in 1..n}] \qquad \upsilon_j \in \{^{o}, ^{+}\} \qquad j \in 1..n}{E \vdash a.l_j : B_j}$$

(Val Record Update) (where $A' \equiv [l_i \upsilon_i : B_i \ ^{i \in 1..n}]$)

$$\frac{E \vdash a : A \qquad E \vdash A <: A' \qquad E \vdash b : B_j \qquad \upsilon_j \in \{^{o}, ^{-}\} \qquad j \in 1..n}{E \vdash a.l_j := b : A}$$

Simple Objects

(Type Simple Object) (l_i distinct)

$$\frac{E \vdash B_i \qquad \forall i \in 1..n}{E \vdash [l_i \upsilon_i : B_i \ ^{i \in 1..n}]}$$

(Sub Simple Object) (l_i distinct)

$$\frac{E \vdash \upsilon_i B_i <: \upsilon_i' B_i' \qquad \forall i \in 1..n}{E \vdash [l_i \upsilon_i : B_i \ ^{i \in 1..n+m}] <: [l_i \upsilon_i' : B_i' \ ^{i \in 1..n}]}$$

(Sub Invariant)

$$\frac{E \vdash B}{E \vdash {}^{o} B <: {}^{o} B}$$

(Sub Covariant)

$$\frac{E \vdash B <: B' \qquad \upsilon \in \{^{o}, ^{+}\}}{E \vdash \upsilon B <: {}^{+} B'}$$

(Sub Contravariant)

$$\frac{E \vdash B' <: B \qquad \upsilon \in \{^{o}, ^{-}\}}{E \vdash \upsilon B <: {}^{-} B'}$$

(Val Simple Object) (l_i distinct) (where $A \equiv [l_i \upsilon_i : B_i \ ^{i \in 1..n}]$)

$$\frac{E, x_i{:}A \vdash b_i : B_i \qquad \forall i \in 1..n}{E \vdash [l_i = \varsigma(x_i{:}A) b_i \ ^{i \in 1..n}] : A}$$

(Val Simple Object Select)

$$\frac{E \vdash a : [l_i \upsilon_i : B_i \ ^{i \in 1..n}] \qquad \upsilon_j \in \{^{o}, ^{+}\} \qquad j \in 1..n}{E \vdash a.l_j : B_j}$$

(Val Simple Object Override) (where $A' \equiv [l_i \upsilon_i : B_i \ ^{i \in 1..n}]$)

$$\frac{E \vdash a : A \qquad E \vdash A <: A' \qquad E, x{:}A \vdash b : B_j \qquad \upsilon_j \in \{^{o}, ^{-}\} \qquad j \in 1..n}{E \vdash a.l_j \Leftarrow \varsigma(x{:}A) b : A}$$

(Val Simple Object Clone) (where $A' \equiv [l_i \upsilon_i : B_i \ ^{i \in 1..n}]$)

$$\frac{E \vdash a : A \qquad E \vdash A <: A'}{E \vdash \mathrm{clone}(a) : A}$$

Objects with Self

(Type Object) (l_i distinct) ($B\{X^+\} \triangleq B$ covariant in X)

$$\frac{E,X<:Top \vdash B_i\{X^+\} \qquad \forall i\in 1..n}{E \vdash Obj(X)[l_i\upsilon_i:B_i\{X\}^{\,i\in 1..n}]}$$

(Sub Object) (l_i distinct)

$$\frac{E,Y<:Obj(X)[l_i\upsilon_i:B_i\{X\}^{\,i\in 1..n+m}] \vdash \upsilon_i\, B_i\{Y\} <: \upsilon_i'\, B_i'\{Y\} \qquad \forall i\in 1..n}{E \vdash Obj(X)[l_i\upsilon_i:B_i\{X\}^{\,i\in 1..n+m}] <: Obj(X)[l_i\upsilon_i':B_i'\{X\}^{\,i\in 1..n}]}$$

(Sub Invariant)

$$\frac{E \vdash B}{E \vdash {}^{o}B <: {}^{o}B}$$

(Sub Covariant)

$$\frac{E \vdash B <: B' \qquad \upsilon\in\{^{o},^{+}\}}{E \vdash \upsilon\, B <: {}^{+}B'}$$

(Sub Contravariant)

$$\frac{E \vdash B' <: B \qquad \upsilon\in\{^{o},^{-}\}}{E \vdash \upsilon\, B <: {}^{-}B'}$$

(Val Object) (l_i distinct) (where $A\equiv Obj(X)[l_i\upsilon_i:B_i\{X\}^{\,i\in 1..n}]$)

$$\frac{E, x_i:A \vdash b_i\{A\} : B_i\{A\} \qquad \forall i\in 1..n}{E \vdash obj(X=A)[l_i=\varsigma(x_i:X)b_i\{X\}^{\,i\in 1..n}] : A}$$

(Val Select) (where $A'\equiv Obj(X)[l_i\upsilon_i:B_i\{X\}^{\,i\in 1..n}]$)

$$\frac{E \vdash a : A \qquad E \vdash A<:A' \qquad \upsilon_j\in\{^{o},^{+}\} \qquad j\in 1..n}{E \vdash a.l_j : B_j\{A\}}$$

(Val Override) (where $A'\equiv Obj(X)[l_i\upsilon_i:B_i\{X\}^{\,i\in 1..n}]$)

$$\frac{E \vdash a : A \qquad E \vdash A<:A' \qquad E, Y<:A, x:Y \vdash b\{Y,x\} : B_j\{Y\} \qquad \upsilon_j\in\{^{o},^{-}\} \qquad j\in 1..n}{E \vdash a.l_j \Leftarrow \varsigma(Y<:A, x:Y)b\{Y,x\} : A}$$

(Val Clone) (where $A'\equiv Obj(X)[l_i\upsilon_i:B_i\{X\}^{\,i\in 1..n}]$)

$$\frac{E \vdash a : A \qquad E \vdash A<:A'}{E \vdash clone(a) : A}$$

Variations for ML2000

1) Records/Objects components may be permutable (multiple subtyping) or non permutable (prefix subtyping). The latter allows direct access. The former allows practically-constant-time access, but with a factor of 3-5 in performance.

2) Subsumption of invariant components into covariant components may be allowed (enables “write-protection” of fields after initial definition) or forbidden (allows better compiler flow analysis).

3) Fields should be separated from methods, instead of being regarded as encoded from methods (otherwise fields can be overridden with methods and vice versa). This is necessary to allow fast access to fields. It is not clear, though, what is the best place to make this distinction, along the path between surface syntax and code generation.

Encoding of Class Types

TOOPL style:

Requires F-bounded quantification and recursive types, but only simple objects. Allows binary methods. Causes widespread loss of subsumption. The latter can be remedied by F-bounded matching, but the effect is to force widespread F-bounded parameterization.

Objects-with-Self style:

Requires ordinary bounded quantification and built-in Self types. Disallows binary methods, requiring them to be defined as binary functions. (Actually, binary methods can still be emulated by recursive types; this may be satisfactory in practice.) Causes no loss of subsumption.

TOOPL style

$A \equiv \mathrm{ObjectType}(X)[l_i\upsilon_i{:}B_i\{X\}^{\,i\in 1..n}] \triangleq \mu(X)[l_i\upsilon_i{:}B_i\{X\}^{\,i\in 1..n}]$

$A(C) \triangleq [l_i\upsilon_i{:}B_i\{C\}^{\,i\in 1..n}]$

$\mathrm{ClassType}(X)[l_i\upsilon_i{:}B_i\{X\}^{\,i\in 1..n}] \triangleq [\mathrm{new}{:}\ A,\ l_i{:}\ \forall(X{<:}A(X))X \rightarrow B_i\{X\}]$

Objects-with-Self style

$A \equiv \mathrm{ObjectType}(X)[l_i\upsilon_i{:}B_i\{X^+\}^{\,i\in 1..n}] \triangleq \mathrm{Obj}(X)[l_i\upsilon_i{:}B_i\{X^+\}^{\,i\in 1..n}]$

$\mathrm{ClassType}(X)[l_i\upsilon_i{:}B_i\{X^+\}^{\,i\in 1..n}] \triangleq [\mathrm{new}{:}\ A,\ l_i{:}\ \forall(X{<:}A)X \rightarrow B_i\{X\}]$

In either style one can define classes (members of ClassTypes) and subclasses that reuse or override methods of superclasses. The Objects-with-Self style may in fact be used together with simple recursive definitions to support binary methods, but without possibility to inherit them.

Compatibility of Subtyping and Method Overriding

When does this inclusion hold?

$$\mathrm{ObjectType}(X)[l_i\upsilon_i{:}B_i\{X\}^{\,i\in 1..n+m}] <: \mathrm{ObjectType}(X)[l_i\upsilon_i'{:}B_i'\{X\}^{\,i\in 1..n}]$$

(1) In Objects-with-Self style (by the rules of object types with Self) it holds if:

$$X{<:}\mathrm{ObjectType}(X)[l_i\upsilon_i{:}B_i\{X^+\}^{\,i\in 1..n+m}] \text{ implies } \upsilon_i\ B_i\{X^+\} <: \upsilon_i'\ B_i'\{X^+\}$$

Morale: subtyping is incompatibe with method overriding for methods whose types contain contravariant occurrences of Self types (binary methods). However, covariant occurrences of Self types cause no problems, since the inclusion above is easily satisfied.

(2) In TOOPL-style (by the rules for μ and simple objects) it holds if:

$$X{<:}Y \text{ implies } \upsilon_i\ B_i\{X\} <: \upsilon_i'\ B_i'\{Y\}$$

This condition can be further analyzed as follows:

If X,Y occur in matching contravariant positions (binary methods), the requirement is Y<:X, hence there can be no inclusion. So let's assume there are only covariant occurrences.

If $\upsilon_i{\equiv}\upsilon_i'{\equiv}^o$ then we have inclusion only if $B_i\{X^+\} \equiv B_i'\{Y^+\}$, that is if X,Y do not occur at all. In this case we can have inclusion and method override, but the types of the overridden methods cannot contain any instance of the Self type.

If $\upsilon_i{\equiv}\upsilon_i'{\equiv}^+$ then we have inclusion if $B_i\{X^+\} <: B_i'\{Y^+\}$; matching occurrences of X,Y cause no problem. Here we can have inclusions, but method overriding is precluded by $\upsilon_i{\equiv}\upsilon_i'{\equiv}^+$.

Morale: subtyping is incompatibe with method overriding for methods whose types contain Self types, either contravariantly or covariantly.

Motivations for Objects with Self

The motivations for moving from simple object to objects with self are:

1) Full support for delegation (i.e. overriding even methods that return self, as just explained.)

2) Disallowing contravariant occurrences of Self, so that subsumption is not impeded. (Actually, one can still code binary methods, and impede subsumption, "by hand" with recursive definitions. However, these binary methods, unlike in TOOPL-style, cannot then be easily inherited.)

3) Removing the need for F-bounded quantification, and the associated need for equality of recursive types up to unfolding. That is, simplifying the necessary ambient type theory.

Laws

1) Subsumption is good: it shoud not be impeded.

2) Self is good, provided it does not impede (1).

3) Delegation is good provided it does not impede (1,2).

No other features shall impede (1,2,3).

N.B. the popular “Inheritance is not subtyping” view favors binary methods and F-bounded quantification over subsumption and bounded quantification.

Expectations:

Class-based languages will evolve into delegation-based languages, because the latter are both simpler and more powerful.

Class mechanisms can be derived from delegation mechanisms, in a suffuciently rich ambient type theory.

Therefore, class and delegation mechanisms can be be smoothly integrated in a delegation-based framework.

Class and Prototyping Constructs

A,B ::=	*Bonus*
X, $A \rightarrow B$	$\forall(X<:A)B$
Object(X)[$l_i \upsilon_i : B_i \{X^+\}^{\ i \in I}$]	variance annotations
Class(X)[$l_i \upsilon_i : B_i \{X^+\}^{\ i \in I}$]	
a,b ::=	*Bonus*
x, λ(x:A)b, b(a)	λ(X<:A)B, b(A)
class (x:X<:A) $l_i = b_i^{\ i \in I}$ end	
extend a with (x:X<:A) $l_i = b_i^{\ i \in I}$ end	
override a by (x:X<:A) $l_i = b_i^{\ i \in I}$ end	
new(a)	object (x:X=A) $l_i = b_i^{\ i \in I}$ end
a.l	
a gets [$l_i = b_i^{\ i \in I}$]	modify a by (x:X<:A) $l_i = b_i^{\ i \in I}$ end
Class-based	**Prototype-based**

N.B. Both features are available when adopting Objects withSelf.