There is a definition of an operational semantics for a core subset of Muli (see Dageförde, J. C., & Kuchen, H. (2017). An operational semantics for constraint-logic imperative programming. In *Declarative Programming and Knowledge Management* (pp. 64-80). Springer, Cham., https://doi.org/10.1007/978-3-030-00801-7_5). Here, we first display an excerpt from the reduction rules that are relevant to a rule that we modify. Subsequently, the modified rule is presented. Throughout the definitions, modifications to their respective originals that were necessary in order to add support for free objects are indicated in red.

Symbols In this reduction semantics, computations depend on an environment, a state, and a constraint store.

- $\mathit{Env} = (\mathit{Var} \cup \mathcal{M}) \to (\mathcal{A} \cup (\mathit{Var}^* \times \mathit{Stat}))$: Set of all environments, mapping variables $\in \mathit{Var}$ to addresses $\in \mathcal{A}$ and methods $\mathcal{M}$ to a tuple $((x_0, x_1, \ldots, x_k), s)$, signifying parameters and a code body s. Note that we add x_0 to the original definition. x_0 shall hold the object that a method was invoked on, unless the method is static.
- $\rho_0 \in \mathit{Env}$ is a special initial environment that maps functions to their respective parameters and code (under the simplifying assumption that classes and their methods are in global scope).
- $\Sigma = \mathcal{A} \to (\{\bot\} \cup \mathit{Tree}(\mathcal{A}, \mathbb{Z}))$: Set of all possible memory states.
- A special address α_0 with $\sigma(\alpha_0) = \bot$ is reserved for holding return values of method invocations.
- $\mathit{CS} = \{\texttt{true}\} \cup \mathit{Tree}(\mathcal{A}, \mathbb{Z})$: Set of all possible constraint store states.
- $\rho \in \mathit{Env}$, $\sigma \in \Sigma$, $\gamma \in \mathit{CS}$. Discriminating indices are added if necessary.
- $a[x/d]$ is used for modifications to a state or environment a, meaning

$$a[x/d](b) = \begin{cases} d & \text{, if } b = x \\ a(b) & \text{, otherwise.} \end{cases}$$

- The semantics of expressions is described with the infix relation $\to \; \subset (\mathit{Expr} \times \mathit{Env} \times \Sigma \times \mathit{CS}) \times ((\mathbb{B} \cup \mathit{Tree}(\mathcal{A}, \mathbb{Z})) \times \Sigma \times \mathit{CS})$.
- The semantics of statements is described by the infix relation $\leadsto \; \subset (\mathit{Stat} \times \mathit{Env} \times \Sigma \times \mathit{CS}) \times (\mathit{Env} \times \Sigma \times \mathit{CS})$.

Syntax The grammar is only modified slightly, incorporating method invocations on objects and reference type variables. Otherwise, taken from the original semantics.

$$\begin{aligned}
e ::= {} & c \mid x \mid e_1 \oplus e_2 \mid x.m(e_1, \ldots, e_k) \\
& \text{where } c \in \mathbb{Z},\ x \in Var,\ e_1, \ldots, e_k \in AExpr,\ \oplus \in AOp,\ k \in \mathbb{N}, \\
& x.m \text{ can be resolved to an implementation } i \in \mathcal{M},
\end{aligned}$$

$$\begin{aligned}
b ::= {} & e_1 \odot e_2 \mid b_1 \otimes b_2 \mid \texttt{true} \mid \texttt{false} \\
& \text{where } e_1, e_2 \in AExpr,\ b_1, b_2 \in BExpr,\ \odot \in ROp, \otimes \in BOp,
\end{aligned}$$

$$\begin{aligned}
s ::= {} & ; \mid \texttt{int } x; \mid \texttt{int } x \texttt{ free}; \mid \texttt{T } x; \mid \texttt{T } x \texttt{ free}; \mid x = e; \mid e; \mid \{s\} \mid s_1\ s_2 \mid \\
& \texttt{if } (b)\ s_1 \texttt{ else } s_2 \mid \texttt{while } (b)\ s \mid \texttt{return } e; \mid \texttt{fail}; \\
& \text{where } x \in Var,\ e \in AExpr,\ b \in BExpr,\ s, s_1, s_2 \in Stat,\ \texttt{T} \text{ is a class or interface type.}
\end{aligned}$$

Reduction Rules The following reproduces reduction rules from the original semantics as preliminaries, before presenting a rule modification for non-deterministic invocation.

Variable resolution:

$$\langle x, \rho, \sigma, \gamma \rangle \to (\sigma(\rho(x)), \sigma, \gamma) \tag{Var}$$

Arithmetic expressions, resulting either in a constant value if nested expressions are constant, or in a symbolic expression otherwise:

$$\frac{\langle e_1, \rho, \sigma, \gamma \rangle \to (v_1, \sigma_1, \gamma_1),\ \langle e_2, \rho, \sigma_1, \gamma_1 \rangle \to (v_2, \sigma_2, \gamma_2),\quad v_1, v_2, v = v_1 \oplus v_2 \in \mathbb{Z}}{\langle e_1 \oplus e_2, \rho, \sigma, \gamma \rangle \to (v, \sigma_2, \gamma_2)} \tag{AOp1}$$

$$\frac{\langle e_1, \rho, \sigma, \gamma \rangle \to (v_1, \sigma_1, \gamma_1),\ \ \langle e_2, \rho, \sigma_1, \gamma_1 \rangle \to (v_2, \sigma_2, \gamma_2),\ \ \{v_1, v_2\} \nsubseteq \mathbb{Z}}{\langle e_1 \oplus e_2, \rho, \sigma, \gamma \rangle \to (\oplus(v_1, v_2), \sigma_2, \gamma_2)} \tag{AOp2}$$

Non-deterministic Invocation Under the assumption of global functions and disregarding object-oriented features, the original definition of the operational semantics of invocation is a deterministic operation:

$$\frac{\begin{array}{r}\langle e_1, \rho, \sigma, \gamma \rangle \to (v_1, \sigma_1, \gamma_1),\ \ \langle e_2, \rho, \sigma_1, \gamma_1 \rangle \to (v_2, \sigma_2, \gamma_2),\ \ \ldots, \\ \langle e_k, \rho, \sigma_{k-1}, \gamma_{k-1} \rangle \to (v_k, \sigma_k, \gamma_k),\ \rho(m) = (\bar{x}_k,\ s), \\ \langle s, \rho_0[\bar{x}_k/\bar{\alpha}_k], \sigma_k[\bar{\alpha}_k/\bar{v}_k], \gamma_k \rangle \rightsquigarrow (\rho_{k+1}, \sigma_{k+1}, \gamma_{k+1}),\ \ \sigma_{k+1}(\alpha_0) = r\end{array}}{\langle m(e_1, \ldots, e_k), \rho, \sigma, \gamma \rangle\ \to (r, \sigma_{k+1}[\alpha_0/\ \bot], \gamma_{k+1})}$$

Adding support for object-orientation and the availability of multiple implementations for an object, we modify and replace the rule as presented subsequently (changes highlighted in red). The new rule uses the set $implementations(o, m)$ that calculates possible implementing types.

$$\frac{\begin{array}{c}\langle o, \rho, \sigma, \gamma\rangle \rightarrow (v_0, \sigma, \gamma),\ \ \langle e_1, \rho, \sigma, \gamma\rangle \rightarrow (v_1, \sigma_1, \gamma_1),\ \ \langle e_2, \rho, \sigma_1, \gamma_1\rangle \rightarrow (v_2, \sigma_2, \gamma_2),\\ \ldots,\ \ \langle e_k, \rho, \sigma_{k-1}, \gamma_{k-1}\rangle \rightarrow (v_k, \sigma_k, \gamma_k),\ i \in implementations(v_0, m),\\ \rho(i) = (\bar{x}_k,\ s),\ \ \langle s, \rho_0[{}^{\bar{x}_k}/{}_{\bar{\alpha}_k}], \sigma_k[{}^{\bar{\alpha}_k}/{}_{\bar{v}_k}], \gamma_k \wedge types(o) = T\rangle \rightsquigarrow (\rho_{k+1}, \sigma_{k+1}, \gamma_{k+1}),\ \ \sigma_{k+1}(\alpha_0) = r\end{array}}{\langle o.m(e_1, \ldots, e_k), \rho, \sigma, \gamma\rangle \ \rightarrow (r, \sigma_{k+1}[\alpha_0/\ \bot], \gamma_{k+1})} \tag{Invoke-ND}$$

For non-free objects $target$ whose class has a definition for m, $implementations(target, m)$ is a singleton. Therefore, invocation is deterministic. For free objects, $implementations$ $(target, m)$ may have more elements. In that case, the evaluation of this rule becomes non-deterministic. Moreover, note that a constraint $types(o) = T$ is added after selecting a specific implementation. T depends on the selected implementation alternative.

The new rule depends on the rule in Equation (Var) for resolving the object variable `o` based on the state of environment and memory, and on the rules in Equations (AOp1) and (AOp2) for substituting parameter expressions. The definition assumes that the environment ρ contains every method definition, comprising a parameter definition $\bar{x}_k$ and a body s.