Intersection types for Mixin composition

Ugo de'Liguoro

University of Turin

BETTY - London, April 2015

Based on the STSM:

- Turin: Rehof, Urzyczyn et alii, October 2013
- Dortmund: Chen, de'Liguoro, February 2014
- Turin: Düdder, March 2015

An STSM of de'Liguoro to Dortmund is scheduled in May 2015.

Papers:

- J. Rehof: *Towards Combinatory Logic Synthesis*.
  Presented at the 1st International Workshop on Behavioural Types (BEAT), 2013.
- B. Düdder, M. Martens and J. Rehof:
  Staged Composition Synthesis.
  Proceedings of ESOP'14, *LNCS* 8410, 2014.
- J. Bessai, B. Düdder, A. Dudenhefner, T-C. Chen and U. de'Liguoro:
  Typing Classes and Mixins with Intersection Types. Proceedings of ITRS'14, *EPTCS* 177, 2015.
- J. Bessai, A. Dudenhefner, B. Düdder, T-C. Chen, U. de'Liguoro, and J. Rehof:
  Mixin Composition Synthesis based on Intersection Types.
  to appear in the Proceedings of TLCA'15.

Motivation

- Composition synthesis = relativized inhabitation
- the method is based on combinatory logic and intersection types
- by understanding components as classes and mixins, we propose an application of the inhabitation method to libraries for object-oriented programming languages

Synthesis via inhabitation

Semantic specification:

$$succ : (\mathsf{Int} \to \mathsf{Int}) \cap (\mathsf{Odd} \to \mathsf{Even}) \cap (\mathsf{Even} \to \mathsf{Odd})$$

Consider a typing

$$\Gamma \vdash e : \tau$$

where e is a combination of the variables x_i such that $x_i : \sigma_i \in \Gamma$.

- variables x_i are names of *components*, with specifications σ_i
- Γ is the interface of a repository to build the *program* e satisfying the specification τ

Solving the inhabitation problem

$$\Gamma \vdash \ ? \ : \tau$$

is the same as synthesizing the program e.

The calculus Λ_R

Terms:

$$\begin{array}{rcll} M, N & ::= & x \mid \lambda x.M \mid MN \mid R \mid M.a \mid M \oplus R & \text{term} \\ R & ::= & \langle \ell_i = M_i \mid i \in I \rangle & \text{record} \end{array}$$

The empty record $\langle \ell_i = M_i \mid i \in \emptyset \rangle \equiv \langle\rangle$.

Reduction:

$$\begin{array}{lll} (\beta) & (\lambda x.M)N \longrightarrow M\{N/x\} & \\ (R_1) & \langle \ell_i = M_i \mid i \in I \rangle.\ell_k \longrightarrow M_k & \text{if } k \in I \\ (R_2) & \langle \ell_i = M_i \mid i \in I \rangle \oplus \langle \ell_j = N_j \mid j \in J \rangle & \\ & \qquad \longrightarrow \langle \ell_i = M_i,\ \ell_j = N_j \mid i \in I \setminus J, j \in J \rangle & \end{array}$$

Intersection types

Types:

$$\begin{array}{rcll} \sigma, \tau & ::= & \alpha \mid a \mid \omega \mid \sigma \to \tau \mid \sigma \cap \tau \mid \rho & \text{type} \\ \rho & ::= & \langle\rangle \mid \langle \ell : \sigma \rangle \mid \rho_1 \cap \rho_2 & \text{record type} \end{array}$$

Semantics:

$$\begin{array}{rcl} [\![\langle\rangle]\!] & = & \{M \mid \exists R.\ M \longrightarrow^* R \ \ \& \ \ R \text{ is a record}\} \\ [\![\langle \ell : \sigma \rangle]\!] & = & \{M \in \Lambda_R \mid M \in [\![\langle\rangle]\!] \ \ \& \ \ M.\ell \in [\![\sigma]\!]\} \end{array}$$

Interpreting $\leq$ by $\subseteq$ we have:

$$\begin{array}{rcl} \langle \ell : \sigma \rangle & \leq & \langle\rangle \\ \langle \ell : \sigma \rangle \cap \langle \ell : \tau \rangle & = & \langle \ell : \sigma \cap \tau \rangle \\ \sigma \leq \tau & \Rightarrow & \langle \ell : \sigma \rangle \leq \langle \ell : \tau \rangle \end{array}$$

Typing rules

Rules:

$$\frac{}{\Gamma \vdash \langle\rangle : \langle\rangle} \qquad \frac{\Gamma \vdash M : \sigma \qquad \ell = M \in R}{\Gamma \vdash R : \langle \ell : \sigma \rangle} \qquad \frac{\Gamma \vdash M : \langle \ell : \sigma \rangle}{\Gamma \vdash M.a : \sigma}$$

If we abbreviate

$$\langle \ell_i : \sigma_i \mid i \in I \rangle \equiv \bigcap_{i \in I} \langle \ell_i : \sigma_i \rangle$$

then we get by $\cap$-introduction:

$$\frac{\Gamma \vdash M_j : \sigma_j \qquad \forall i \in J \subseteq I}{\Gamma \vdash \langle \ell_i = M_i \mid i \in I \rangle : \langle \ell_j : \sigma_j \mid j \in J \rangle}$$

Note that possibly $J \subset I$.

Typing rules for $\oplus$ (merge)

Define

$$\ell(\langle \ell_i = M_i \mid i \in I \rangle) = \{\ell_i \mid i \in I\}$$

$$\frac{\Gamma \vdash M : \langle\rangle \quad \Gamma \vdash R : \langle \ell : \sigma \rangle}{\Gamma \vdash M \oplus R : \langle a : \sigma \rangle} (\oplus_1) \qquad \frac{\Gamma \vdash M : \langle a : \sigma \rangle \qquad \ell \notin \ell(R)}{\Gamma \vdash M \oplus R : \langle \ell : \sigma \rangle} (\oplus_2)$$

so that the following rule is admissible

$$\frac{\Gamma \vdash M : \langle \ell_i : \sigma_i \mid i \in I \rangle \qquad \Gamma \vdash R : \langle j : \tau_j \mid j \in J \rangle \qquad I \cap \ell(R) = \emptyset}{\Gamma \vdash M \oplus R : \langle \ell_i : \sigma_i, \ell_j : \tau_j \mid i \in I \setminus J, j \in J \rangle}$$

A combinatorial view of classes and mixins

$$
\begin{array}{rcll}
\mathcal{C} \ni C & ::= & \mathbf{Y}(\lambda \mathsf{self}.\langle m_i = b_i \mid i \in I\rangle) & \bigcup_i fv(b_i) \subseteq \{\mathsf{self}\} \\
\Delta & ::= & \langle n_j = b_j \mid j \in J\rangle & \bigcup_i fv(b_i) \subseteq \{\mathsf{self}, \mathsf{super}\} \\
\mathcal{M} \ni M & ::= & \lambda \mathsf{super}.\mathbf{Y}(\lambda \mathsf{self}.\ \mathsf{super} \oplus \Delta) &
\end{array}
$$

$$
\begin{array}{rll}
 & M\ C & \\
\longrightarrow_\beta & \mathbf{Y}(\lambda \mathsf{self}'.\ C \oplus \Delta') & \Delta' \equiv \Delta\{C/\mathsf{super}\} \\
\longrightarrow_\beta & \mathbf{Y}(\lambda \mathsf{self}'.\ \langle m_i = b_i' \mid i \in I\rangle \oplus \Delta') & b_i' \equiv b_i\{C/\mathsf{self}\} \\
\longrightarrow_\oplus & \mathbf{Y}(\lambda \mathsf{self}'.\ \langle m_i = b_i', n_j = b_j' \mid i \in I \setminus J, j \in J\rangle) & \bigcup_{k \in I \cup J} fv(b_k') \subseteq \{\mathsf{self}'\}
\end{array}
$$

hence a *mixin* is a function of classes.

Remark

In the definition of mixins:

$$\lambda \text{super}.\mathbf{Y}(\lambda \text{self}.\ \text{super} \oplus \Delta)$$

we have *static binding* of self, like with C++ non-virtual methods.

A definition closer to Cook and other's model is:

$$\begin{array}{rcll} \mathcal{C}_s \ni C_s & ::= & \lambda \text{self}.\langle m_i = b_i \mid i \in I \rangle & \bigcup_i fv(b_i) \subseteq \{\text{self}\} \\ \Delta & ::= & \langle n_j = b_j \mid j \in J \rangle & \bigcup_i fv(b_i) \subseteq \{\text{self}, \text{super}\} \\ \mathcal{M}_s \ni M_s & ::= & \lambda \text{super} \lambda \text{self}.\ (\text{super self}) \oplus \Delta & \end{array}$$

- advantages: this models the *late binding* of self in the super-class, like with Java methods;
- disadvantages: types are much more complex.

Typing classes and mixins with intersection types

To type the class $C \equiv \mathbf{Y}(\lambda \mathsf{self}.R) \in \mathcal{C}$ consider:

$$\mathbf{Y} : (\omega \to \rho_1) \cap (\rho_1 \to \rho_2) \cap \cdots \cap (\rho_{n-1} \to \rho_n) \to \rho_n$$

We have to find $\rho_0 = \omega, \rho_1, \ldots, \rho_n = \rho$ such that

$$\frac{\mathsf{self} : \rho_i \vdash R : \rho_{i+1} \qquad \forall i < n}{\vdash \lambda \mathsf{self}.R : (\omega \to \rho_1) \cap (\rho_1 \to \rho_2) \cap \cdots \cap (\rho_{n-1} \to \rho_n)}$$

A way to generate the ρ_i is to define $R_0 \equiv \Omega$, and $R_{n+1} \equiv R\{R_n/\mathsf{self}\}$:

Fact

- *self* : $\rho_i \vdash R : \rho_{i+1}$, *and hence* $\vdash R_{i+1} : \rho_{i+1}$ *for all* $i < n$
- $C : \rho$ *if and only if* $\rho_n \leq \rho$ *for some* n.

Typing classes and mixins with intersection types

Let $M \equiv \lambda\text{super}.\mathbf{Y}(\lambda\text{self}.(\text{super} \oplus \Delta))$

$$\dfrac{\dfrac{\text{super} : \rho, \text{self} : \rho_1 \vdash \Delta : \rho_1' \qquad \ell(\rho) \cap \ell(\Delta) = \emptyset}{\text{super} : \rho, \text{self} : \rho_1 \vdash \text{super} \oplus \Delta : \rho \cap \rho_1'}}{\text{super} : \rho \vdash \lambda\text{self}.\,(\text{super} \oplus \Delta) : \rho_1 \to \rho \cap \rho_1'}$$

Setting $\rho_2 = \rho \cap \rho_1'$ we obtain the typing

$$\text{super} : \rho \vdash \lambda\text{self}.\,(\text{super} \oplus \Delta) : \rho_2 \to \rho \cap \rho_2'$$

So

$$\dfrac{\dfrac{\text{super} : \rho \vdash \lambda\text{self}.\,(\text{super} \oplus \Delta) : \rho_i \to \rho \cap \rho_i' \qquad i = 1, \ldots, n}{\text{super} : \rho \vdash \mathbf{Y}(\lambda\text{self}.\,(\text{super} \oplus \Delta)) : \rho \cap \rho_n}}{\vdash M : \rho \to \rho \cap \rho_n}$$

The synthesis by inhabitation problem for mixins

Given a collection of classes and mixins with the respective types:

$$M_1 : \sigma_1, \ldots, M_h : \sigma_h, C_1 : \tau_1, \ldots, C_k : \tau_k$$

solve the inhabitation problem

$$x_1 : \sigma_1, \ldots, x_h : \sigma_h, y_1 : \tau_1, \ldots, y_k : \tau_k \vdash \ ? \ : \tau$$

and then replace $x_i \mapsto M_i$ and $y_j \mapsto C_j$.

Remark:

- in practice the goal is $\tau = \rho$ (a record type),
- the synthesized class C has the shape

$$(M_{i_1} \circ \cdots \circ M_{i_n})\, C_j$$

The tool

```
let input = """
base: Int

combinatorsC

    //class combinators
    Point : Int -> Record([get(Int), set(Int -> Int), shift(Int -> Int)])

    //mixin combinators
    Movable :
        w_set_move(Record(alpha.k))
            -> (Int -> Record([alpha.k, set(Int -> Int), shift(Int -> Int)]
                -> (Int -> Record([alpha.k, set(Int), move(Int)]))

    //logical combinators
    W_get : w_get(Record([set(alpha1.k), shift(alpha2.k), move(alpha3.k)]))
    W_set : w_set(Record([get(alpha1.k), shift(alpha2.k), move(alpha3.k)]))
    W_shift : w_shift(Record([get(alpha1.k), set(alpha2.k), move(alpha3.k)]
    W_move : w_move(Record([get(alpha1.k), set(alpha2.k), shift(alpha3.k)])
    W_set_move : w_set(alpha.k) -> w_move(alpha.k) -> w_set_move(alpha.k)
```

Future directions

- to relate intersection types to F-bounded types, to extract automatically the specifications of classes and mixins
- to consider mixins with arity greater than one, and in general other forms of modules
- to develop synthesis of Java code via staged composition synthesis