Structural type inference in Java-like languages

Martin Plümicke

Baden-Wuerttemberg Cooperative State University
Stuttgart/Horb

25. Februar 2016

Overview

Motivating Example

Type inference example

```
class A {

    m (v) {
        return v.elementAt(0);
    }
}
```

Motivating Example

Type inference example

```
class A {

    m (v) {
        return v.elementAt(0);
    }
}
```

No type inference result for `m`

Motivating Example

Type inference example

```
import java.util.Vector;

class A {

    m (v) {
        return v.elementAt(0);
    }
}
```

Motivating Example

Type inference example

```
import java.util.Vector;

class A {

    m (v) {
        return v.elementAt(0);
    }
}
```

Result: `m : Vector<T>` → `T`.

Motivating Example

Type inference example

```
class A {

    m (v) {
        return v.elementAt(0);
    }
}
```

Motivating Example

Type inference example

```
interface I<T> { T elementAt(int x); }

class A {

    m (v) {
        return v.elementAt(0);
    }
}
```

Result: `m : I<T> → T`.

Input language (extended Featherweight Java)

$$
\begin{array}{lll}
\texttt{L} & ::= & \texttt{class C extends (CT)}^{*}\ \{\overline{\texttt{f}};\ \overline{\texttt{M}}\} \\
\texttt{M} & ::= & \texttt{m}(\overline{\texttt{x}})\{\ \texttt{return e;}\ \} \\
\texttt{e} & ::= & \texttt{x} \mid \texttt{e.f} \mid \texttt{e.m}(\overline{\texttt{e}}) \mid \texttt{new NCT}(\overline{\texttt{e}}) \mid \texttt{(CT)e} \\
\texttt{NCT} & ::= & \texttt{CT} \mid \texttt{C<}\overline{\texttt{TVar = CT}}\texttt{>} \\
\texttt{CT} & ::= & \texttt{C<}\overline{\texttt{CT}}\texttt{>}
\end{array}
$$

Output language

$$
\begin{array}{lll}
\mathtt{L}_t & ::= & \mathtt{I}^* \ \mathtt{CL}_t \\
\mathtt{CL}_t & ::= & \mathtt{class\ C<}\overline{\mathtt{TVar}}\mathtt{>[}\overline{\mathtt{CONS}}\mathtt{]\ extends\ (CT)}^* \ \{\overline{\mathtt{T\ f}};\overline{\mathtt{M}_t}\} \\
\mathtt{CONS} & ::= & \mathtt{T\ extends\ T} \\
\mathtt{T} & ::= & \mathtt{CT} \mid \mathtt{TVar} \\
\mathtt{MH} & ::= & \mathtt{T\ m}(\overline{\mathtt{T}}\ \overline{\mathtt{x}}); \\
\mathtt{M}_t & ::= & \mathtt{MH}\ \{\ \mathtt{return\ e}_t;\ \} \\
\mathtt{e}_t & ::= & \mathtt{x} : \mathtt{T} \mid \mathtt{e.f} : \mathtt{T} \mid \mathtt{e.m}(\overline{\mathtt{e}}) : \mathtt{T} \mid \mathtt{new\ NCT}(\overline{\mathtt{e}}) : \mathtt{CT} \mid (\mathtt{CT})\mathtt{e} : \mathtt{CT} \\
\mathtt{I} & ::= & \mathtt{interface\ I<}\overline{\mathtt{TVar}}\mathtt{>}\{\overline{\mathtt{T}}\ \overline{\mathtt{f}};\overline{\mathtt{MH}}\}
\end{array}
$$

The Algorithm

TI: $\texttt{TypeAssumptions} \times \texttt{L} \rightarrow \texttt{L}_t$

TI $(\ Ass, cl\ ) =$
  let
    $(cl_t, C) =$ **TYPE**$(\ Ass, cl\ )$
    $(I_1 \ldots I_m\ cl_t) =$ **construct**$(\ cl_t, C\ )$
  in
    $(I_1 \ldots I_m$ **solve**$(\ cl_t\ ))$

The function TYPE

TYPE: $\texttt{TypeAssumptions} \times \mathtt{L} \rightarrow \mathtt{L}_t \times \texttt{ConstraintsSet}$

The function TYPE

TYPE: $\texttt{TypeAssumptions} \times \texttt{L} \to \texttt{L}_t \times \texttt{ConstraintsSet}$

TYPE$(\mathit{Ass}, \texttt{class A extends } \overline{\texttt{B}}\ \{\ \overline{\texttt{f}};\ \overline{\texttt{M}}\ \})$ = **let**

$\mathit{fass} := \{\ \texttt{this.f} : \delta^{\texttt{f}}_{\texttt{A}} \mid \texttt{f} \in \overline{\texttt{f}}\ \} \cup \{\ \texttt{this.f} : \texttt{T} \mid \texttt{T f} \in \mathit{fields}(\overline{\texttt{B}})\ \}$

The function TYPE

$$\textbf{TYPE}: \texttt{TypeAssumptions} \times \mathtt{L} \to \mathtt{L}_t \times \texttt{ConstraintsSet}$$

$$\textbf{TYPE}(\mathit{Ass}, \texttt{class A extends } \overline{\mathtt{B}}\ \{\ \overline{\mathtt{f}};\ \overline{\mathtt{M}}\ \}) = \textbf{let}$$

$$\mathit{fass} := \{\ \texttt{this.f} : \delta^{\mathtt{f}}_{\mathtt{A}} \mid \mathtt{f} \in \overline{\mathtt{f}}\ \} \cup \{\ \texttt{this.f} : \mathtt{T} \mid \mathtt{T\ f} \in \mathit{fields}(\overline{\mathtt{B}})\ \}$$

$$\mathit{mass} := \{\ \texttt{this.m} : \overline{\alpha^{\mathtt{m}}_{\mathtt{A}}} \to \gamma^{\mathtt{m}}_{\mathtt{A}} \mid \ \mathtt{m}(\overline{\mathtt{x}})\{\ \texttt{return e};\ \} \in \overline{\mathtt{M}}\ \}$$

$$\cup \{\ \texttt{this.m} : \overline{\mathit{aty}} \to \mathit{rty} \mid \ \mathit{mtype}(\mathtt{m}, \overline{\mathtt{B}}) = \overline{\mathit{aty}} \to \mathit{rty}\ \}$$

The function TYPE

$$\textbf{TYPE}: \texttt{TypeAssumptions} \times \texttt{L} \to \texttt{L}_t \times \texttt{ConstraintsSet}$$

$$\textbf{TYPE}(\mathit{Ass}, \texttt{class A extends } \overline{\texttt{B}}\ \{\ \overline{\texttt{f}};\ \overline{\texttt{M}}\ \}) = \textbf{let}$$

$$\mathit{fass} := \{\ \texttt{this.f} : \delta^{\texttt{f}}_{\texttt{A}} \mid \texttt{f} \in \overline{\texttt{f}}\ \} \cup \{\ \texttt{this.f} : \texttt{T} \mid \texttt{T f} \in \mathit{fields}(\overline{\texttt{B}})\ \}$$

$$\mathit{mass} := \{\ \texttt{this.m} : \overline{\alpha^{\texttt{m}}_{\texttt{A}}} \to \gamma^{\texttt{m}}_{\texttt{A}} \mid \texttt{m}(\overline{\texttt{x}})\{\ \texttt{return e};\ \} \in \overline{\texttt{M}}\ \}$$

$$\cup \{\ \texttt{this.m} : \overline{\mathit{aty}} \to \mathit{rty} \mid \mathit{mtype}(\texttt{m}, \overline{\texttt{B}}) = \overline{\mathit{aty}} \to \mathit{rty}\ \}$$

$$\mathit{AssAll} = \mathit{Ass} \cup \mathit{fass} \cup \mathit{mass} \cup \{\ \texttt{this} : \texttt{A}\ \}$$

The function TYPE

TYPE: $\texttt{TypeAssumptions} \times \texttt{L} \to \texttt{L}_t \times \texttt{ConstraintsSet}$

TYPE$(\, Ass, \texttt{class A extends B } \{\, \overline{\texttt{f}};\ \overline{\texttt{M}} \,\}\,)$ = **let**

$fass := \{\, \texttt{this.f} : \delta^{\texttt{f}}_{\texttt{A}} \mid \texttt{f} \in \overline{\texttt{f}} \,\} \cup \{\, \texttt{this.f} : \texttt{T} \mid \texttt{T f} \in fields(\, \texttt{B} \,) \,\}$

$mass := \{\, \texttt{this.m} : \overline{\alpha^{\texttt{m}}_{\texttt{A}}} \to \gamma^{\texttt{m}}_{\texttt{A}} \mid \ \texttt{m}(\overline{\texttt{x}})\{\ \texttt{return e;}\ \} \in \overline{\texttt{M}} \,\}$

$\quad \cup \{\, \texttt{this.m} : \overline{aty} \to rty \mid \ mtype(\, \texttt{m}, \texttt{B} \,) = \overline{aty} \to rty \,\}$

$AssAll = Ass \cup fass \cup mass \cup \{\, \texttt{this} : \texttt{A} \,\}$

For $\texttt{m}(\overline{\texttt{x}})\{\ \texttt{return e;}\ \} \in \overline{\texttt{M}}$ {

$Ass = AssAll \cup \{\, \texttt{x}_j : \alpha^{\texttt{m,j}}_{\texttt{A}} \mid \overline{\texttt{x}} = \texttt{x}_1 \ldots \texttt{x}_{n_i} \,\}$

$(\texttt{e}_t : rty, C') =$ **TYPEExpr**$(\, Ass, \texttt{e} \,)$

$C = C \cup C'[\gamma^{\texttt{m}}_{\texttt{A}} \mapsto rty]\}$

$\overline{\texttt{M}}_t = \{\, rty \ \ \texttt{m}(\overline{\alpha^{\texttt{m}}_{\texttt{A}}\ \texttt{x}})\{\ \texttt{return e}_t\texttt{;}\ \} \mid \texttt{m}(\overline{\texttt{x}})\{\ \texttt{return e;}\ \} \in \overline{\texttt{M}} \,\}$

The function TYPE

TYPE: $\texttt{TypeAssumptions} \times \texttt{L} \to \texttt{L}_t \times \texttt{ConstraintsSet}$

TYPE$(\, Ass, \texttt{class A extends } \overline{\texttt{B}}\ \{\, \overline{\texttt{f}};\ \overline{\texttt{M}} \,\}\,) =$ **let**

$fass := \{\, \texttt{this.f} : \delta^{\texttt{f}}_{\texttt{A}} \mid \texttt{f} \in \overline{\texttt{f}} \,\} \cup \{\, \texttt{this.f} : \texttt{T} \mid \texttt{T f} \in fields(\, \overline{\texttt{B}} \,) \,\}$

$mass := \{\, \texttt{this.m} : \overline{\alpha^{\texttt{m}}_{\texttt{A}}} \to \gamma^{\texttt{m}}_{\texttt{A}} \mid \ \texttt{m}(\overline{\texttt{x}})\{\ \texttt{return e;}\ \} \in \overline{\texttt{M}} \,\}$
$\cup \{\, \texttt{this.m} : \overline{aty} \to rty \mid \ mtype(\, \texttt{m}, \overline{\texttt{B}} \,) = \overline{aty} \to rty \,\}$

$AssAll = Ass \cup fass \cup mass \cup \{\, \texttt{this} : \texttt{A} \,\}$

For $\texttt{m}(\overline{\texttt{x}})\{\ \texttt{return e;}\ \} \in \overline{\texttt{M}}$ {

$Ass = AssAll \cup \{\, \texttt{x}_j : \alpha^{\texttt{m,j}}_{\texttt{A}} \mid \overline{\texttt{x}} = \texttt{x}_1 \ldots \texttt{x}_{n_i} \,\}$

$(\texttt{e}_t : rty, C') =$ **TYPEExpr**$(\, Ass, \texttt{e} \,)$

$C = C \cup C'[\gamma^{\texttt{m}}_{\texttt{A}} \mapsto rty]\}$

$\overline{\texttt{M}}_t = \{\, rty \ \ \texttt{m}(\overline{\alpha^{\texttt{m}}_{\texttt{A}}\ \texttt{x}})\{\ \texttt{return e}_t\texttt{;}\ \} \mid \texttt{m}(\overline{\texttt{x}})\{\ \texttt{return e;}\ \} \in \overline{\texttt{M}} \,\}$

in

$(\texttt{class A extends } \overline{\texttt{B}}\ \{\, \overline{\delta_{\texttt{A}}\ \texttt{f}}; \overline{\texttt{M}_t} \,\}, C)$

Type constraints

- $c \lessdot c'$ means c *is a subtype of* c'.
- $\phi(\, c, f, c' \,)$ means c *provides a field* f *with type* c'.
- $\mu(\, c, m, \overline{c}, (c', \overline{c}') \,)$ means c *provides a method* m *applicable to arguments of type* $\overline{c}$*, with return type* c' *and parameters of type* $\overline{c}'$.

  $\mu(\, c, m, \overline{c}, (c', \overline{c}') \,)$ implicitly includes the constraints $\overline{c} \leq \overline{c}'$.

TYPEExpr for Method-call

$$\textbf{TYPEExpr}(\, Ass, \mathtt{e}_0.\mathtt{m}(\overline{e})\,) = \textbf{let}$$
$$(\mathtt{e}_{0_t} : ty_0, C_0) = \textbf{TYPEExpr}(\, Ass, \mathtt{e}_0\,)$$
$$(\overline{\mathtt{e}_t : ty}, \overline{C}) = \textbf{TYPEExpr}(\, Ass, \overline{\mathtt{e}}\,)$$

TYPEExpr for Method-call

TYPEExpr($Ass, \mathtt{e}_0.\mathtt{m}(\overline{e})$) = **let**

$(\mathtt{e}_{0_t} : ty_0, C_0)$ = **TYPEExpr**($Ass, \mathtt{e}_0$)

$(\overline{\mathtt{e}_t : ty}, \overline{C})$ = **TYPEExpr**($Ass, \overline{\mathtt{e}}$)

in

if (ty_0 *is a type variable*) **then**

$$((\mathtt{e}_{0_t} : ty_0).\mathtt{m}(\overline{e_t : ty}) : \gamma^{\mathtt{m}}_{\mathtt{ty}_0}, (C_0 \cup \overline{C}) \cup \{\, \mu(\, ty_0, \mathtt{m}, \overline{ty}, (\gamma^{\mathtt{m}}_{\mathtt{ty}_0}, \overline{\beta^{\mathtt{m}}_{\mathtt{ty}_0}})\,)\,\})$$

TYPEExpr for Method-call

$$\textbf{TYPEExpr}(\, Ass, \mathtt{e}_0.\mathtt{m}(\overline{e})\,) = \textbf{let}$$
$$(\mathtt{e}_{0_t} : ty_0, C_0) = \textbf{TYPEExpr}(\, Ass, \mathtt{e}_0\,)$$
$$(\overline{\mathtt{e}_t : ty}, \overline{C}) = \textbf{TYPEExpr}(\, Ass, \overline{\mathtt{e}}\,)$$
$$\textbf{in}$$
$$\textbf{if}\ (ty_0\ \textit{is a type variable})\ \textbf{then}$$
$$((\mathtt{e}_{0_t} : ty_0).\mathtt{m}(\overline{e_t : ty}) : \gamma^{\mathtt{m}}_{\mathtt{ty}_0}, (C_0 \cup \overline{C}) \cup \{\, \mu(\, ty_0, \mathtt{m}, \overline{ty}, (\gamma^{\mathtt{m}}_{\mathtt{ty}_0}, \overline{\beta^{\mathtt{m}}_{\mathtt{ty}_0}})\,)\,\})$$
$$\textbf{else if}(ty_0 \in Ass)\ \&\&\ (mtype(\,\mathtt{m}, ty_0\,) = \overline{aty} \to rty)\ \textbf{then}$$
$$((\mathtt{e}_{0_t} : ty_0).\mathtt{m}(\overline{e_t : ty}) : rty, (C_0 \cup \overline{C}) \cup \{\, \overline{ty \lessdot aty}\,\})$$

The function construct

1. For any $\phi(\ ty1, \texttt{f}, \delta\ )$ and $\mu(\ ty2, \texttt{m}, \overline{\alpha}, (\gamma, \overline{\beta})\ ) \in C$ interfaces are generated:

```
interface ty1< ...,δ, ...>  {
    δ f;
}

interface ty2 <...,γ, β̄, ...> {
    γ m(β̄ x1);
}
```

2. fresh type variables ν_1, ν_2 and constraints are introduced:

$$\nu_1 \lessdot \texttt{ty1<...>},$$

$$\nu_2 \lessdot \texttt{ty2<...>}$$

3. the type variables ν_1, ν_2 are introduced as generics in the class:

$$\texttt{class A<} \ldots, \nu_1, \nu_2, \ldots \texttt{> extends} \ldots$$

The function solve

solve: $\mathrm{L}_t \to \mathrm{L}_t$

solve($\texttt{class A<}\overline{\texttt{T}}\texttt{>[}\overline{ty \lessdot ty'}\texttt{]}$ $\texttt{extends}$ $\overline{\texttt{B}}$ $\{\overline{\texttt{ty f;}}$ $\overline{\texttt{M}_\texttt{t}}\}$) =

$$\texttt{class A<...>[...] extends } \overline{\texttt{B}} \{\overline{\sigma(\texttt{ty})\ \texttt{f}};\ \overline{\sigma(\texttt{M}_\texttt{t})}\},$$

with

- σ = **TypeUnify**($\overline{ty \lessdot ty'}$) :

  For the pairs $\overline{ty \lessdot ty'}$ a substitution σ is determined such that:

$$\overline{\sigma(ty) \leq^* \sigma(ty')}$$

- $\leq^*$ is the subtyping relation.

Example (input syntax)

```
class A {

    mt(x, y, z) {
        return x.sub(y).add(z);
    }
}
```

Result of TYPE

```
mt(x, y, z) {
    return x.sub(y).add(z);
}
```

```
class A {

    γ^add_{γ^sub_{α_A^{mt,1}}}  mt(α_A^{mt,1} x, α_A^{mt,2} y, α_A^{mt,3} z) {

        return e_t;
    }
}
```

with e_t = $[[[\mathtt{x} : \alpha_A^{mt,1}].\mathtt{sub}([\mathtt{y} : \alpha_A^{mt,2}]) : \gamma^{sub}_{\alpha_A^{mt,1}}].\mathtt{add}(\mathtt{z} : \alpha_A^{mt,3}) : \gamma^{add}_{\gamma^{sub}_{\alpha_A^{mt,1}}}]$

Result of TYPE

```
mt(x, y, z) {
    return x.sub(y).add(z);
}
```

```
class A {

    γ^{add}_{γ^{sub}_{α_A^{mt,1}}}  mt(α_A^{mt,1} x, α_A^{mt,2} y, α_A^{mt,3} z) {

        return e_t;
    }
}
```

with e_t = $[[[\mathtt{x} : \alpha_{\mathtt{A}}^{\mathtt{mt},1}].\mathtt{sub}([\mathtt{y} : \alpha_{\mathtt{A}}^{\mathtt{mt},2}]) : \gamma^{\mathtt{sub}}_{\alpha_{\mathtt{A}}^{\mathtt{mt},1}}].\mathtt{add}(\mathtt{z} : \alpha_{\mathtt{A}}^{\mathtt{mt},3}) : \gamma^{\mathtt{add}}_{\gamma^{\mathtt{sub}}_{\alpha_{\mathtt{A}}^{\mathtt{mt},1}}}]$

and

$$C = \{\, \mu(\, \alpha_{\mathtt{A}}^{\mathtt{mt},1}, \mathtt{sub}, \alpha_{\mathtt{A}}^{\mathtt{mt},2}, (\gamma^{\mathtt{sub}}_{\alpha_{\mathtt{A}}^{\mathtt{mt},1}}, \beta^{\mathtt{sub},1}_{\alpha_{\mathtt{A}}^{\mathtt{mt},1}})\,),$$
$$\mu(\, \gamma^{\mathtt{sub}}_{\alpha_{\mathtt{A}}^{\mathtt{mt},1}}, \mathtt{add}, \alpha_{\mathtt{A}}^{\mathtt{mt},3}, (\gamma^{\mathtt{add}}_{\gamma^{\mathtt{sub}}_{\alpha_{\mathtt{A}}^{\mathtt{mt},1}}}, \beta^{\mathtt{add},1}_{\gamma^{\mathtt{sub}}_{\alpha_{\mathtt{A}}^{\mathtt{mt},1}}}))\,\}$$

The result of construct (output syntax)

```
interface α_A^{mt,1} <Gamma_m, Beta_m> { Gamma_m sub(Beta_m x); }

interface γ^{sub}_{α_A^{mt,1}} <Gamma_n, Beta_n> { Gamma_n add(Beta_n x); }

class A <ν1,ν2,ν3,ν4,ν5,ν6,ν7>
        [ν3 ◁ ν5, ν4 ◁ ν7, ν1 ◁ α_A^{mt,1}<ν2,ν5>, ν2 ◁ γ^{sub}_{α_A^{mt,1}}<ν6,ν7>]{
    ν6 mt(ν1 x, ν3 y, ν4 z) {
        return x.sub(y).add(z);
    }
}
```

The result of construct (output syntax)

```
interface α_A^{mt,1} <Gamma_m, Beta_m> { Gamma_m sub(Beta_m x); }

interface γ^{sub}_{α_A^{mt,1}} <Gamma_n, Beta_n> { Gamma_n add(Beta_n x); }

class A <ν1,ν2,ν3,ν4,ν5,ν6,ν7>
        [ν3 ◁ ν5, ν4 ◁ ν7, ν1 ◁ α_A^{mt,1}<ν2,ν5>, ν2 ◁ γ^{sub}_{α_A^{mt,1}}<ν6,ν7>]{
    ν6 mt(ν1 x, ν3 y, ν4 z) {
        return x.sub(y).add(z);
    }
}
```

The application of **solve** changes nothing.

Instance Example (input syntax)

```
class myInteger extends α_A^mt,1<myInteger, myInteger>,
                        γ^sub_α_A^mt,1<myInteger, myInteger> {

    Integer i;

    myInteger sub(myInteger x) {
        return new myInteger(i - x.i);
    }

    myInteger add(myInteger x) {
        return new myInteger(i + x.i);
    }
}
```

```
class A <ν1,ν2,ν3,ν4,ν5,ν6,ν7>
        [ν3 ⋖ ν5, ν4 ⋖ ν7, ν1 ⋖ α_A^{mt,1}<ν2,ν5>, ν2 ⋖ γ^{sub}_{α_A^{mt,1}}<ν6,ν7>]{

    ν6 mt(ν1 x, ν3 y, ν4 z) {
        return x.sub(y).add(z);
    }
}
```

```
class Main {

    main() {
        return new A<ν1=myInteger, ν2=myInteger>()
                .mt(new myInteger(2),
                    new myInteger(1),
                    new myInteger(3));
    }
}
```

The result of TYPE

$$
\begin{aligned}
C = \{ \quad & \nu_3 \lessdot \nu_5, \\
& \nu_4 \lessdot \nu_7, \\
& \texttt{myInteger} \lessdot \alpha_{\texttt{A}}^{\texttt{mt},1}\texttt{<myInteger},\nu_5\texttt{>}, \\
& \texttt{myInteger} \lessdot \gamma_{\alpha_{\texttt{A}}^{\texttt{mt},1}}^{\texttt{sub}}\texttt{<}\nu_6,\nu_7\texttt{>}, \\
& \texttt{myInteger} \lessdot \nu_3, \\
& \texttt{myInteger} \lessdot \nu_4 \}
\end{aligned}
$$

The result of TYPE

$$
\begin{aligned}
C = \{ \quad & \nu_3 \lessdot \nu_5, \\
& \nu_4 \lessdot \nu_7, \\
& \texttt{myInteger} \lessdot \alpha_{\texttt{A}}^{\texttt{mt},1}\texttt{<myInteger},\nu_5\texttt{>}, \\
& \texttt{myInteger} \lessdot \gamma_{\alpha_{\texttt{A}}^{\texttt{mt},1}}^{\texttt{sub}}\texttt{<}\nu_6,\nu_7\texttt{>}, \\
& \texttt{myInteger} \lessdot \nu_3, \\
& \texttt{myInteger} \lessdot \nu_4 \}
\end{aligned}
$$

construct adds no interfaces, as there is no call of abstract fields or methods.

The result of solve

$$\begin{aligned}\sigma = \{\, & \nu_5 \mapsto \texttt{myInteger}, \\ & \nu_6 \mapsto \texttt{myInteger}, \\ & \nu_7 \mapsto \texttt{myInteger}, \\ & \nu_3 \mapsto \texttt{myInteger}, \\ & \nu_4 \mapsto \texttt{myInteger} \,\}\end{aligned}$$

The result of solve

$$\sigma = \{\, \nu_5 \mapsto \texttt{myInteger},\ \nu_6 \mapsto \texttt{myInteger},\ \nu_7 \mapsto \texttt{myInteger},\ \nu_3 \mapsto \texttt{myInteger},\ \nu_4 \mapsto \texttt{myInteger} \,\}$$

```
class Main {

    myInteger main() {
        return new A<myInteger, myInteger, myInteger, myInteger,
                     myInteger, myInteger, myInteger>()
                .mt(new myInteger(2),
                    new myInteger(1),
                    new myInteger(3));
    }
}
```

Summary

Conclusion

We have presented a type inference algorithm for a Java-like language.

- type-less Java classes independently from any environment
- separate compilation of Java classes without relying on type informations of other classes.
- infers structural types, given as generated interfaces

Summary

Conclusion

We have presented a type inference algorithm for a Java-like language.

- type-less Java classes independently from any environment
- separate compilation of Java classes without relying on type informations of other classes.
- infers structural types, given as generated interfaces

Ongoing work

- complete **construct** and **solve** algorithms
- soundness and completeness proof