Java & Lambda: a Featherweight Story

join work with Lorenzo Bettini, Viviana Bono, Paola Giannini and Betti Venneri

IFIP W.G. 2.2 Brno 17-19/9/2018

Motivation

Question: how does Java use intersection types and λ-expressions?

Motivation

Question: how does Java use intersection types and λ-expressions?

Our answer: a formal calculus extending Featherweight Java

Motivation

Question: how does Java use intersection types and λ-expressions?

Our answer: a formal calculus extending Featherweight Java

Atsushi Igarashi, Benjamin C. Pierce, and Philip Wadler
Featherweight Java: A Minimal Core Calculus for Java and GJ
ACM Transactions on Programming Languages and Systems, 23(3):396–450, 2001

Motivation

Question: how does Java use intersection types and λ-expressions?

Our answer: a formal calculus extending Featherweight Java

Atsushi Igarashi, Benjamin C. Pierce, and Philip Wadler
Featherweight Java: A Minimal Core Calculus for Java and GJ
ACM Transactions on Programming Languages and Systems, 23(3):396–450, 2001

trying to formalise Java 8 following

Motivation

Question: how does Java use intersection types and λ-expressions?

Our answer: a formal calculus extending Featherweight Java

Atsushi Igarashi, Benjamin C. Pierce, and Philip Wadler
Featherweight Java: A Minimal Core Calculus for Java and GJ
ACM Transactions on Programming Languages and Systems, 23(3):396–450, 2001

trying to formalise Java 8 following

James Gosling, Bill Joy, Guy L. Steele, Gilad Bracha, and Alex Buckley
The Java Language Specification
Java SE 8 Edition, Oracle, 2015.

Typing λ-expressions

- λ-expressions are poly expressions

Typing λ-expressions

- λ-expressions are poly expressions
- contexts prescribe target types to λ-expressions

Typing λ-expressions

- λ-expressions are poly expressions
- contexts prescribe target types to λ-expressions

problem: contexts change by reduction

Typing λ-expressions

- λ-expressions are poly expressions
- contexts prescribe target types to λ-expressions

problem: contexts change by reduction

our solution: λ-expressions are decorated by their target types at run time

Class Declarations

$$\mathsf{CD} \quad ::= \quad \mathsf{class}\ \mathsf{C}\ \mathsf{extends}\ \mathsf{D}\ \mathsf{implements}\ \overrightarrow{\mathsf{I}}\ \{\overline{\mathsf{T}}\,\overline{\mathsf{f}};\ \mathsf{K}\ \overline{\mathsf{M}}\}$$

Class Declarations

$$\begin{aligned}\mathsf{CD} &::= \mathsf{class\ C\ extends\ D\ implements}\ \overrightarrow{\mathsf{I}}\ \{\overline{\mathsf{T}}\,\overline{\mathsf{f}};\ \mathsf{K}\ \overline{\mathsf{M}}\}\\ \mathsf{ID} &::= \mathsf{interface\ I\ extends}\ \overrightarrow{\mathsf{I}}\ \{\overline{\mathsf{H}};\ \overline{\mathsf{M}}\}\end{aligned}$$

Class Declarations

$$\begin{aligned}
\mathsf{CD} &::= \mathsf{class\ C\ extends\ D\ implements}\ \overrightarrow{\mathsf{I}}\ \{\overline{\mathsf{T}}\,\overline{\mathsf{f}};\ \mathsf{K}\,\overline{\mathsf{M}}\} \\
\mathsf{ID} &::= \mathsf{interface\ I\ extends}\ \overrightarrow{\mathsf{I}}\ \{\overline{\mathsf{H}};\ \overline{\mathsf{M}}\} \\
\mathsf{K} &::= \mathsf{C}(\overrightarrow{\mathsf{T}}\ \overrightarrow{\mathsf{f}})\{\mathsf{super}(\overrightarrow{\mathsf{f}});\ \mathsf{this}.\overline{\mathsf{f}} = \overline{\mathsf{f}};\}
\end{aligned}$$

Class Declarations

$$\mathsf{CD} \quad ::= \quad \mathsf{class\ C\ extends\ D\ implements}\ \overrightarrow{\mathsf{I}}\ \{\overline{\mathsf{T}}\,\overline{\mathsf{f}};\ \mathsf{K}\ \overline{\mathsf{M}}\}$$

$$\mathsf{ID} \quad ::= \quad \mathsf{interface\ I\ extends}\ \overrightarrow{\mathsf{I}}\ \{\overline{\mathsf{H}};\ \overline{\mathsf{M}}\}$$

$$\mathsf{K} \quad ::= \quad \mathsf{C}(\overrightarrow{\mathsf{T}}\ \overrightarrow{\mathsf{f}})\{\mathsf{super}(\overrightarrow{\mathsf{f}});\ \mathsf{this}.\bar{\mathsf{f}} = \bar{\mathsf{f}};\}$$

$$\mathsf{H} \quad ::= \quad \mathsf{T}\,\mathsf{m}(\overrightarrow{\mathsf{T}}\ \overrightarrow{\mathsf{x}})$$

Class Declarations

$$
\begin{array}{rcl}
CD & ::= & \text{class } C \text{ extends } D \text{ implements } \overrightarrow{I}\ \{\overline{T}\ \overline{f};\ K\ \overline{M}\} \\
ID & ::= & \text{interface } I \text{ extends } \overrightarrow{I}\ \{\overline{H};\ \overline{M}\} \\
K & ::= & C(\overrightarrow{T}\ \overrightarrow{f})\{\text{super}(\overrightarrow{f});\ \text{this}.\bar{f} = \bar{f};\} \\
H & ::= & T m(\overrightarrow{T}\ \overrightarrow{x}) \\
M & ::= & H\ \{\text{return } t;\}
\end{array}
$$

Method Headers

function `A-mh`: gives the method headers of abstract methods defined in interfaces

Method Headers

function `A-mh`: gives the method headers of abstract methods defined in interfaces
function `D-mh`: gives the method headers of default methods defined in interfaces

Method Headers

function A-mh: gives the method headers of abstract methods defined in interfaces
function D-mh: gives the method headers of default methods defined in interfaces
function mh: gives the method headers of methods defined in interfaces and classes

Method Headers

function `A-mh`: gives the method headers of abstract methods defined in interfaces
function `D-mh`: gives the method headers of default methods defined in interfaces
function `mh`: gives the method headers of methods defined in interfaces and classes

interface I { C m( );C n(){return new C( );}}

Method Headers

function `A-mh`: gives the method headers of abstract methods defined in interfaces
function `D-mh`: gives the method headers of default methods defined in interfaces
function `mh`: gives the method headers of methods defined in interfaces and classes

```
interface I { C m( );C n(){return new C( );}}
class C extends Object implements I {C( ) {super( );}
                                      C m(){return new C( );}}
```

Method Headers

function A-mh: gives the method headers of abstract methods defined in interfaces
function D-mh: gives the method headers of default methods defined in interfaces
function mh: gives the method headers of methods defined in interfaces and classes

```
interface I { C m( );C n(){return new C( );}}
class C extends Object implements I {C( ) {super( );}
                                      C m(){return new C( );}}
```

A-mh(I)=C m( )

Method Headers

function `A-mh`: gives the method headers of abstract methods defined in interfaces
function `D-mh`: gives the method headers of default methods defined in interfaces
function `mh`: gives the method headers of methods defined in interfaces and classes

```
interface I { C m( );C n(){return new C( );}}
class C extends Object implements I {C( ) {super( );}
                                        C m(){return new C( );}}
```

`A-mh`(I)=C m( )   `D-mh`(I)=C n( )

Method Headers

function A-mh: gives the method headers of abstract methods defined in interfaces
function D-mh: gives the method headers of default methods defined in interfaces
function mh: gives the method headers of methods defined in interfaces and classes

```
interface I { C m( );C n(){return new C( );}}
class C extends Object implements I {C( ) {super( );}
                                      C m(){return new C( );}}
```

A-mh(I)=C m( )   D-mh(I)=C n( )

mh(I)=C m( ), C n( )

Method Headers

function A-mh: gives the method headers of abstract methods defined in interfaces
function D-mh: gives the method headers of default methods defined in interfaces
function mh: gives the method headers of methods defined in interfaces and classes

```
interface I { C m( );C n(){return new C( );}}
class C extends Object implements I {C( ) {super( );}
                                       C m(){return new C( );}}
```

A-mh(I)=C m( )   D-mh(I)=C n( )

mh(I)=C m( ), C n( )   mh(C)=C m( ), C n( )

Types

Pre-types

$$\iota ::= \mathsf{I} \mid \iota \& \mathsf{I}$$

$$\tau ::= \mathsf{C} \mid \iota \mid \mathsf{C}\&\iota \mid \mathsf{boolean}$$

Types

Pre-types

$$\iota ::= \mathsf{I} \mid \iota\&\mathsf{I}$$

$$\tau ::= \mathsf{C} \mid \iota \mid \mathsf{C}\&\iota \mid \mathsf{boolean}$$

A pre-type τ is a type if $\mathtt{mh}(\tau)$ is defined

Types

Pre-types

$$\iota ::= \mathsf{I} \mid \iota \& \mathsf{I}$$

$$\tau ::= \mathsf{C} \mid \iota \mid \mathsf{C} \& \iota \mid \mathsf{boolean}$$

A pre-type τ is a type if $\mathtt{mh}(\tau)$ is defined

A type ι is a functional type if
$\mathtt{A\text{-}mh}(\iota)$ contains exactly one method header

Subtyping

$$\frac{CT(\mathsf{C}) = \mathsf{class\ C\ extends\ D\ implements}\ \overrightarrow{\mathsf{I}}\ \{\overline{\mathsf{T}}\,\overline{\mathsf{f}};\mathsf{K}\,\overline{\mathsf{M}}\}}{\mathsf{C} <: \mathsf{D} \quad \mathsf{C} <: \mathsf{I}_j \quad \forall\ \mathsf{I}_j \in \overrightarrow{\mathsf{I}}}\ [<:\mathsf{C}]$$

Subtyping

$$\frac{CT(\mathsf{C}) = \mathsf{class\ C\ extends\ D\ implements\ } \overrightarrow{\mathsf{I}}\ \{\overline{\mathsf{T}}\,\overline{\mathsf{f}};\ \mathsf{K}\ \overline{\mathsf{M}}\}}{\mathsf{C} <: \mathsf{D} \quad \mathsf{C} <: \mathsf{I}_j \quad \forall\ \mathsf{I}_j \in \overrightarrow{\mathsf{I}}}\ [<:\mathsf{C}]$$

$$\frac{CT(\mathsf{I}) = \mathsf{interface\ I\ extends\ } \overrightarrow{\mathsf{I}}\ \{\overline{\mathsf{H}};\ \overline{\mathsf{M}}\}}{\mathsf{I} <: \mathsf{I}_j \quad \forall\ \mathsf{I}_j \in \overrightarrow{\mathsf{I}}}\ [<:\mathsf{I}]$$

Subtyping

$$\frac{CT(\mathsf{C}) = \mathsf{class\ C\ extends\ D\ implements}\ \overrightarrow{\mathsf{I}}\ \{\overline{\mathsf{T}}\,\overline{\mathsf{f}};\mathsf{K}\,\overline{\mathsf{M}}\}}{\mathsf{C} <: \mathsf{D} \quad \mathsf{C} <: \mathsf{I}_j \quad \forall\ \mathsf{I}_j \in \overrightarrow{\mathsf{I}}}\ [<:\mathsf{C}]$$

$$\frac{CT(\mathsf{I}) = \mathsf{interface\ I\ extends}\ \overrightarrow{\mathsf{I}}\ \{\overline{\mathsf{H}};\overline{\mathsf{M}}\}}{\mathsf{I} <: \mathsf{I}_j \quad \forall\ \mathsf{I}_j \in \overrightarrow{\mathsf{I}}}\ [<:\mathsf{I}] \qquad \mathsf{T} <: \mathsf{Object}\ [<:\mathsf{Object}]$$

Subtyping

$$\frac{CT(\mathsf{C}) = \mathsf{class\ C\ extends\ D\ implements}\ \overrightarrow{\mathsf{I}}\ \{\overline{\mathsf{T}}\ \overline{\mathsf{f}};\mathsf{K}\ \overline{\mathsf{M}}\}}{\mathsf{C} <: \mathsf{D} \quad \mathsf{C} <: \mathsf{I}_j \quad \forall\ \mathsf{I}_j \in \overrightarrow{\mathsf{I}}}\ [<: \mathsf{C}]$$

$$\frac{CT(\mathsf{I}) = \mathsf{interface\ I\ extends}\ \overrightarrow{\mathsf{I}}\ \{\overline{\mathsf{H}};\overline{\mathsf{M}}\}}{\mathsf{I} <: \mathsf{I}_j \quad \forall\ \mathsf{I}_j \in \overrightarrow{\mathsf{I}}}\ [<: \mathsf{I}] \qquad \mathsf{T} <: \mathsf{Object}\ [<: \mathsf{Object}]$$

$$\frac{\tau <: \mathsf{T}_i \quad \text{for all } 1 \leq i \leq n}{\tau <: \mathsf{T}_1 \& \ldots \& \mathsf{T}_n}\ [<: \&\mathsf{R}]$$

Subtyping

$$\frac{CT(\mathsf{C}) = \text{class C extends D implements } \overrightarrow{\mathsf{I}}\ \{\overline{\mathsf{T}}\ \overline{\mathsf{f}};\ \mathsf{K}\ \overline{\mathsf{M}}\}}{\mathsf{C} <: \mathsf{D} \quad \mathsf{C} <: \mathsf{I}_j \quad \forall\ \mathsf{I}_j \in \overrightarrow{\mathsf{I}}}\ [<:\ \mathsf{C}]$$

$$\frac{CT(\mathsf{I}) = \text{interface I extends } \overrightarrow{\mathsf{I}}\ \{\overline{\mathsf{H}};\ \overline{\mathsf{M}}\}}{\mathsf{I} <: \mathsf{I}_j \quad \forall\ \mathsf{I}_j \in \overrightarrow{\mathsf{I}}}\ [<:\ \mathsf{I}] \qquad \mathsf{T} <: \text{Object}\ [<:\ \text{Object}]$$

$$\frac{\tau <: \mathsf{T}_i \quad \text{for all } 1 \leq i \leq n}{\tau <: \mathsf{T}_1 \& \ldots \& \mathsf{T}_n}\ [<:\ \&\mathsf{R}]$$

$$\frac{\mathsf{T}_i <: \tau \quad \text{for some } 1 \leq i \leq n}{\mathsf{T}_1 \& \ldots \& \mathsf{T}_n <: \tau}\ [<:\ \&\mathsf{L}]$$

Terms

$$
\begin{aligned}
\mathsf{t} ::= &\\
&\mathsf{v}\\
&\mathsf{x}\\
&\mathsf{t.f}\\
&\mathsf{t.m}(\overrightarrow{\mathsf{t}})\\
&\mathsf{new\ C}(\overrightarrow{\mathsf{t}})\\
&(\tau)\,\mathsf{t}\\
&\mathsf{t}\,?\,\mathsf{t} : \mathsf{t}
\end{aligned}
$$

Terms

$t ::=$

v

x

$t.f$

$t.m(\overrightarrow{t})$

$\text{new } C(\overrightarrow{t})$

$(\tau)\, t$

$t\, ?\, t : t$

$v ::=$

w

$\overrightarrow{p} \to t$

$w ::=$

true

false

$\text{new } C(\overrightarrow{v})$

$(\overrightarrow{p} \to t)^{\varphi}$

$p ::=$

x

$T\, x$

Lookup Functions

`fields`(C): gives the field names in class C

Lookup Functions

`fields`(C): gives the field names in class C

`A-mtype`(m; I): gives the type of the abstract method m in interface I

Lookup Functions

`fields`(C): gives the field names in class C

`A-mtype`(m; I): gives the type of the abstract method m in interface I

`D-mtype`(m; I): gives the type of default method m in interface I

Lookup Functions

fields(C): gives the field names in class C

A-mtype(m; I): gives the type of the abstract method m in interface I

D-mtype(m; I): gives the type of default method m in interface I

mtype(m; T): gives the type of method m in class in interface T

Lookup Functions

`fields`(C): gives the field names in class C

`A-mtype`(m; I): gives the type of the abstract method m in interface I

`D-mtype`(m; I): gives the type of default method m in interface I

`mtype`(m; T): gives the type of method m in class in interface T

`mbody`(m; T): gives the formal parameters and the body of method m in class in interface T

Some Reduction Rules

we want to decorate only λ-expressions
(also inside conditional branches)

Some Reduction Rules

we want to decorate only λ-expressions
(also inside conditional branches)

$$(\mathsf{t})^{?\tau} = \begin{cases} (\mathsf{t})^{\tau} & \text{if t is a pure } \lambda\text{-expression,} \\ \mathsf{t}_0 ? (\mathsf{t}_1)^{?\tau} : (\mathsf{t}_2)^{?\tau} & \text{if } \mathsf{t} = \mathsf{t}_0 ? \mathsf{t}_1 : \mathsf{t}_2 \\ \mathsf{t} & \text{otherwise} \end{cases}$$

Some Reduction Rules

$$\frac{\texttt{mbody}(\mathsf{m}; \mathsf{C}) = (\overrightarrow{\mathsf{x}}, \mathsf{t}) \quad \texttt{mtype}(\mathsf{m}; \mathsf{C}) = \overrightarrow{\mathsf{T}} \to \mathsf{T}}{\mathsf{new}\,\mathsf{C}(\overrightarrow{\mathsf{v}}).\mathsf{m}(\overrightarrow{\mathsf{u}}) \longrightarrow [\overrightarrow{\mathsf{x}} \mapsto (\overrightarrow{\mathsf{u}})^{?\overrightarrow{\mathsf{T}}}, \mathsf{this} \mapsto \mathsf{new}\,\mathsf{C}(\overrightarrow{\mathsf{v}})](\mathsf{t})^{?\mathsf{T}}} \text{[E-InvkNew]}$$

Some Reduction Rules

$$\frac{\texttt{mbody}(\mathsf{m};\mathsf{C}) = (\overrightarrow{\mathsf{x}},\mathsf{t}) \quad \texttt{mtype}(\mathsf{m};\mathsf{C}) = \overrightarrow{\mathsf{T}} \to \mathsf{T}}{\mathsf{new\ C}(\overrightarrow{\mathsf{v}}).\mathsf{m}(\overrightarrow{\mathsf{u}}) \longrightarrow [\overrightarrow{\mathsf{x}} \mapsto (\overrightarrow{\mathsf{u}})^{?\overrightarrow{\mathsf{T}}}, \mathsf{this} \mapsto \mathsf{new\ C}(\overrightarrow{\mathsf{v}})](\mathsf{t})^{?\mathsf{T}}} \text{[E-InvkNew]}$$

$$\frac{\texttt{A-mtype}(\mathsf{m};\varphi) = \overrightarrow{\mathsf{T}} \to \mathsf{T}}{(\overrightarrow{\mathsf{y}} \to \mathsf{t})^{\varphi}.\mathsf{m}(\overrightarrow{\mathsf{v}}) \longrightarrow [\vec{\mathsf{y}} \mapsto (\vec{\mathsf{v}})^{?\vec{\mathsf{T}}}](\mathsf{t})^{?\mathsf{T}}} \text{[E-Invk}\lambda\text{U-A]}$$

Some Reduction Rules

$$\frac{\texttt{mbody}(\mathsf{m}; \mathsf{C}) = (\overrightarrow{\mathsf{x}}, \mathsf{t}) \quad \texttt{mtype}(\mathsf{m}; \mathsf{C}) = \overrightarrow{\mathsf{T}} \to \mathsf{T}}{\mathsf{new\ C}(\overrightarrow{\mathsf{v}}).\mathsf{m}(\overrightarrow{\mathsf{u}}) \longrightarrow [\overrightarrow{\mathsf{x}} \mapsto (\overrightarrow{\mathsf{u}})^{?\overrightarrow{\mathsf{T}}}, \mathsf{this} \mapsto \mathsf{new\ C}(\overrightarrow{\mathsf{v}})](\mathsf{t})^{?\mathsf{T}}} \text{[E-InvkNew]}$$

$$\frac{\texttt{A-mtype}(\mathsf{m}; \varphi) = \overrightarrow{\mathsf{T}} \to \mathsf{T}}{(\overrightarrow{\mathsf{y}} \to \mathsf{t})^{\varphi}.\mathsf{m}(\overrightarrow{\mathsf{v}}) \longrightarrow [\vec{\mathsf{y}} \mapsto (\vec{\mathsf{v}})^{?\vec{\mathsf{T}}}](\mathsf{t})^{?\mathsf{T}}} \text{[E-Invk}\lambda\text{U-A]}$$

$$\frac{\texttt{A-mtype}(\mathsf{m}; \varphi) = \overrightarrow{\mathsf{T}} \to \mathsf{T}}{(\overrightarrow{\mathsf{T}}\,\overrightarrow{\mathsf{y}} \to \mathsf{t})^{\varphi}.\mathsf{m}(\overrightarrow{\mathsf{v}}) \longrightarrow [\vec{\mathsf{y}} \mapsto (\vec{\mathsf{v}})^{?\vec{\mathsf{T}}}](\mathsf{t})^{?\mathsf{T}}} \text{[E-Invk}\lambda\text{T-A]}$$

Some Reduction Rules

$$\frac{\texttt{mbody}(\mathsf{m};\mathsf{C}) = (\vec{\mathsf{x}},\mathsf{t}) \quad \texttt{mtype}(\mathsf{m};\mathsf{C}) = \vec{\mathsf{T}} \to \mathsf{T}}{\mathsf{new}\ \mathsf{C}(\vec{\mathsf{v}}).\mathsf{m}(\vec{\mathsf{u}}) \longrightarrow [\vec{\mathsf{x}} \mapsto {\color{red}(\vec{\mathsf{u}})^{?\vec{\mathsf{T}}}}, \mathsf{this} \mapsto \mathsf{new}\ \mathsf{C}(\vec{\mathsf{v}})]{\color{red}(\mathsf{t})^{?\mathsf{T}}}}\ [\text{E-InvkNew}]$$

$$\frac{\texttt{A-mtype}(\mathsf{m};\varphi) = \vec{\mathsf{T}} \to \mathsf{T}}{(\vec{\mathsf{y}} \to \mathsf{t})^{\varphi}.\mathsf{m}(\vec{\mathsf{v}}) \longrightarrow [\vec{\mathsf{y}} \mapsto (\vec{\mathsf{v}})^{?\vec{\mathsf{T}}}](\mathsf{t})^{?\mathsf{T}}}\ [\text{E-Invk}\lambda\text{U-A}]$$

$$\frac{\texttt{A-mtype}(\mathsf{m};\varphi) = \vec{\mathsf{T}} \to \mathsf{T}}{(\vec{\mathsf{T}}\,\vec{\mathsf{y}} \to \mathsf{t})^{\varphi}.\mathsf{m}(\vec{\mathsf{v}}) \longrightarrow [\vec{\mathsf{y}} \mapsto (\vec{\mathsf{v}})^{?\vec{\mathsf{T}}}](\mathsf{t})^{?\mathsf{T}}}\ [\text{E-Invk}\lambda\text{T-A}]$$

$$\frac{\texttt{mbody}(\mathsf{m};\varphi) = (\vec{\mathsf{x}},\mathsf{t}) \quad \texttt{D-mtype}(\mathsf{m};\varphi) = \vec{\mathsf{T}} \to \mathsf{T}}{(\mathsf{t}_{\lambda})^{\varphi}.\mathsf{m}(\vec{\mathsf{v}}) \longrightarrow [\vec{\mathsf{x}} \mapsto (\vec{\mathsf{v}})^{?\vec{\mathsf{T}}}, \mathsf{this} \mapsto (\mathsf{t}_{\lambda})^{\varphi}](\mathsf{t})^{?\mathsf{T}}}\ [\text{E-Invk}\lambda\text{-D}]$$

Some Reduction Rules

$$\text{class C extends Object } \{\text{C( ) } \{\text{super( ); }\} \text{ C m(I x)}\{\text{return x.n( ); }\}\}$$

$$\text{interface I } \{\text{C n( ); }\}$$

$$\text{new C( ).m}(\epsilon \to \text{ new C( )}) \longrightarrow (\epsilon \to \text{ new C( )})^{\mathsf{I}}\text{.n( )} \longrightarrow \text{ new C( )}$$

Typing arguments

m(T x)

Typing arguments

$$m(T\,x)$$

- new $C(\overrightarrow{v})$

Typing arguments

$$m(T\,x)$$

- $\text{new}\,C(\overrightarrow{v})$ $\qquad C <: T$
- $\overrightarrow{p} \to t$

Typing arguments

$$m(T\,x)$$

- $\text{new}\, C(\overrightarrow{v})$ $\qquad C <: T$
- $\overrightarrow{p} \to t$ $\qquad (\overrightarrow{p} \to t)^T$ and T is a functional type

Typing arguments

$$m(T\,x)$$

- $\text{new}\, C(\overrightarrow{v})$ $C <: T$
- $\overrightarrow{p} \to t$ $(\overrightarrow{p} \to t)^T$ and T is a functional type

$$\frac{\Gamma \vdash t : \sigma \quad \sigma <: \tau}{\Gamma \vdash^* t : \tau}$$

Typing arguments

$$m(T\,x)$$

- $\text{new}\ C(\overrightarrow{v})$ $\qquad C <: T$
- $\overrightarrow{p} \to t$ $\qquad (\overrightarrow{p} \to t)^T$ and T is a functional type

$$\frac{\Gamma \vdash t : \sigma \quad \sigma <: \tau}{\Gamma \vdash^* t : \tau} \qquad \frac{\Gamma \vdash (t_\lambda)^\varphi : \varphi}{\Gamma \vdash^* t_\lambda : \varphi}$$

Typing arguments

$$m(T\,x)$$

- new $C(\overrightarrow{v})$ $\quad C <: T$
- $\overrightarrow{p} \to t$ $\quad (\overrightarrow{p} \to t)^T$ and T is a functional type

$$\frac{\Gamma \vdash t : \sigma \quad \sigma <: \tau}{\Gamma \vdash^* t : \tau} \qquad \frac{\Gamma \vdash (t_\lambda)^\varphi : \varphi}{\Gamma \vdash^* t_\lambda : \varphi}$$

$$\frac{\Gamma \vdash (t)^{?\tau} : \sigma \quad \sigma <: \tau}{\Gamma \vdash^* t : \tau}\,[\vdash \vdash^*]$$

Some Typing Rules

$$\frac{\Gamma \vdash \mathsf{t} : \tau \quad \mathtt{mtype}(\mathsf{m}; \tau) = \overrightarrow{\mathsf{T}} \to \mathsf{T} \quad \Gamma \vdash^{*} \overrightarrow{\mathsf{t}} : \overrightarrow{\mathsf{T}}}{\Gamma \vdash \mathsf{t}.\mathsf{m}(\overrightarrow{\mathsf{t}}) : \mathsf{T}} \text{ [T-INVK]}$$

Some Typing Rules

$$\frac{\Gamma \vdash \mathsf{t} : \tau \quad \mathtt{mtype}(\mathsf{m}; \tau) = \overrightarrow{\mathsf{T}} \to \mathsf{T} \quad \Gamma \vdash^* \overrightarrow{\mathsf{t}} : \overrightarrow{\mathsf{T}}}{\Gamma \vdash \mathsf{t.m}(\overrightarrow{\mathsf{t}}) : \mathsf{T}} \text{[T-INVK]}$$

$$\frac{\mathtt{fields}(\mathsf{C}) = \overrightarrow{\mathsf{T}}\, \overrightarrow{\mathsf{f}} \quad \Gamma \vdash^* \overrightarrow{\mathsf{t}} : \overrightarrow{\mathsf{T}}}{\Gamma \vdash \mathsf{new}\, \mathsf{C}(\overrightarrow{\mathsf{t}}) : \mathsf{C}} \text{[T-NEW]}$$

Some Typing Rules

$$\frac{\Gamma \vdash \mathsf{t} : \tau \quad \mathtt{mtype}(\mathsf{m}; \tau) = \overrightarrow{\mathsf{T}} \to \mathsf{T} \quad \Gamma \vdash^* \overrightarrow{\mathsf{t}} : \overrightarrow{\mathsf{T}}}{\Gamma \vdash \mathsf{t.m}(\overrightarrow{\mathsf{t}}) : \mathsf{T}} \text{[T-INVK]}$$

$$\frac{\mathtt{fields}(\mathsf{C}) = \overrightarrow{\mathsf{T}}\,\overrightarrow{\mathsf{f}} \quad \Gamma \vdash^* \overrightarrow{\mathsf{t}} : \overrightarrow{\mathsf{T}}}{\Gamma \vdash \mathsf{new\,C}(\overrightarrow{\mathsf{t}}) : \mathsf{C}} \text{[T-NEW]}$$

$$\frac{\mathtt{A\text{-}mh}(\varphi) = \mathsf{Tm}(\overrightarrow{\mathsf{T}}\,\overrightarrow{\mathsf{x}}) \quad \Gamma, \overrightarrow{\mathsf{y}} : \overrightarrow{\mathsf{T}} \vdash^* \mathsf{t} : \mathsf{T}}{\Gamma \vdash (\overrightarrow{\mathsf{y}} \to \mathsf{t})^{\varphi} : \varphi} \text{[T-}\lambda\text{UD]}$$

Some Typing Rules

$$\frac{\Gamma \vdash \mathsf{t} : \tau \quad \mathtt{mtype}(\mathsf{m}; \tau) = \overrightarrow{\mathsf{T}} \to \mathsf{T} \quad \Gamma \vdash^* \overrightarrow{\mathsf{t}} : \overrightarrow{\mathsf{T}}}{\Gamma \vdash \mathsf{t.m}(\overrightarrow{\mathsf{t}}) : \mathsf{T}} \text{[T-INVK]}$$

$$\frac{\mathtt{fields}(\mathsf{C}) = \overrightarrow{\mathsf{T}}\,\overrightarrow{\mathsf{f}} \quad \Gamma \vdash^* \overrightarrow{\mathsf{t}} : \overrightarrow{\mathsf{T}}}{\Gamma \vdash \mathsf{new}\ \mathsf{C}(\overrightarrow{\mathsf{t}}) : \mathsf{C}} \text{[T-NEW]}$$

$$\frac{\mathtt{A\text{-}mh}(\varphi) = \mathsf{Tm}(\overrightarrow{\mathsf{T}}\,\overrightarrow{\mathsf{x}}) \quad \Gamma, \overrightarrow{\mathsf{y}} : \overrightarrow{\mathsf{T}} \vdash^* \mathsf{t} : \mathsf{T}}{\Gamma \vdash (\overrightarrow{\mathsf{y}} \to \mathsf{t})^{\varphi} : \varphi} \text{[T-}\lambda\text{UD]}$$

$$\frac{\mathtt{A\text{-}mh}(\varphi) = \mathsf{Tm}(\overrightarrow{\mathsf{T}}\,\overrightarrow{\mathsf{x}}) \quad \Gamma, \overrightarrow{\mathsf{y}} : \overrightarrow{\mathsf{T}} \vdash^* \mathsf{t} : \mathsf{T}}{\Gamma \vdash (\overrightarrow{\mathsf{T}}\,\overrightarrow{\mathsf{y}} \to \mathsf{t})^{\varphi} : \varphi} \text{[T-}\lambda\text{TD]}$$

Some Typing Rules

$$\frac{\texttt{fields}(\mathsf{C}) = \mathsf{Tf} \text{ if } \mathsf{t} \text{ is a pure } \lambda\text{-expression then} \quad \Gamma \vdash (\mathsf{t})^{\mathsf{T}} : \mathsf{T} \text{ else } \Gamma \vdash \mathsf{t} : \tau \text{ with } \tau <: \mathsf{T}}{\Gamma \vdash \mathsf{new}\, \mathsf{C}(\mathsf{t}) : \mathsf{C}}$$

Some Typing Rules

$$\text{class } C \text{ extends Object } \{C(\,)\ \{\text{super}(\,);\}\ C\ m(I\ x)\{\text{return } x.n(\,);\}\}$$

$$\text{interface } I\ \{C\ n(\,);\}$$

$$\dfrac{\dfrac{\texttt{fields}(C) = \epsilon}{\vdash \text{new } C(\,) : C} \qquad \texttt{mtype}(m; C) = I \to C \qquad \dfrac{\dfrac{\texttt{mh}(I) = C\ n(\,) \quad \dfrac{\texttt{fields}(C) = \epsilon}{\vdash \text{new } C(\,) : C}}{\vdash (\epsilon \to \text{new } C(\,))^{I} : I}}{\vdash^{*} \epsilon \to \text{new } C(\,) : I}}{\vdash \text{new } C(\,).m(\epsilon \to \text{new } C(\,)) : C}$$

Intersection Types for Conditionals

$$\frac{\Gamma \vdash \mathsf{t} : \mathsf{boolean} \quad \Gamma \vdash \mathsf{t}_1 : \tau_1 \quad \Gamma \vdash \mathsf{t}_2 : \tau_2}{\Gamma \vdash \mathsf{t}\,?\,\mathsf{t}_1 : \mathsf{t}_2 : \mathsf{C}\&\mathsf{I}_1\& \ldots \&\mathsf{I}_n} \text{[T-COND]}$$

where C is the minimal common superclass and $\mathsf{I}_1, \ldots, \mathsf{I}_n$ are the minimal common super-interfaces of τ_1, τ_2

Intersection Types for Conditionals

$$\frac{\Gamma \vdash \mathsf{t} : \mathsf{boolean} \quad \Gamma \vdash \mathsf{t}_1 : \tau_1 \quad \Gamma \vdash \mathsf{t}_2 : \tau_2}{\Gamma \vdash \mathsf{t} ?\, \mathsf{t}_1 : \mathsf{t}_2 : \mathsf{C} \& \mathsf{I}_1 \& \ldots \& \mathsf{I}_n} \text{ [T-COND]}$$

where C is the minimal common superclass and $I_1, \ldots, I_n$ are the minimal common super-interfaces of τ_1, τ_2

if t_1 or t_2 is a λ-expression the conditional is typed by its target type

Intersection Types for Conditionals

$$\text{class C extends Object } \{\text{C}(\,)\ \{\text{super}(\,);\}\ \text{C m}(\text{I x})\{\text{return x.n}(\,);\}\}$$

$$\text{interface I } \{\text{C n}(\,);\}$$

$$\text{class B extends Object implements I } \{\text{C}(\,)\ \{\text{super}(\,);\}\ \text{C n}(\,)\{\text{return new C}(\,);\}\}$$

$$\dfrac{\dfrac{\texttt{fields}(\mathsf{C}) = \epsilon}{\vdash \mathsf{new\ C}(\,) : \mathsf{C}} \quad \texttt{mtype}(\mathsf{m}; \mathsf{C}) = \mathsf{I} \to \mathsf{C} \quad \dfrac{\dfrac{\vdash \mathsf{true} : \mathsf{boolean} \quad \dfrac{\textcolor{red}{\texttt{mh}}(\mathsf{I}) = \mathsf{C\ n}(\,) \quad \dfrac{\texttt{fields}(\mathsf{C}) = \epsilon}{\vdash \mathsf{new\ C}(\,) : \mathsf{C}}}{\vdash (\epsilon \to \mathsf{new\ C}(\,))^{\mathsf{I}} : \mathsf{I}} \quad \dfrac{\texttt{fields}(\mathsf{B}) = \epsilon}{\vdash \mathsf{new\ B}(\,) : \mathsf{B}}}{\vdash \mathsf{true}?\,(\epsilon \to \mathsf{new\ C}(\,))^{\mathsf{I}} : \mathsf{new\ B}(\,) : \mathsf{I}}}{\vdash^{*} \mathsf{true}?\ \epsilon \to \mathsf{new\ C}(\,) : \mathsf{new\ B}(\,) : \mathsf{I}}}{\vdash \mathsf{new\ C}(\,).\mathsf{m}(\mathsf{true}?\ \epsilon \to \mathsf{new\ C}(\,) : \mathsf{new\ B}(\,)) : \mathsf{C}}$$

Well-formed Class Table

$$\frac{\vec{x} : \vec{T}, \mathsf{this} : I \vdash^* t : T \quad Tm(\vec{T}\,\vec{x}) \in \texttt{D-mh}(I)}{Tm(\vec{T}\,\vec{x})\{\mathsf{return}\ \ t;\} \ \ \mathit{OK}\ \mathsf{in}\ I}\ [M \quad \mathit{OK} \quad \mathsf{in} \quad I]$$

$$\frac{\vec{x} : \vec{T}, \mathsf{this} : C \vdash^* t : T \quad Tm(\vec{T}\,\vec{x}) \in \texttt{mh}(C)}{Tm(\vec{T}\,\vec{x})\{\mathsf{return}\ \ t;\} \ \ \mathit{OK}\ \mathsf{in}\ C}\ [M \quad \mathit{OK}\ \mathsf{in}\ C]$$

$$\frac{\overline{M}\ \ \mathit{OK}\ \mathsf{in}\ I \quad \texttt{mh}(I)}{\mathsf{interface}\ I\ \mathsf{extends}\ \ \vec{I}\ \{\overline{H}; \overline{M}\}\ \ \mathit{OK}}\ [I \quad \mathit{OK}]$$

$$\frac{\begin{array}{c} K = C(\vec{U}\,\vec{g}, \vec{T}\,\vec{f})\{\mathsf{super}(\vec{g}); \mathsf{this}.\bar{f} = \bar{f};\} \\ \texttt{fields}(D) = \vec{U}\,\vec{g} \quad \vec{M}\ \ \mathit{OK}\ \mathsf{in}\ C \\ \texttt{mh}(C) \quad \texttt{mtype}(m;C)\ \mathsf{defined\ implies}\ \texttt{mbody}(m;C)\ \mathsf{defined} \end{array}}{\mathsf{class}\ C\ \mathsf{extends}\ D\ \mathsf{implements}\ \ \vec{I}\ \{\overline{T}\,\bar{f}; K\,\overline{M}\}\ \ \mathit{OK}}\ [C \quad \mathit{OK}]$$

Properties

$$\frac{\Gamma \vdash \mathsf{t} : \tau \quad \tau \sim \mathsf{C}[\&\iota] \quad \sigma \sim \mathsf{D}[\&\iota'] \quad \tau \not<: \sigma \quad \text{either } \mathsf{C} <: \mathsf{D} \text{ or } \mathsf{D} <: \mathsf{C}}{\Gamma \vdash (\sigma)\, t : \sigma} \text{[T-UDCAST]}$$

Properties

$$\frac{\Gamma \vdash \mathsf{t} : \tau \quad \tau \sim \mathsf{C}[\&\iota] \quad \sigma \sim \mathsf{D}[\&\iota'] \qquad \tau \not<: \sigma \quad \text{either } \mathsf{C} <: \mathsf{D} \text{ or } \mathsf{D} <: \mathsf{C}}{\Gamma \vdash (\sigma)\, t : \sigma} \text{ [T-UDCAST]}$$

Subject Reduction: If $\Gamma \vdash \mathsf{t} : \tau$ without using rule [T-UDCAST] and $\mathsf{t} \longrightarrow \mathsf{t}'$, then $\Gamma \vdash \mathsf{t}' : \sigma$ for some $\sigma <: \tau$.

Properties

$$\frac{\Gamma \vdash \mathsf{t} : \tau \quad \tau \sim \mathsf{C}[\&\iota] \quad \sigma \sim \mathsf{D}[\&\iota'] \qquad \tau \not<: \sigma \quad \text{either } \mathsf{C} <: \mathsf{D} \text{ or } \mathsf{D} <: \mathsf{C}}{\Gamma \vdash (\sigma)\, t : \sigma} \text{[T-UDCAST]}$$

Subject Reduction: If $\Gamma \vdash \mathsf{t} : \tau$ without using rule [T-UDCAST] and $\mathsf{t} \longrightarrow \mathsf{t}'$, then $\Gamma \vdash \mathsf{t}' : \sigma$ for some $\sigma <: \tau$.

$$(\mathsf{B})\,((\mathsf{Object})\,\mathsf{newC}(\,))\longrightarrow(\mathsf{B})\,\mathsf{newC}(\,)$$

Properties

$$\frac{\Gamma \vdash \mathsf{t} : \tau \quad \tau \sim \mathsf{C}[\&\iota] \quad \sigma \sim \mathsf{D}[\&\iota'] \quad \tau \not<: \sigma \quad \text{either } \mathsf{C} <: \mathsf{D} \text{ or } \mathsf{D} <: \mathsf{C}}{\Gamma \vdash (\sigma)\, t : \sigma} \text{ [T-UDCAST]}$$

Subject Reduction: If $\Gamma \vdash \mathsf{t} : \tau$ without using rule [T-UDCAST] and $\mathsf{t} \longrightarrow \mathsf{t}'$, then $\Gamma \vdash \mathsf{t}' : \sigma$ for some $\sigma <: \tau$.

$$(\mathsf{B})\,((\mathsf{Object})\,\mathsf{newC}(\,)) \longrightarrow (\mathsf{B})\,\mathsf{newC}(\,)$$

Progress: If $\vdash \mathsf{t} : \tau$ without using rule [T-UDCAST] and t cannot reduce, then t is a proper value.

Properties

$$\frac{\Gamma \vdash \mathsf{t} : \tau \quad \tau \sim \mathsf{C}[\&\iota] \quad \sigma \sim \mathsf{D}[\&\iota'] \qquad \tau \not<: \sigma \quad \text{either } \mathsf{C} <: \mathsf{D} \text{ or } \mathsf{D} <: \mathsf{C}}{\Gamma \vdash (\sigma)\, t : \sigma} \text{[T-UDCAST]}$$

Subject Reduction: If $\Gamma \vdash \mathsf{t} : \tau$ without using rule [T-UDCAST] and $\mathsf{t} \longrightarrow \mathsf{t}'$, then $\Gamma \vdash \mathsf{t}' : \sigma$ for some $\sigma <: \tau$.

$$(\mathsf{B})\,((\mathsf{Object})\,\mathsf{newC}(\,)) \longrightarrow (\mathsf{B})\,\mathsf{newC}(\,)$$

Progress: If $\vdash \mathsf{t} : \tau$ without using rule [T-UDCAST] and t cannot reduce, then t is a proper value.

$$(\mathsf{C})\,(\mathsf{I})\,(\epsilon \to \mathsf{new\,Object}(\,)) \longrightarrow (\mathsf{C})\,(\epsilon \to \mathsf{new\,Object}(\,))^{\mathsf{I}}$$

The Power of Intersection Types

$$\dfrac{\dfrac{\dfrac{\mathsf{x} : \alpha\&(\alpha \to \beta)}{\mathsf{x} : \alpha \to \beta} \qquad \dfrac{\mathsf{x} : \alpha\&(\alpha \to \beta)}{\mathsf{x} : \alpha}}{\mathsf{xx} : \alpha}}{\lambda \mathsf{x}.\mathsf{xx} : \alpha\&(\alpha \to \beta) \to \alpha}$$

Auto-Application in Java

$$\text{interface Arg}\ \{\text{C mArg(C y);}\}\qquad \text{interface Fun}\ \{\text{Arg mFun(Arg z)}\}$$

$$\text{C auto(Arg\&Fun x)}\{\text{return x.mFun(x).mArg(newC( ));}\}$$

Intersection Types in Java: back to the future

joint work with Paola Giannini and Betti Venneri

Intersection Types in Java: back to the future

joint work with Paola Giannini and Betti Venneri

Two extensions:

- intersection types as types of arguments in objects and methods and as return types in methods
- target types with an arbitrary number of abstract methods

joint work with Paola Giannini and Betti Venneri

Two extensions:

- intersection types as types of arguments in objects and methods and as return types in methods
- target types with an arbitrary number of abstract methods

$$\frac{\tau \mathsf{m}(\overrightarrow{\tau}\,\overrightarrow{\mathsf{x}}) \in \texttt{A-mh}(\iota) \text{ implies } \Gamma, \overrightarrow{\mathsf{y}} : \overrightarrow{\tau} \vdash^{*} \mathsf{t} : \tau}{\Gamma \vdash (\overrightarrow{\mathsf{y}} \to \mathsf{t})^{\iota} : \iota}\ [\text{T-}\lambda\text{U}]$$

Questions

Thank you


obrigado
Dank U
Merci
mahalo
Köszi
спасибо
Grazie
Thank you
mauruuru
Takk
Gracias
Dziękuję
Děkuju
danke
Kiitos