Foundations of the UML

Lecture 17: Semantics of OCL Expressions

Joost-Pieter Katoen

Lehrstuhl für Informatik 2
Software Modeling and Verification Group

http://moves.rwth-aachen.de/i2/370

24. Januar 2010

RWTH AACHEN UNIVERSITY

OCL expressions

Definition (OCL expressions)

The syntax of OCL expressions is defined by the grammar:

$$\xi \ ::= \ \texttt{self} \mid z \mid \texttt{result} \mid \xi@\texttt{pre} \mid \xi.a \mid \omega(\xi,\ldots,\xi) \mid$$
$$\xi.\omega(\xi,\ldots,\xi) \mid \xi \rightarrow \omega(\xi,\ldots,\xi) \mid \xi \rightarrow \texttt{iterate}(x_1; x_2 := \xi \mid \xi)$$

where:

- `self` refers to the context object of class C
- z represents either an attribute of the context object, or a formal parameter of a context method, or a logical variable
- `result` refers to the value returned by the context method (which is undefined if this method has not yet returned a value)
- @`pre` refers to the value of its operand on invoking this method
- expressions `result` and @`pre` can be used in `postconditions` only

OCL operations

- $\xi.a$ is an attribute/parameter navigation
  - ξ is an object reference to an object with attribute a, or
  - ξ is a reference to a method occurrence with a formal parameter a
  - $\xi.a$ denotes the value of this attribute/parameter
  - e.g.: (h.rooms).guests, h.rooms, $m.g$ for method invocation$m$
- $\omega(\xi_1, \ldots, \xi_n)$ denotes the application of the n-ary operator ω to the arguments ξ_1 through ξ_n

  some examples are: isEqual(g_1, g_2) and ifThenElse(b, ξ_1, ξ_2), etc.
- $\xi.\omega(\xi_1, \ldots, \xi_n)$ represents an operator ω on basic types applied on ξ and arguments ξ_1 through ξ_n
- $\xi \rightarrow \omega(\xi_1, \ldots, \xi_n)$ represents an operator ω on collection types applied on collection ξ and arguments ξ_1 through ξ_n

Preliminaries

The OCL semantics is defined using an operational model (intuitively: a transition system) of an object-based system.

We first need to fix a set of variable, method and class names

Definition (Data types for logical variables)

- VName is a countable set of variable names
- MName is a countable set of method names (ranged over by M)
- CName is a countable set of class names (ranged over by C)

RWTHAACHEN UNIVERSITY

Semantic data types

Definition (Semantic types)

The language TYPE of data types is defined by the grammar:

$$\tau \ ::= \ \text{void} \mid \text{nat} \mid \text{bool} \mid \tau \text{ list} \mid C \text{ ref} \mid C.M \text{ ref}$$

where $C \in$ CNAME and $M \in$ MNAME.

- void represents the unit type with trivial value (),
- τ list denotes the type of lists of τ with elements [] (the empty list) and $h :: w$ (list with head h of type τ and tail w of type τ list); notation $1 :: 2 :: []$ as $[1, 2]$ and $(1 :: []) :: (2 :: []) :: []$ as $[[1], [2]]$
- C ref is the type of objects of class C
- $C.M$ ref is the type of method occurrences of method M of class C

Formal definitions

Definition (Variable, method and class definitions)

We define the following sets of partial functions:

- $\textsc{Vdecl} = \{ \text{VN}\textsc{ame} \to \textsc{Type} \}$ set of variable declarations

  Each variable declaration maps variable names to types
- $\textsc{Mdecl} = \{ \text{MN}\textsc{ame} \to \textsc{Vdecl} \times \textsc{Type} \}$ set of methods

  Each method declaration maps a method name onto its formal parameters and return type
- $\textsc{Cdecl} = \{ \text{CN}\textsc{ame} \to \textsc{Vdecl} \times \textsc{Mdecl} \}$ set of class declarations

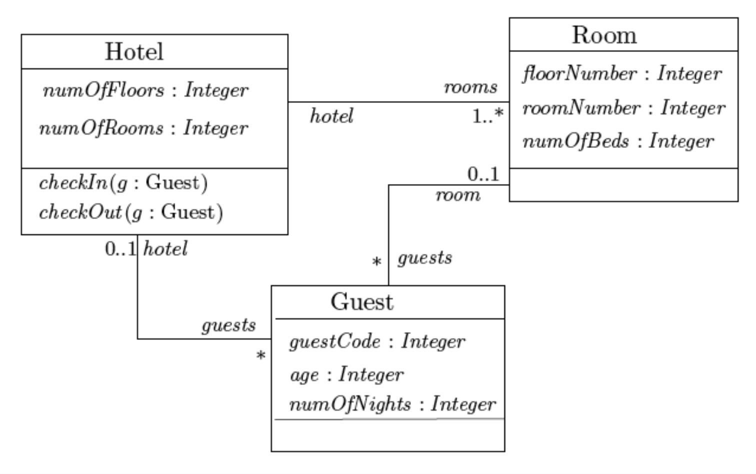
  Each class declaration maps a class name to a set of attributes and methods

Notations

Notations

- Let $D \in \textsc{CDecl}$. For $C \in dom(D)$, let $C.attrs$ denote its attributes and $C.meths$ its methods
- For method M of class C, $M.fpars$ are its formal parameters, and $M.retty$ is its return type
- Thus: $C.meths(M) = (M.fpars, M.retty)$

OCL example



Hotel
numOfFloors : Integer
numOfRooms : Integer
checkIn(g : Guest)
checkOut(g : Guest)
Room
floorNumber : Integer
roomNumber : Integer
numOfBeds : Integer
Guest
guestCode : Integer
age : Integer
numOfNights : Integer
rooms
hotel
1..*
0..1
room
0..1
hotel
*
guests
guests
*

Example

$$
\mathrm{Hotel}.\mathit{attrs}(v) = \begin{cases} \textsf{nat} & \text{if } v \in \{\mathit{numOfFloors}, \mathit{numOfRooms}\} \\ \mathrm{Room}\ \textsf{list} & \text{if } v = \mathit{rooms} \\ \mathrm{Guest}\ \textsf{list} & \text{if } v = \mathit{guests} \\ \bot & \text{otherwise} \end{cases}
$$

$$
\mathrm{Room}.\mathit{attrs}(v) = \begin{cases} \textsf{nat} & \text{if } v \in \{\mathit{floorNumber}, \mathit{roomNumber}, \mathit{numOfBeds}\} \\ \mathrm{Hotel} & \text{if } v = \mathit{hotel} \\ \mathrm{Guest}\ \textsf{list} & \text{if } v = \mathit{guests} \\ \bot & \text{otherwise} \end{cases}
$$

$$
\mathrm{Guest}.\mathit{attrs}(v) = \begin{cases} \textsf{nat} & \text{if } v \in \{\mathit{guestCode}, \mathit{age}, \mathit{numOfNights}\} \\ \mathrm{Hotel} & \text{if } v = \mathit{hotel} \\ \mathrm{Room} & \text{if } v = \mathit{room} \\ \bot & \text{otherwise} \end{cases}
$$

$$
\mathit{checkIn}.\mathit{fpars}(v) = \begin{cases} \mathrm{Guest} & \text{if } v = g \\ \bot & \text{otherwise} \end{cases}
$$

$$
\mathit{checkOut}.\mathit{fpars}(v) = \mathit{checkIn}.\mathit{fpars}(v)
$$

$$
\mathrm{Hotel}.\mathit{meths}(M) = \begin{cases} (\mathit{checkIn}.\mathit{fpars}, \textsf{void}) & \text{if } M = \mathit{checkIn} \\ (\mathit{checkOut}.\mathit{fpars}, \textsf{void}) & \text{if } M = \mathit{checkOut} \\ \bot & \text{otherwise} \end{cases}
$$

$$
\mathrm{Room}.\mathit{meths}(M) = \bot
$$

$$
\mathrm{Guest}.\mathit{meths}(M) = \bot
$$

RWTH AACHEN UNIVERSITY

Objects

Objects

Objects will be numbered instances of their class $C \in \textsc{CName}$.

Let

- The domain of object ids of class C is defined by $\textsc{Oid}^C = \{C\} \times \mathbb{N}$.
- Let $\textsc{Oid} = \bigcup_C \textsc{Oid}^C$ denote the set of object ids.

Thus:

Elements of $\textsc{Oid}$ are pairs (C, n), denoting the n-th instance of class C.

Events

Method invocations

Method occurrences, also called events, will be numbered instances of method $M \in \textsc{MName}$ plus an indication of the object executing M.

Let:

- $\textsc{Evt}^{C,M} = \textsc{Oid}^C \times \{M\} \times \mathbb{N}$ be the domain of method invocations (= events) of M of class C
- Let $\textsc{Evt} = \bigcup_C \bigcup_M \textsc{Evt}^{C,M}$ denote the set of events.

Thus:

Elements of $\textsc{Evt}$ are tuples $((C, n), M, k)$ denoting the k-th method invocation of M which currently is executed by object (C, n).

Example

Example 3.2.1. Consider the Hotel class diagram of Figure 2.3. The following are instances of the class Hotel:

$$(\text{Hotel}, 1) \quad (\text{Hotel}, 2) \quad (\text{Hotel}, 31) \quad (\text{Hotel}, 127) \quad \ldots$$

The following are events related to the method *checkIn*:

$$\begin{array}{ll} ((\text{Hotel}, 1), \mathit{checkIn}, 1) & ((\text{Hotel}, 1), \mathit{checkIn}, 2) \\ ((\text{Hotel}, 31), \mathit{checkIn}, 1) & ((\text{Hotel}, 127), \mathit{checkIn}, 3) \quad \ldots \end{array}$$

Note that the first two events represent different executions of method *checkIn* performed by the same object. □

Values and operations

The combined universe of values will be denoted by $\textsc{Val}$; the set of values of a given type $\tau \in \textsc{Type}$ is denoted by $\textsc{Val}^\tau$. We define:

$$
\begin{aligned}
\textsc{Val}^{\mathsf{void}} &= \{()\} \\
\textsc{Val}^{\mathsf{nat}} &= \mathbb{N} \\
\textsc{Val}^{\mathsf{bool}} &= \{\mathsf{ff}, \mathsf{tt}\} \\
\textsc{Val}^{\tau\ \mathsf{list}} &= \{[]\} \cup \{h :: w \mid h \in \textsc{Val}^\tau, w \in \textsc{Val}^{\tau\ \mathsf{list}}\} \\
\textsc{Val}^{C\ \mathsf{ref}} &= \{\mathsf{null}\} \cup \textsc{Oid}^C \\
\textsc{Val}^{C.M\ \mathsf{ref}} &= \textsc{Evt}^{C,M} \quad .
\end{aligned}
$$

- $+ : \textsc{Val}^{\mathsf{nat}} \times \textsc{Val}^{\mathsf{nat}} \rightarrow \textsc{Val}^{\mathsf{nat}}$ is the standard sum on natural numbers.
- $\mathit{sort} : \textsc{Val}^{\tau\ \mathsf{list}} \rightarrow \textsc{Val}^{\tau\ \mathsf{list}}$ orders a given list of values of type τ.
- $\mathit{flat} : \textsc{Val}^{\tau\ \mathsf{list\ list}} \rightarrow \textsc{Val}^{\tau\ \mathsf{list}}$ flattens nested lists.

Finally, there is a special element $\bot \notin \textsc{Val}$ that is used to model the "undefined" value: we write $\textsc{Val}_\bot = \textsc{Val} \cup \{\bot\}$. All operations are extended to $\bot$ by requiring them to be *strict* (meaning that if any operand equals $\bot$, the entire expression equals $\bot$). For instance, for lists we have $\bot :: w = \bot$ and $h :: \bot = \bot$.

Configurations

Definition (Configuration)

A configuration is a tuple (O, E, σ, γ) with:

- $O \subseteq \textsc{Oid}$, the currently alive objects
- $E \subseteq \textsc{Evt}$, the currently running method invocations
- $\sigma : O \to \textsc{Vname} \to \textsc{Val}$, the local state of objects in O
- $\gamma : E \to (\textsc{Vname} \to \textsc{Val}) \times \textsc{Val}_{\perp}$, the state of method invocations

State information

- $\sigma(o)$ is the local state of object o such that $\sigma(o) = \ell$ with $o \in \textsc{Oid}^C$ implies $dom(\ell) = dom(C.attrs)$ and $\ell(a) \in \textsc{Val}^{C.attrs(a)}$ for each $a \in dom(\ell)$.
- σ is extended point-wise to lists of objects, i.e.,

$$\sigma([])(a) = [] \quad \text{and} \quad \sigma(h :: w)(a) = \sigma(h)(a) :: \sigma(w)(a).$$

Configurations

Method invocations

Recall: $\gamma : E \to (\textsc{Vname} \to \textsc{Val}) \times \textsc{Val}_\perp$

If $\gamma(e) = (\ell, v)$ for $e \in \textsc{Evt}^{C,M}$ then:

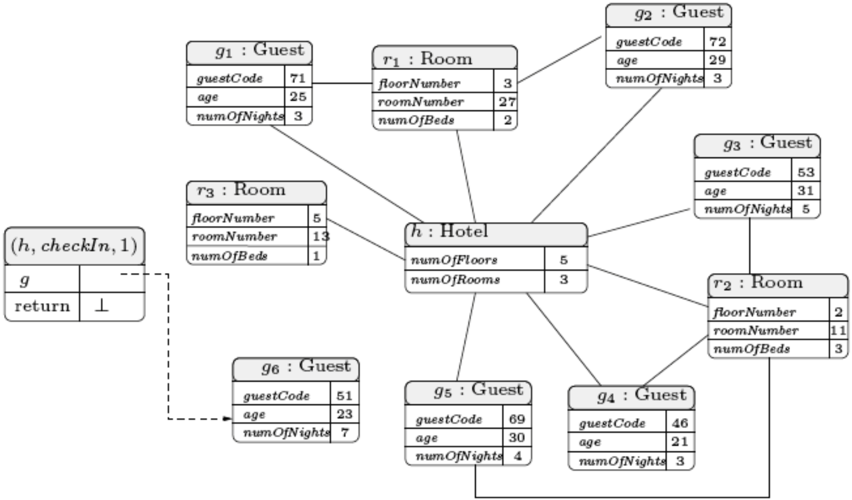
- $dom(\ell) = dom(M.fpars)$,
- $\ell(p) \in \textsc{Val}^{M.fpars(p)}$ for $p \in dom(\ell)$, the value of M's parameters
- $v \in \textsc{Val}_\perp^{M.retty}$, the returned value

Method termination

A method invocation has terminated in the current configuration if it becomes inactive in the next state. On termination, the method has a well-defined value (i.e., different from $\perp$).
If the system transits from configuration (O, E, σ, γ) to $(O', E', \sigma', \gamma')$ then:

$$e \in E - E' \quad \text{implies} \quad \exists v \in \mathit{Val}.\, \gamma(e) = (\ell, v)$$

Configurations: example



Configurations: example

Example 3.3.2. Figure 3.1 depicts a possible configuration of the Hotel model. In particular we have:

$$O = \{h, r_1, r_2, r_3, g_1, g_2, g_3, g_4, g_5, g_6\}$$
$$E = \{(h, \mathit{checkIn}, 1)\}$$

where we have adopted the following abbreviation: $h = (\text{Hotel}, 1)$, $g_i = (\text{Guest}, i)$ and $r_i = (\text{Room}, i)$ for $i \in \mathbb{N}$. The objects show the values of the components ς and γ. For example, for object g_6 we have:

$$\varsigma(g_6)(\mathit{guestCode}) = 51$$
$$\varsigma(g_6)(\mathit{age}) = 23$$
$$\varsigma(g_6)(\mathit{numOfNights}) = 7$$

For the other objects, ς can be obtained in a similar way. The γ component for the only active method is $\gamma(h, \mathit{checkIn}, 1) = (g \mapsto g_6,\ \bot)$. □

Static expressions

We will translate OCL expressions into static expressions and provide these with a formal semantics

Definition (Static expressions)

The syntax of static expressions is defined by the grammar:

$$\begin{aligned} \xi \quad ::= \quad & x \mid \xi.a \mid \xi.\texttt{owner} \mid \xi.\texttt{return} \mid \xi\ \texttt{new} \mid \xi\ \texttt{alive} \mid \\ & \omega(\xi, \ldots, \xi) \mid \texttt{with}\ \ x_1 \in \xi\ \texttt{from}\ \ x_2 := \xi\ \texttt{do}\ \ x_2 := \xi \end{aligned}$$

where x, x_1 and x_2 are logical variables.

Static expressions

- $\xi.a$ stands for parameter/attribute navigation
- $\xi.\texttt{owner}$ denotes the object executing method instance ξ
- $\xi.\texttt{return}$ denotes the return value of method instance ξ
- ξ $\texttt{new}$ is a predicate denoting that the object or method referred to by ξ is "fresh" in the current state
  1. an object is new just after its creation
  2. a method instance is new when it is just invoked

RWTH AACHEN UNIVERSITY

Static expressions

- ξ `alive` is a predicate denoting that the object or method referred to by ξ is currently alive
  1. an object becomes alive when it is created and remains alive until it is deallocated
  2. a method instance becomes alive on invoking that method and remains alive until its return value has been returned

- `with` $x_1 \in \xi$ `from` $x_2 := \xi$ `do` $x_2 := \xi$ corresponds to the OCL expression `iterate`

Semantics of static expressions

Definition (Semantics of static expressions)

The semantics of static expression ξ is a value in $\textsc{Val}_\perp$ assigned by function $[\![\xi]\!]_{q,N,\theta}$ where

- $q = (O_q, E_q, \sigma_q, \gamma_q)$ is a configuration, with O_q is the set of alive objects, E_q the current events, σ_q is the state of all alive objects, and γ_q the state of all events
- $N \subseteq O_q \cup E_q$ denotes the set of new objects and events in the configuration q
- $\theta : \textsc{Lvar} \rightarrow \textsc{Val}$ is a partial function assigning values to logical variables, i.e., for $x \in dom(\textsc{Lvar})$, $\theta(x)$ is the value of x

Semantics of static expressions

Variables

$[\![x]\!]_{q,N,\theta} \;=\; \theta(x)$ is the logical valuation of x

Owner expressions

$$[\![\xi.\texttt{owner}]\!]_{q,N,\theta} \;=\; o \text{ where } \underbrace{[\![\xi]\!]_{q,N,\theta} = (o, M, j)}_{\text{object } o \text{ invoked } j\text{-th instance of } M}$$

Return expressions

$$[\![\xi.\texttt{return}]\!]_{q,N,\theta} \;=\; v \text{ where } \underbrace{\gamma_q\left([\![\xi]\!]_{q,N,\theta}\right) = (\ell, v)}_{\text{value of } \xi \text{ in configuration } q}$$

Semantics of static expressions

New expressions

$[\![\xi\ \texttt{new}]\!]_{q,N,\theta} \ = \ ([\![\xi]\!]_{q,N,\theta} \in N)$, i.e., ξ `new` yields true if the object (or method) referred to by ξ is in N.

Alive expressions

$[\![\xi\ \texttt{alive}]\!]_{q,N,\theta} \ = \ ([\![\xi]\!]_{q,N,\theta} \in O_q \cup E_q)$

Operations

$[\![\omega(\xi_1, \ldots, \xi_n)]\!]_{q,N,\theta} \ = \ [\![\omega]\!]\,([\![\xi_1]\!]_{q,N,\theta}, \ldots, [\![\xi_n]\!]_{q,N,\theta})$ where $[\![\omega]\!]$ is the semantic counterpart (on the domain $\textsc{Val}_\perp$) of OCL operation ω

Semantics of static expressions

Navigation expressions

1. If $[\![\xi]\!]$ is a reference then $[\![\xi.a]\!]_{q,N,\theta} \;=\; \ell(a)$ where either
   - $[\![\xi]\!]_{q,N,\theta} \in C$ ref and $\underbrace{\sigma_q\left([\![\xi]\!]_{q,N,\theta}\right) = \ell}_{\text{state of object } \xi}$, or
   - $[\![\xi]\!]_{q,N,\theta} \in C.M$ ref and $\underbrace{\gamma_q\left([\![\xi]\!]_{q,N,\theta}\right) = (\ell, v)}_{\text{state of method occurrence } \xi}$
2. If $[\![\xi]\!]$ is a list then $[\![\xi.a]\!]_{q,N,\theta} \;=\; \vec{\ell}(a)$ where either
   - $[\![\xi]\!]_{q,N,\theta} \in C$ ref list and $\sigma_q\left([\![\xi]\!]_{q,N,\theta}\right) = \vec{\ell}$, or
   - $[\![\xi]\!]_{q,N,\theta} \in C.M$ ref list and $\gamma_q\left([\![\xi]\!]_{q,N,\theta}\right) = (\vec{\ell}, v)$

Semantics of static expressions

Iterate expressions

$$[\![\texttt{with } x_1 \in \xi_1 \texttt{ from } x_2 \in \xi_2 \texttt{ do } x_2 := \xi_3]\!]_{q,N,\theta}$$

$$=$$

$$[\![\texttt{for } x_1 \in [\![\xi_1]\!]_{q,N,\theta} \texttt{ do } x_2 := \xi_3]\!]_{q,N,\theta'} \text{ with } \theta' = \theta[x_2 := [\![\xi_2]\!]_{q,N,\theta}]$$

and where the semantics of `for`-expressions is defined by:

$$[\![\texttt{for } x_1 \in [\,] \texttt{ do } x_2 := \xi]\!]_{q,N,\theta} = [\![x_2]\!]_{q,N,\theta}$$

$$[\![\texttt{for } x_1 \in [h :: w] \texttt{ do } x_2 := \xi]\!]_{q,N,\theta} = [\![\texttt{for } x_1 \in w \texttt{ do } x_2 := \xi]\!]_{q,N,\theta''}$$

$$\text{where } \theta'' = \theta[x_2 := [\![\xi]\!]_{q,N,\theta[x_1:=h]}]$$

Example

OCL types

OCL allows sets, bags, and lists, but no nested lists.

Definition (OCL types)

OCL types are defined by the following grammar:

$$\begin{aligned} \rho &::= \text{nat} \mid \text{bool} \mid C \text{ ref} \\ \tau &::= \rho \mid \rho \text{ list} \mid \rho \text{ set} \mid \rho \text{ bag} \end{aligned}$$

OCL values

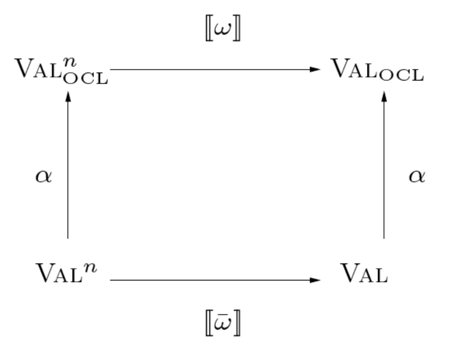
Definition (Universe of values)

The set of OCL values is defined by $\textsc{Val}_{ocl} = \bigcup_\tau \textsc{Val}^\tau$ where $\textsc{Val}^\tau$ is defined inductively as follows:

$$
\begin{aligned}
\textsc{Val}^{\text{nat}} &= \mathbb{N} \\
\textsc{Val}^{\text{bool}} &= \{\texttt{tt}, \texttt{ff}\} \\
\textsc{Val}^{C\ \text{ref}} &= \{\text{null}\} \cup \text{Oid}^C \\
\textsc{Val}^{\rho\ \text{list}} &= \{\,[\,]\,\} \cup \{h :: w \mid h \in \textsc{Val}^\rho, w \in \textsc{Val}^{\rho\ \text{list}}\} \\
\textsc{Val}^{\rho\ \text{set}} &= 2^{\textsc{Val}^\rho} \\
\textsc{Val}^{\rho\ \text{bag}} &= \textsc{Val}^\rho \rightarrow \mathbb{N}
\end{aligned}
$$

Semantics of OCL operations

For each operation $\xi_1 \rightarrow \omega(\xi_2, ldots, \xi_n)$ in OCL on sets of bags, there exists a corresponding operator $\overline{\omega}(\xi_1, \ldots, \xi_n)$ such that the following diagram commutes:



where α is an abstraction function that maps lists to sets or bags, respectively.

Abstraction of lists

For sets, α is defined by:

$$\alpha_{set}(v) \; = \; \begin{cases} \varnothing & \text{if } v = [] \\ \{\, h \,\} \cup \alpha_{set}(w) & \text{if } v = h :: w \\ v & \text{otherwise} \end{cases}$$

For bags, α is defined by:

$$\alpha_{bag}(v) \; = \; \begin{cases} \{\!|\;|\!\} & \text{if } v = [] \\ \{\!|\, h \,|\!\} \cup \alpha_{bag}(w) & \text{if } v = h :: w \\ v & \text{otherwise} \end{cases}$$

Example

Translating OCL expressions

Function δ maps an OCL expression ξ for a given object o, method occurrence m with formal parameters $\vec{p}$ into a static expressions

$$\delta_{o,m,\vec{p}}(\mathtt{self}) = o$$

$$\delta_{o,m,\vec{p}}(x) = \begin{cases} o.x & \text{if } o \in C \text{ ref and } x \in \mathrm{dom}(C.\mathit{attrs}) \\ m.x & \text{if } x \in \vec{p} \\ x & \text{otherwise} \end{cases}$$

$$\delta_{o,m,\vec{p}}(\mathtt{result}) = m.\mathsf{return}$$

$$\delta_{o,m,\vec{p}}(\xi @_{\mathtt{i}}\mathtt{pre}) = u_i$$

$$\delta_{o,m,\vec{p}}(\xi.a) = \begin{cases} \mathit{flat}(\delta_{o,m,\vec{p}}(\xi).a) & \text{if } \xi \in C \text{ ref list and } C.\mathit{attrs}(a) = \tau \text{ list} \\ \delta_{o,m,\vec{p}}(\xi).a & \text{otherwise} \end{cases}$$

$$\delta_{o,m,\vec{p}}(\omega(\xi_1,\ldots,\xi_n)) = \bar{\omega}(\delta_{o,m,\vec{p}}(\xi_1),\ldots,\delta_{o,m,\vec{p}}(\xi_n))$$

$$\delta_{o,m,\vec{p}}(\xi.\omega(\xi_1,\ldots,\xi_n)) = \bar{\omega}(\delta_{o,m,\vec{p}}(\xi),\delta_{o,m,\vec{p}}(\xi_1),\ldots,\delta_{o,m,\vec{p}}(\xi_n))$$

$$\delta_{o,m,\vec{p}}(\xi \rightarrow \omega(\xi_1,\ldots,\xi_n)) = \bar{\omega}(\delta_{o,m,\vec{p}}(\xi),\delta_{o,m,\vec{p}}(\xi_1),\ldots,\delta_{o,m,\vec{p}}(\xi_n))$$

$$\delta_{o,m,\vec{p}}(\xi_1 \rightarrow \mathtt{iterate}(x_1; x_2 = \xi_2 \mid \xi_3)) =$$
$$\quad \textsf{with } x_1 \in \delta_{o,m,\vec{p}}(\xi_1) \textsf{ from } x_2 := \delta_{o,m,\vec{p}}(\xi_2) \textsf{ do } x_2 := \delta_{o,m,\vec{p}}(\xi_3)\ .$$