Polymorphic Bytecode

Sophia Drossopoulou (Imperial College London)

joint work with Davide Ancona and Elena Zucca (Univ. Genova),
Ferruccio Damiani (Univ. Torino), and Alex Buckley (Imperial College)

Contents:

1. The Problem

Java/C# compilers annotate the bytecode with information about compilation environment –runtime environment may differ from compilation environment

Java/C# compilers annotate bytecode with information about the compilation environment. In particular, a field access is annotated with the class containing the field, and the type of the field. This annotation is used for 1) field resolution and 2) to help the verifier.

E.g. the souce

$$S = \text{class A\{ D m(B x)\{ return x.f.g; \} \}}$$

compiled in environment Δ_1

$$\Delta_1 = \text{class B\{ C f\} class C\{ D g \}}$$

produces a bytecode corresponding to:

$$B_1 = \text{class A\{ D m(B x)\{ return x[B.f C][C.g D]; \}}$$

Runtime environment may differ from compilation environment.
If difference small, then OK; if difference large, then runtime ERROR.

From previous, S = class A{ D m(B x){ return x.f.g; } }
compiled in Δ_1 = class B{ C f} class C{ D g }
produces B_1 = class A{ D m(B x){ return x[B.f C][C.g D]; }

If B_1 is run in a context
of Δ_1 then OK.
of Δ_2 = class B {C f; D g } class C extends B { } then OK.
of Δ_3 = class B{ E f} class E{ D g } then ERROR.

Davide: Isn't that annoying?
Sophia: It is unavoidable.
Davide/Elena: It *is* avoidable!

2. Overview of the Solution

Davide's aim: Compositional compilation

1. decouple compilation from the particular compilation environment as far as possible.
2. make bytecode easily retargetable.

Aims are achieved through:

1. Compilation produces annotations with *type variables* rather than types; type variables can be replaced later;
2. Compilation does not check the presence of members, (or subtypes); instead, it produces *constraints*.

Thus, compilation is context *independent*.

Global compilation will have format: $\Delta \vdash_{GL} S : \Delta' \parallel B$

Compositional compilation will have format: $\vdash_{CO} S : \Delta \parallel \Gamma \parallel B$.

In terms of our example, take

S = class A{ D m(B x){ return x.f.g; }

Δ = class A{ D m(B) }

Δ_1 = class B{ C f} class C{ D g }

B_1 = class A{ D m(B x){ return x[B.f C][C.g D]; } }

Then, traditional, global compilation gives

$$\Delta_1 \vdash_{GL} S : \Delta \parallel B_1$$

Now, take polymorphic bytecode B_2 and constraints Γ:

B_2 = class A{ D m(B x){ return x[B.f α][α.g α']; } }

Γ = fld(B.f α), fld(α.g α'), α'≤D

Then, compositional compilation will give

$$\vdash_{CO} S : \Delta \parallel \Gamma \parallel B_2.$$

In order to execute the polymorphic bytecode we can:

1. consider a linking step, where type variables are replaced by class names before execution – linking.

or

2. extend the virtual machine so that type variables are replaced at runtime – flexible dynamic linking (Alex Buckley).

In this talk we take the first approach.

Taking the first approach, we will define a linking step:

$$\Gamma \parallel B \rightsquigarrow \Gamma' \parallel B' \ \text{in}\ \Delta$$

uses the context Δ; resolves the constraints in Γ; applies the solution to Γ, obtaining remaining constraints Γ' and more defined binary B'.

For example, take

$B_2 =$ class A{ D m(B x){ return x[B.f α][α.g α']; } }

$\Gamma =$ fld(B.f α), fld (α.g α'), $\alpha' \leq D$

$\Delta_3 =$ class B{ C f}

$B_3 =$ class A{ D m(B x){ return x[B.f C][C.g α']; } }

$\Gamma_3 =$ fld (C.g α'), $\alpha' \leq D$

Then, linking will give

$$\Gamma \parallel B_2 \rightsquigarrow \Gamma_3 \parallel B_3 \ \text{in}\ \Delta_3$$

Furthermore, take

B_3 = class A{ D m(B x){ return x[B.f C][C.g α']; } }

Γ_3 = fld (C.g α'), $\alpha' \leq D$

Δ_4 = class C{ H g} class H extends D { }

B_4 = class A{ D m(B x){ return x[B.f C][C.g H]; } }

Then, linking will give

$\Gamma_3 \parallel B_3 \leadsto \epsilon \parallel B_4$ in Δ_4

and also

$\Gamma \parallel B_2 \leadsto \epsilon \parallel B_4$ in $\Delta_3\Delta_4$

What about the relation between global compilation, compositional compilation, and linking?

In our example, we have

- $\Delta_1 \vdash_{GL} S : \Delta \parallel B_1$ (global compilation),
- $\vdash_{CO} S : \Delta \parallel \Gamma \parallel B_2$ (compositional compilation),
- $\Gamma \parallel B_2 \rightsquigarrow \epsilon \parallel B_1$ in Δ_1 (linking).

We shall require that compositional compilation followed by linking that resolves all constraints is “equivalent” to global compilation.

end of overview of the solution

Structure of the talk:

3. A language independent framework for compilation

Global Compilation	$\Delta \vdash_{gl} s : \delta \parallel b$	for one fragment
	$\Delta \vdash_{GL} S : \Delta \parallel B$	for many fragments

where	δ, Δ	one, many class signatures
	s, S	one, many source fragments
	b, B	one, many binary fragments

Compositional Compilation	$\vdash_{co} s : \delta \parallel \Gamma \parallel b$	one fragm.
	$\vdash_{CO} S : \Delta \parallel \Gamma \parallel B$	many frgms

where γ, Γ one, many constraints

Linking $\Gamma \parallel B \rightsquigarrow \Gamma' \parallel B'$ in Δ

Our framework is parametric wrt. compilation of *one* fragment.
In other words,

- Global compilation of *one* fragment,

  $$\Delta \vdash_{gl} s : \delta \parallel b,$$

  language dependent, outside the framework.
- Global compilation of many fragments,

  $$\Delta \vdash_{GL} S : \Delta \parallel B,$$

  part of our framework; defined in terms of $\Delta \vdash_{gl} s : \delta \parallel b$.
- Compositional compilation of *one* fragment,

  $$\vdash_{co} s : \delta \parallel \Gamma \parallel b,$$

  language dependent, and outside the framework.
- Global compilation of many fragments,

  $$\vdash_{CO} S : \Delta \parallel \Gamma \parallel B,$$

  part of framework; defined in terms of $\vdash_{co} s : \delta \parallel \Gamma \parallel b$.

Global Compilation

Global compilation for one fragment, $\Delta \vdash_{gl} s : \delta \parallel b$
given by the particular programming language.

Global compilation for many fragments:

$$\frac{\Delta \vdash_{gl} s : \delta \parallel b}{\Delta \vdash_{GL} s : \delta \parallel b}$$

$$\frac{\Delta\ \Delta_1 \ldots \Delta_{k-1}\ \Delta_{k+1} \ldots \Delta_n \vdash_{GL} S_k : \Delta_k \parallel B_k \qquad \text{for } k \in 1..n}{\Delta \vdash_{GL} S_1 \ldots S_n : \Delta_1 \ldots \Delta_n \parallel B_1 \ldots B_1}$$

where δ, Δ class signatures, s, S source fragments
b, B binary fragments

Compositional Compilation

Compositional compilation for one fragment, $\vdash_{co}\ s : \delta \parallel \Gamma \parallel b$, given by the particular programming language.

Compositional compilation for many fragments:

$$\frac{\vdash_{co}\ s : \delta \parallel \Gamma \parallel b \qquad \Gamma \parallel b \rightsquigarrow \Gamma' \parallel B' \text{ in } \Delta}{\vdash_{co}\ s : \delta \parallel \Gamma' \parallel b'}$$

$$\frac{\vdash_{co}\ S_k : \Delta_k \parallel \Gamma_k \parallel B_k \quad \text{for } k \in 1..n \qquad \Gamma_1 \ldots \Gamma_n \parallel B_1 \ldots B_n \rightsquigarrow \Gamma' \parallel B' \text{ in } \Delta_1 \ldots \Delta_n}{\vdash_{co}\ S_1 \ldots S_n : \Delta_1 \ldots \Delta_n \parallel \Gamma' \parallel B'}$$

where δ, Δ class signatures, γ, Γ constraints,
s, S source fragments b, B binary fragments

Sound and Complete Compositional Compilation

Definition Compositional compilation is *sound*, iff

$$\left.\begin{array}{l} \vdash_{CO}\ S : \Delta \parallel \Gamma \parallel B, \\ \text{and} \\ \Gamma \parallel B \rightsquigarrow \epsilon \parallel B' \ \text{ in } \Delta\,\Delta' \end{array}\right\} \Rightarrow \quad \Delta\,\Delta' \vdash_{GL}\ : S : \Delta \parallel B'$$

Definition Compositional compilation is *complete*, iff

$$\Delta' \vdash_{GL}\ S : \Delta \parallel B \quad \left.\right\} \Rightarrow \begin{array}{l} \exists\, \Gamma', B': \\ \vdash_{CO}\ S : \Delta \parallel \Gamma' \parallel B', \\ \text{and} \\ \Gamma' \parallel B' \rightsquigarrow \epsilon \parallel B \ \text{ in } \Delta\,\Delta' \end{array}$$

Note, that for $\Delta' = \epsilon$, we obtain that sound and complete means:

$$\vdash_{CO}\ S : \Delta \parallel \epsilon \parallel B \iff \vdash_{GL}\ S : \Delta \parallel B$$

Theorem 1 (Sufficient Conditions for Soundness)

If

1. $\left.\begin{array}{l} \vdash_{co}\ s : \delta \parallel \Gamma \parallel b, \text{ and} \\ \Gamma \parallel b \rightsquigarrow \epsilon \parallel b' \text{ in } \Delta\delta \end{array}\right\} \Rightarrow \quad \Delta\delta \vdash_{gl} s : \delta \parallel b'$

2. $\left.\begin{array}{l} \Gamma \parallel B \rightsquigarrow \Gamma' \parallel B' \text{ in } \Delta, \text{ and} \\ \Gamma' \parallel B' \rightsquigarrow \epsilon \parallel B'' \text{ in } \Delta' \Delta \end{array}\right\} \Rightarrow \quad \Gamma \parallel B \rightsquigarrow \epsilon \parallel B'' \text{ in } \Delta'\Delta$

3. $\Gamma_1 \ldots \Gamma_n \parallel B_1 \ldots B_n \rightsquigarrow \epsilon \parallel B' \text{ in } \Delta \Rightarrow \left\{\begin{array}{l} \exists\, B_1', \ldots, B_n': \\ \quad B' = B_1', \ldots, B_n' \text{ and} \\ \quad \Gamma_k \parallel B_k \rightsquigarrow \epsilon \parallel B_k' \text{ in } \Delta \end{array}\right.$

4. $\epsilon \parallel B \rightsquigarrow \epsilon \parallel B \text{ in } \epsilon$ for all B

then compositional compilation is sound.

1st Condition

$$\left.\begin{array}{l} \vdash_{co}\ s : \delta \parallel \Gamma \parallel b,\ \text{and} \\ \Gamma \parallel b \leadsto \epsilon \parallel b' \ \text{in}\ \Delta\delta \end{array}\right\} \Rightarrow \quad \Delta\delta \vdash_{gl} s : \delta \parallel b'$$

means that compositional compilation "in the small" is sound.

2nd Condition

$$\left.\begin{array}{l} \Gamma \parallel B \leadsto \Gamma' \parallel B' \ \text{in}\ \Delta,\ \text{and} \\ \Gamma' \parallel B' \leadsto \epsilon \parallel B'' \ \text{in}\ \Delta'\,\Delta \end{array}\right\} \Rightarrow \quad \Gamma \parallel B \leadsto \epsilon \parallel B'' \ \text{in}\ \Delta'\Delta$$

means that two linking steps, with the second step in a larger environment $\Delta'\,\Delta$ resolving all constraints,
correspond to one linking step in a larger environment resolving all constraints.

3rd Condition

$$\Gamma_1 \ldots \Gamma_n \parallel B_1 \ldots B_n \rightsquigarrow \epsilon \parallel B' \text{ in } \Delta \quad \Rightarrow \quad \begin{cases} \exists\, B_1', \ldots, B_n' : \\ \quad B' = B_1', \ldots, B_n' \text{ and} \\ \quad \Gamma_k \parallel B_k \rightsquigarrow \epsilon \parallel B_k' \text{ in } \Delta \\ \qquad\qquad k \in 1..n \end{cases}$$

means that linking a sequence of binaries $B_1 \ldots B_n$ resolving all constraints,
correspond to a sequence of linking binary B_k and each step resolving all constraints.

4th Condition

$$\epsilon \parallel B \rightsquigarrow \epsilon \parallel B \text{ in } \epsilon \quad \text{for all } B$$

means that linking in an empty environment, and empty constraints has no effect.

Theorem 2 (Sufficient Conditions for Completeness)

If

1. $\Delta\delta \vdash_{gl} s : \delta \parallel b \Rightarrow \left\{ \begin{array}{l} \exists\, b', \Gamma: \\ \vdash_{co} s : \delta \parallel \Gamma \parallel b', \\ \Gamma \parallel b' \rightsquigarrow \epsilon \parallel b' \text{ in } \Delta\delta \end{array} \right.$

2. $\Gamma \parallel B \rightsquigarrow \epsilon \parallel B' \text{ in } \Delta'\Delta \Rightarrow \left\{ \begin{array}{l} \exists\, B'', \Gamma'': \\ \Gamma \parallel B \rightsquigarrow \Gamma'' \parallel B'' \text{ in } \Delta, \\ \Gamma'' \parallel B'' \rightsquigarrow \epsilon \parallel B' \text{ in } \Delta'\Delta \end{array} \right.$

3. $\left. \begin{array}{l} B' = B_1',\ldots,B_n' \text{ and} \\ \Gamma_k \parallel B_k \rightsquigarrow \epsilon \parallel B_k' \text{ in } \Delta \end{array} \right\} \Rightarrow \Gamma_{1\ldots}\Gamma_n \parallel B_1\ldots B_n \rightsquigarrow \epsilon \parallel B' \text{ in } \Delta$

then compositional compilation is compete.

1st Condition – Complete

$$\Delta\delta \vdash_{gl} s : \delta \parallel b \quad \Rightarrow \begin{cases} \exists\, b', \Gamma: \\ \vdash_{co} s : \delta \parallel \Gamma \parallel b', \\ \Gamma \parallel b' \rightsquigarrow \epsilon \parallel b' \text{ in } \Delta\delta \end{cases}$$

means that compositional compilation “in the small” is compete.

2nd Condition – Complete

$$\Gamma \parallel B \rightsquigarrow \epsilon \parallel B' \text{ in } \Delta'\Delta \quad \Rightarrow \begin{cases} \exists\, B'', \Gamma'': \\ \Gamma \parallel B \rightsquigarrow \Gamma'' \parallel B'' \text{ in } \Delta, \\ \Gamma'' \parallel B'' \rightsquigarrow \epsilon \parallel B' \text{ in } \Delta'\Delta \end{cases}$$

means that one linking step which resolves all constraints, can be broken down into two steps, the first in a smaller environment.

3rd Condition – Complete

$$\left.\begin{array}{l} B' = B_1',\ldots,B_n' \ \text{ and} \\ \Gamma_k \ \| \ B_k \rightsquigarrow \epsilon \ \| \ B_k' \text{ in } \ \Delta \end{array}\right\} \Rightarrow \quad \Gamma_1\ldots\Gamma_n \ \| \ B_1\ldots B_n \rightsquigarrow \epsilon \ \| \ B' \text{ in } \ \Delta$$

means that a sequence of linking steps which resolves all constraints, can be subsumed in one step.

Seeking sound and complete compositional compilation …

FJ0 Source Syntax

$S ::= s_1 \ldots s_n$

$s ::=$ class c extends c’ { fd md^S }

$fd ::=$ c f

$md^S ::=$ c m(c’ x){ return e^S; }

$e^S ::= x \mid e^S.f \mid e^S m(e^S) \mid$ new $c(e^S{}_1 \ldots e^S{}_n) \mid (c)e^S$

Notes

1. Superscripts distinguish source/binary, eg e^S vs e^B.
2. One field, one method per class.
3. One parameter, x, per method.
4. No imperative features.
5. Cast expression $(c)e^S$.
6. No overloading

where 2–4 not a restriction, 5 extra to FJ, 6 as in FJ.

FJ0 Binary Syntax

```
B     ::=  b1 … bn
b     ::=  class c extends c' { fd mdB }
fd    ::=  c f
mdB   ::=  c m(c' x){ return eB; }
eS    ::=  x  |
           eB[c.f c']   |               field f from class c, type c'
           eB [c.m(c') c"](eB) |
                                        meth m from class c,type c'→ c"
           new [c c1…cn](eB1…eBn)   |
                                         constr. for class c, fld types c1,…,cn
           <<c>>eB                       potential cast
```

$B ::= b_1 \ldots b_n$

$b ::=$ class c extends c' { fd md^B }

$fd ::= c\ f$

$md^B ::= c\ m(c'\ x)$\{ return e^B; \}

$e^S ::= x \ |$

$e^B[c.f\ c'] \ |$ — field f from class c, type c'

$e^B\,[c.m(c')\ c''](e^B) \ |$ — meth m from class c,type $c' \rightarrow c''$

new $[c\ c_1 \ldots c_n](e^B_1 \ldots e^B_n) \ |$ — constr. for class c, fld types $c_1, \ldots, c_n$

$<<c>>e^B$ — potential cast

FJ Class Signatures

Δ ::= $\delta_1 \ldots \delta_n$

δ ::= class c extends c' { c'' f   c''' m(c'''' x) }

Constraints

Γ	::=	$\gamma_1 \ldots \gamma_n$	
γ	::=	$c \leq c'$ |	class c is a subclass of c'
		fld(c.f c') |	class c has field f of type c'
		mth(c.m(c') c"] |	class c has method m of type $c' \rightarrow c''$
		fldTypes(c $c_1 \ldots c_n$) |	the fields of class c have types $c_1, \ldots, c_n$

The judgment

$$\Delta \vdash \gamma$$

means that the environment Δ satisfies constraint γ .

We skip the details here, but e.g. take

Δ_2 = class B { D g … } class C extends B { }

then

$\Delta_2 \vdash C \leq B$ and $\Delta_2 \vdash$ fld(C.g D)

We also introduce local variable declararions, Π, which maps this and x to a type.

We define $\Delta \vdash_{gl} s : \delta \parallel b$ in terms of the judgments

1. $\Delta \vdash \gamma$

   ie the environment Δ satisfies constraint γ.

2. $\Delta, \Pi \vdash_{gl} e^S : t \parallel e^b$.

   i.e. e^S has type , and compliation produces e^b.

Global Compilation Rules – in the small

$$\frac{\Pi(x) = t}{\Delta, \Pi \vdash_{gl} x : t \parallel x}$$

$$\frac{\Delta, \Pi \vdash_{gl} e^S : c \parallel e^B \qquad \Delta \vdash fld(c.f, c')}{\Delta, \Pi \vdash_{gl} e^S.f : c' \parallel e^B[c.f,c']}$$

$$\frac{\Delta, \Pi \vdash_{gl} e^S : c \parallel e^B \qquad \Delta \vdash mth(c.m(c'),c'') \qquad \Delta, \Pi \vdash_{gl} e_1{}^S : c''' \parallel e_1{}^B \qquad \Delta \vdash c''' \leq c'}{\Delta, \Pi \vdash_{gl} e^S .m(e_1{}^S) : c'' \parallel e^B[c.m(c'), c''] (e_1{}^B)}$$

$$\frac{\Delta \vdash fldTypes(c, c_1 \dots c_n) \qquad \Delta, \Pi \vdash_{gl} e_k{}^S : c_k \parallel e_k{}^B \quad k=1..n}{\Delta, \Pi \vdash_{gl} new\ c(e_1{}^S \dots e_n{}^S) : c \parallel new\ [c\ c_1 \dots c_n](e^B{}_1 \dots e^B{}_n)}$$

casts later

Global Compilation Rules - in the large

$$\frac{\Delta,\ \text{this}\to c,\ x\to c''' \ \vdash_{gl}\ e^S : c'' \ \|\ e'^B \qquad \Delta \ \vdash c''' \leq c'}{\Delta \ \vdash_{gl}\ \text{class } c \text{ extends } c' \{\ fd\ c''\ m(c''')\{\ e^S\}\ \} : (c, c', fd, c''\ m(c''')) \ \|\ \text{class } c \text{ extends } c' \{\ fd\ c''\ m(c''')\{\ e^S\ \}}$$

Compositional Compilation Rules – in the small

$$\frac{\Pi(x) = t}{\Pi \vdash_{co} x : t \parallel \epsilon \parallel x}$$

$$\frac{\Pi \vdash_{co} e^S : t \parallel \Gamma \parallel e^B \quad \alpha \text{ is a fresh type variable}}{\Pi \vdash_{co} e^S.f : \alpha \parallel \Gamma, fld(t.f, \alpha) \parallel e^B[t.f,\alpha]}$$

$$\frac{\Pi \vdash_{co} e^S : t \parallel \Gamma \parallel e^B \quad \Pi \vdash_{co} e_1{}^S : t' \parallel \Gamma' \parallel e_1{}^B \quad \alpha, \alpha' \text{ are fresh type variables}}{\Pi \vdash_{co} e^S.m(e_1{}^S) : c'' \parallel \Gamma, \Gamma', mth(t.m(\alpha)\ \alpha'), t' \leq \alpha \parallel e^B[t.m(\alpha), \alpha'](e_1{}^B)}$$

$$\frac{\Pi \vdash_{co} e_k{}^S : t_k \parallel \Gamma_k \parallel e_k{}^B \quad k=1..n \qquad \alpha_k \text{ are fresh type variables} \quad k=1..n}{\Pi \vdash_{co} new\ c(e_1{}^S \ldots e_n{}^S) : c \parallel \Gamma_1..\Gamma_n, fldTyps(c, \alpha_1..\alpha_n), t_1 \leq \alpha_1, \ldots t_n \leq \alpha_n \parallel new\ [c\ c_1 \ldots c_n](e^B{}_1 \ldots e^B{}_n)}$$

casts later

Compositional Compilation Rules – in the large

$$\frac{\text{this}\to c,\ x\to c''' \vdash_{co} e^S : t \parallel \Gamma \parallel e^B}{\begin{array}{c}\Delta \vdash_{co} \text{class } c \text{ extends } c' \{\ \text{fd } c''\ m(c''')\{\ e^S\ \}\ : (c, c', \text{fd}, c''\ m(c''')) \parallel \\ \Gamma, t \leq c'' \parallel \text{class } c \text{ extends } c' \{\ \text{fd } c''\ m(c''')\{\ e^S\ \}\end{array}}$$

Comparison of the global and compositional systems ..

in terms of the rule for field access

$$\frac{\Delta, \Pi \vdash_{gl} e^S : c \parallel e^B \qquad \Delta \vdash \mathrm{fld}(c.f, c')}{\Delta, \Pi \vdash_{gl} e^S.f : c' \parallel e^B[c.f,c']}$$

$$\frac{\Pi \vdash_{co} e^S : t \parallel \Gamma \parallel e^B \qquad \alpha \text{ is a fresh type variable}}{\Pi \vdash_{co} e^S.f : \alpha \parallel \Gamma, \mathrm{fld}(t.f, \alpha) \parallel e^B[t.f,\alpha]}$$

- Use of type variables in compositional
- Constraints consumed in the global system vs the constraints are produced in the compositional

4.4 Linking

We are looking for a relation $\Gamma \parallel B \leadsto \Gamma' \parallel B'$ in Δ so that the requirements from Theorem 1 and 2 will be satisfied.

Idea: the linking process replaces type variables in B by classes from Δ which satisfy the constraints from Γ.
Therefore, look for appropriate substitution σ and apply it to B.
Thus, assume a judgment $\Gamma \leadsto \Gamma' \parallel \sigma$ in Δ and define

$$\frac{\Gamma \leadsto \Gamma' \parallel \sigma \ \text{ in } \ \Delta}{\Gamma \parallel B \leadsto \Gamma' \parallel \sigma(B) \ \text{ in } \ \Delta}$$

3rd Theorem: if

1. $\Gamma \rightsquigarrow \Gamma' \parallel \sigma$ in Δ implies $\Delta \vdash \sigma(\Gamma) \setminus \Gamma'$.
2. $\Delta \vdash \sigma(\Gamma)$ implies $\Gamma \rightsquigarrow \epsilon \parallel \sigma$ in Δ.
3. $\Gamma \rightsquigarrow \epsilon \parallel \sigma$ in $\Delta_1\Delta_2$ implies

   $\Gamma \rightsquigarrow \Gamma_1 \parallel \sigma_1$ in Δ_1,

   $\Gamma_1 \rightsquigarrow \epsilon \parallel \sigma_2$ in $\Delta_1\Delta_2$

   $\sigma = \sigma_1\,\sigma_2$. for some σ_1, σ_2, Γ_1.

then, the requirements of theorems 1 and 2 are satisfied (and thus FJ0 compositional compilation is sound and complete.)

The search for substitutions is defined in terms of rules like

$$\frac{\Delta \models \mathrm{fld}(c.f\ c')}{\mathrm{fld}(c.f\ \alpha) \leadsto \alpha \mapsto c' \ \text{ in } \ \Delta} \qquad\qquad \frac{c \text{ is undefrined in } \Delta}{\mathrm{fld}(c.f\ t) \leadsto id \ \text{ in } \ \Delta}$$

$$\frac{\begin{array}{l} t \text{ cannot be unified with } c' \\ \Delta \models \mathrm{fld}(c.f\ c') \end{array}}{\mathrm{fld}(c.f\ t) \leadsto \text{ERROR} \parallel \epsilon \ \text{ in } \ \Delta}$$

$$\frac{\begin{array}{l} \Gamma \leadsto \Gamma' \parallel \sigma \ \text{ in } \ \Delta \\ \sigma(\gamma) \leadsto \sigma' \quad \text{ in } \Delta \end{array}}{\Gamma\ \gamma \leadsto \sigma'(\Gamma') \parallel \sigma'\sigma \ \text{ in } \ \Delta} \qquad\qquad \frac{\begin{array}{l} \Gamma \leadsto \Gamma' \parallel \sigma \ \text{ in } \ \Delta \\ \sigma(\gamma) \leadsto id \quad \text{ in } \Delta \end{array}}{\Gamma\ \gamma \leadsto \sigma(\gamma)\ \Gamma' \parallel \sigma \ \text{ in } \ \Delta}$$

For example, $\Gamma = \mathrm{fld}(B.f\ \alpha),\ \mathrm{fld}\ (\alpha.g\ \ \alpha'),\ \alpha' \leq D,\ \ \Delta_3 = \ \text{class B\{ F f\}}$

$\Gamma_3 = \ \mathrm{fld}\ (C.g\ \ \alpha'),\ \ \alpha' \leq D,$

Then, linking will give $\Gamma \leadsto \Gamma_3 \parallel \alpha \mapsto F \ \text{ in } \ \Delta_3$

Theorem 4

$\Gamma \rightsquigarrow \Gamma' \parallel \sigma$ in Δ satisfies the requirements of theorem 3.

Therefore, FJ0 compositional compilation is sound and complete.

☺

Casts are "delicate" in that the bytecode produced depends on whether the subclass relationship holds; This can be checked in global compilation, but not in compositional compilation.

$$\frac{\Delta, \Pi \;\vdash_{gl}\; e^S : c' \parallel e^B \qquad \Delta \vdash c' \leq c}{\Delta, \Pi \;\vdash_{gl}\; (c)\, e^S : c \parallel e^B}$$

$$\frac{\Delta, \Pi \;\vdash_{gl}\; e^S : c' \parallel e^B \qquad \Delta \vdash c \leq c'}{\Delta, \Pi \;\vdash_{gl}\; (c)\, e^S : c \parallel (c)\, e^B}$$

$$\frac{\Pi \;\vdash_{co}\; e^S : t \parallel \Gamma \parallel e^B}{\Pi \;\vdash_{co}\; (c)e^S : c \parallel \Gamma \parallel <<c>>e^B}$$

The function $I(\Delta, \sigma, e^B)$ replaces $<<c>>e'^B$ by $(c)e'^B$ or e'^B.

Then

$$\frac{\Gamma \leadsto \Gamma' \parallel \sigma \;\text{ in }\; \Delta}{\Gamma \parallel B \leadsto \Gamma' \parallel I(\Delta, \sigma, e^B) \;\text{ in }\; \Delta}$$

In Flexible Dynamic Linking we replace the linking phase by lazy runtime liking interleaved with resolution and verification.

We also allow the use of type variables in the signatures of methods or types of fields.

For verification, we do not load classes, instead we post constraints.

Then, type variables may be replaced very lazily, eg replace α right before the field access in [B.f α].

We have proven the type soundness of the approach.

We are developing one .NET and one JVM implementation. So far, .NET allows less flexibility.

Compositional compilation provides ”compile once, run everywhere”

Compositional compilation achieved through

- Use of type variables in annotations,
- Creation, rather than consumption of constraints,
- Linking step which satisfies constraints through the creation of appropriate substitutions.

Treatment of cases where the instruction created depends on environment requires more sophistication for linking ($l(\Delta, \sigma, \underline{e^{B}})$)

- Eg casts (in talk)
- More such cases, e.g. A.B.C.

Further issues

- overloading
- generics