On the Expressive Power of Pure OCL

María Victoria Cengarle and Alexander Knapp

Ludwig–Maximilians–Universität München
{cengarle, knapp}@informatik.uni-muenchen.de

Abstract. We provide a type inference system and a big-step operational semantics for the "Object Constraint Language" (OCL). Based on these formal systems we prove that pure OCL expressions only compute primitive recursive functions, but not recursive functions in general.

1 Introduction

The "Object Constraint Language" (OCL [10]) allows to define constraints, like invariants, for models of the "Unified Modeling Language" (UML [2]). We analyse the expressive power of OCL in terms of the set of functions that can be computed by OCL expressions. Based on the observations by Mandel and Cengarle [6], we formally demonstrate that what we call "pure OCL," i.e. OCL expressions attached to the empty UML static structure, only allows the computation of primitive recursive functions.

For this purpose we define a formal semantics of OCL 1.3 [8, Ch. 7]: We introduce a type inference system for OCL terms and define a big-step operational semantics for their evaluation; subsequently, we prove that the operational semantics satisfies a subject reduction property with respect to the type inference system. The operational semantic relation gives rise to a set-based denotational semantics that serves as the basis for demonstrating that all functions represented by pure OCL expressions are primitive recursive.

Our type inference system and operational semantics are based on the investigations by Bickford and Guaspari [1], Hamie, Howse, and Kent [5], and Richters and Gogolla [9] for OCL 1.1 and by Clark [3] for OCL 1.3. However, we improve on the treatment of the OCL types `OclAny` and `OclType`, the OCL flattening rules, undefined values, and non-determinism.

We assume a working knowledge of the OCL syntax and informal semantics. The concrete syntax of the OCL sub-language that we consider here can be found in Table 1; in particular, we omit navigation to association classes and through qualified associations, templates, package pathnames, enumerations, the types `OclExpression` and `OclState`, the functions that can be defined by using `iterate`, and pre- and post-conditions.

2 Type System

The OCL specification [8, Ch. 7] defines built-in simple types, such as `Integer`, `Boolean`, &c., and built-in parametric collection types, such as `Set`, `Sequence`,

Term ::= *Constraint* | *Expression*
Constraint ::= `context` *TypeName* `inv :` *Expression* {`inv :` *Expression*}
Expression ::= *Literal* | `self` | *VarName* | *TypeName* |
`Set {` [*Expression* {`,` *Expression*}] `}` |
`Bag {` [*Expression* {`,` *Expression*}] `}` |
`Sequence {` [*Expression* {`,` *Expression*}] `}` |
`Sequence {` *Expression* `..` *Expression* `}` |
`if` *Expression* `then` *Expression* `else` *Expression* `endif` |
`let` *VarName* `:` *TypeName* `=` *Expression* `in` *Expression* |
Expression `.` `oclAsType (` *TypeName* `)` |
Expression `.` `and (` *Expression* `)` |
Expression `.` `or (` *Expression* `)` |
Expression `->` `iterate (` *VarName* `:` *TypeName* `;`
VarName `:` *TypeName* `=` *Expression* `|`
Expression `)` |
Expression `.` *Name* |
Expression `->` *Name* |
Expression `.` *Name* `(` [*Expression* {`,` *Expression*}] `)` |
Expression `->` *Name* `(` [*Expression* {`,` *Expression*}] `)`
TypeName ::= *Name* | `Set (` *Name* `)` | `Bag (` *Name* `)` |
`Sequence (` *Name* `)` | `Collection (` *Name* `)`
Literal ::= *IntegerLiteral* | *RealLiteral* | *BooleanLiteral* | *StringLiteral*
VarName ::= *Name*

Table 1. OCL syntax fragment

&c., that take a type parameter; these built-in types may be used in any OCL term. Moreover, when an OCL term is attached to a UML static structure, it may involve all classifiers of this static structure as types and the typing information of an OCL term may rely on the structural features, i.e. attributes, and behavioural features, i.e. operations and methods, as defined in this static structure. The types are organised into a type hierarchy that extends the (reflexive and transitive closure of the) generalisation relationships between the types originating from a given static structure.

2.1 Types

The types of OCL terms over a UML static structure Ω can be summarised and grouped as follows:

T_Ω ::= A_Ω | $\overline{S}$ `(` A_Ω `)`
A_Ω ::= B | C_Ω | `OclAny` | `OclType`
B ::= `Integer` | `Real` | `Boolean` | `String` | `Void`
$\overline{S}$ ::= S | `Collection`
S ::= `Set` | `Bag` | `Sequence`

where C_Ω denotes the set of all classifiers of Ω that we may always assume does not contain `Integer`, `Real`, `Boolean`, `String`, `Void`, `OclAny`, and `OclType`.

The set B contains all built-in simple OCL types, the so-called *basic types*. The basic type `Void` is not required by the OCL specification; it denotes the empty type. The set A_Ω comprises all basic types, the *classifier types* according to Ω, and the types `OclAny` and `OclType`. The type `OclAny` is the common supertype of all basic and classifier types (see below); we consider `OclAny` to be an abstract type, though this is not stated explicitly in the OCL specification. The type `OclType` is the type of all types (as used in impredicative polymorphism [7]). Finally, the set S defines the concrete collection type functions yielding, when applied to a type parameter, a concrete *collection type*; $\overline{S}$ adds the abstract collection type function `Collection` that yields the abstract collection type.

The type parameter of a collection type may be recovered by a function

$$\mathit{flatten}_\Omega : T_\Omega \to T_\Omega$$

requiring $\mathit{flatten}_\Omega(\sigma(\tau)) = \tau$ for all $\sigma \in \overline{S}$ and $\tau \in A_\Omega$; for simplicity, we set $\mathit{flatten}_\Omega(\tau) = \tau$ if $\tau \in A_\Omega$.

Each *Literal* l has a type, written as $\mathit{type}_\Omega(l)$, such that $\mathit{type}_\Omega(n) = \texttt{Integer}$ if n is an *IntegerLiteral*, &c.

2.2 Type Hierarchy

The type hierarchy for OCL term types over a UML static structure Ω is given by the least partial order, the *subtype relation*, $\leq_\Omega$ that satisfies the following axioms:

1. for all $\tau \in T_\Omega$, $\texttt{Void} \leq_\Omega \tau$
2. for all $\tau \in B$, $\tau \leq_\Omega \texttt{OclAny}$
3. for all $\tau \in C_\Omega$, $\tau \leq_\Omega \texttt{OclAny}$
4. $\texttt{Integer} \leq_\Omega \texttt{Real}$
5. for all $\tau_1, \tau_2 \in C_\Omega$, if τ_2 generalises τ_1 with respect to Ω, then $\tau_1 \leq_\Omega \tau_2$
6. for all $\sigma \in S$ and $\tau \in A_\Omega$, $\sigma(\tau) \leq_\Omega \texttt{Collection}(\tau)$
7. for all $\sigma \in \overline{S}$ and $\tau_1, \tau_2 \in A_\Omega$, if $\tau_1 \leq_\Omega \tau_2$, then $\sigma(\tau_1) \leq_\Omega \sigma(\tau_2)$

Note that axioms (2–3) only apply to basic and classifier types, that is, in particular, $\sigma(\tau) \not\leq_\Omega \texttt{OclAny}$. On the one hand, according to [8, p. 7-8], collection types are basic types; on the other hand, following [8, p. 7-28], these types are *not* basic types. We choose the second definition, in particular because in this way the Russell paradox, that would arise from $\texttt{Set(OclAny)} \leq_\Omega \texttt{OclAny}$, is avoided (cf. [1]). Moreover, $\tau \leq_\Omega \mathit{flatten}_\Omega(\tau)$ if, and only if $\tau \in A_\Omega$.

Given types $\tau_1, \ldots, \tau_n$, we denote by $\bigsqcup_\Omega\{\tau_1, \ldots, \tau_n\}$ the least upper bound of $\tau_1, \ldots, \tau_n$ with respect to $\leq_\Omega$; simultaneously, when writing $\bigsqcup_\Omega\{\tau_1, \ldots, \tau_n\}$ we assume this least upper bound to exist. Note that $\bigsqcup_\Omega \emptyset = \texttt{Void}$.

2.3 Properties and Features

An OCL term may involve, on the one hand, the predefined *properties* of the built-in OCL types, such as $n.\texttt{+}(n')$ for adding the two numbers n and n' or $\tau.\texttt{allInstances}$ for computing all instances of a type τ. On the other hand, when attached to a non-empty UML static structure, an OCL term may also involve all *structural features* and those *behavioural features* of this static structure that are declared as *queries*, that is, as having no side-effects.

The UML specification captures the resolution of overriding and redefinition of features in a static structure by the concept of a *full descriptor* [8, pp. 2-60f.], which for each classifier yields all structural and behavioural features available for this classifier, either by direct definition or by inheritance, together with their signatures (cf. [8, pp. 7-16f.]). However, a detailed definition of full descriptors, in particular, regarding multiple inheritance, is lacking.

We abstractly axiomatise the retrieval of (overridden) properties and features by two functions on a given UML static structure Ω: The partial function

$$fd_\Omega : Name \times T_\Omega \rightharpoonup T_\Omega \times T_\Omega$$

yields, given a name a and a type τ, two types τ' and τ'' such that (a) $\tau \leq_\Omega \tau'$ and (b) the type τ' shows a structural feature or a parameter-less property with name a; in this case, we write $fd_\Omega(a, \tau) = \tau'.a : \tau''$. Moreover, given a name o, a type τ, and a sequence of types $(\tau_i)_{1 \leq i \leq n}$, the partial function

$$fd_\Omega : Name \times T_\Omega \times T_\Omega^* \rightharpoonup T_\Omega \times T_\Omega^* \times T_\Omega$$

yields a type τ', a sequence of types $(\tau_i')_{1 \leq i \leq n}$, and a type τ'' such that (a) $\tau \leq_\Omega \tau'$, $\tau_i \leq_\Omega \tau_i'$ for all $1 \leq i \leq n$, and (b) the type τ' shows a query behavioural feature or a property with parameters with name o; in this case, we write $fd_\Omega(o, \tau, (\tau_i)_{1 \leq i \leq n}) = \tau'.o : (\tau_i')_{1 \leq i \leq n} \to \tau''$.

$fd_\Omega(\texttt{supertypes}, \texttt{OclType}) = \texttt{OclType.supertypes} : \texttt{Set(OclType)}$
$fd_\Omega(\texttt{allInstances}, \tau) = \texttt{OclType.allInstances} : \texttt{Set}(\tau)$
$fd_\Omega(\texttt{first}, \texttt{Sequence}(\alpha)) = \texttt{Sequence}(\alpha)\texttt{.first} : \alpha$
$fd_\Omega(\texttt{=}, \tau, \tau') = \tau\texttt{.=} : \tau' \to \texttt{Boolean}$
$fd_\Omega(\texttt{oclIsKindOf}, \tau, \texttt{OclType}) = \tau\texttt{.oclIsKindOf} : \texttt{OclType} \to \texttt{Boolean}$
$fd_\Omega(\texttt{+}, \texttt{Integer}, \texttt{Integer}) = \texttt{Integer.+} : \texttt{Integer} \to \texttt{Integer}$
$fd_\Omega(\texttt{+}, \texttt{Integer}, \texttt{Real}) = \texttt{Real.+} : \texttt{Real} \to \texttt{Real}$
$fd_\Omega(\texttt{+}, \texttt{Real}, \texttt{Integer}) = \texttt{Real.+} : \texttt{Real} \to \texttt{Real}$
$fd_\Omega(\texttt{+}, \texttt{Real}, \texttt{Real}) = \texttt{Real.+} : \texttt{Real} \to \texttt{Real}$
$fd_\Omega(\texttt{including}, \sigma(\alpha), \alpha) = \sigma(\alpha)\texttt{.including} : \alpha \to \sigma(\alpha)$
$fd_\Omega(\texttt{union}, \sigma(\alpha), \sigma(\alpha)) = \sigma(\alpha)\texttt{.union} : \alpha \to \sigma(\alpha)$

Table 2. Typing of sample built-in OCL properties

Table 2 shows some sample axioms for the full descriptors of OCL properties, where $\tau, \tau' \in T_\Omega$ and $\alpha \in A_\Omega$; all other properties defined in the OCL specification [8, Sect. 7.8] may be added analogously. We require these axioms for all UML static structures Ω.

2.4 Type Inference

The type inference system on the one hand allows to deduce the type of a given OCL term over a given UML static structure, or whether an OCL term is well-typed. On the other hand, the inference system produces a normalised and annotated OCL term adding type information for later evaluation.

The grammar for *annotated* OCL terms transforms the grammar in Table 1 for conventional OCL terms as follows: The syntactical categories *Term*, *Constraint*, and *Expression* are replaced consistently by the syntactical categories *A-Term*, *A-Constraint*, and *A-Expression*, respectively; moreover the original *Expression* clauses

Expression `.` *Name* |
Expression `.` *Name* `(` [*Expression* {`,` *Expression*}] `)` |
Expression `->` *Name* `(` [*Expression* {`,` *Expression*}] `)`

are replaced by

A-Expression `.` *Name TypeName* |
A-Expression `.` *Name TypeName* `(` [*A-Expression* {`,` *A-Expression*}] `)` |
A-Expression `->` *Name TypeName* `(` [*A-Expression* {`,` *A-Expression*}] `)`

Annotations by *TypeName* are written as subscripts. —

A *type environment* is a finite sequence Γ of variable typings of the form $x_1 : \tau_1, \ldots, x_n : \tau_n$ with $x_i \in \mathit{VarName} \cup \{\texttt{self}\}$ and $\tau_i \in \mathit{TypeName}$ for all $1 \leq i \leq n$; for such a type environment Γ we denote $\{x_1, \ldots, x_n\}$ by $\mathrm{dom}(\Gamma)$, and τ_i by $\Gamma(x_i)$ if $x_j \neq x_i$ for all $i < j \leq n$. The empty type environment is denoted by $\emptyset$, concatenation of type environments Γ and Γ' by Γ, Γ'.

A *type environment over a UML static structure* Ω is a type environment Γ with $\Gamma(x) \in T_\Omega$ for all $x \in \mathrm{dom}(\Gamma)$; the domain of type environments over Ω is denoted by $\mathit{TypeEnv}_\Omega$. —

The type inference system consists of judgements of the form

$$\Gamma \vdash_\Omega t \triangleright \tilde{t} : \tau$$

where Ω is a UML static structure, Γ is a type environment in $\mathit{TypeEnv}_\Omega$, t is a *Term*, $\tilde{t}$ is an *A-Term*, and $\tau \in T_\Omega$. When writing such a judgement, we assume that `self`, `oclAsType`, `and`, and `or` are reserved names and that *VarName* and $T_\Omega \subseteq \mathit{TypeName}$ are disjoint. The empty type environment may be omitted.

The judgement relation $\vdash_\Omega$ is defined by the rules in Tables 3–4; a rule may only be applied if all its constituents are well-defined. The meta-variables that are used in the rules and which may be variously decorated range as follows:

$$(\mathrm{Ctxt}^{\tau}) \quad \frac{(\Gamma, \texttt{self} : \zeta \vdash_{\Omega} e_i \triangleright \tilde{e}_i : \texttt{Boolean})_{1 \le i \le n}}{\begin{array}{l}\Gamma \vdash_{\Omega} \texttt{context}\ \zeta\ (\texttt{inv:}\ e_i)_{1 \le i \le n} \triangleright \\ \qquad \texttt{context}\ \zeta\ (\texttt{inv:}\ \tilde{e}_i)_{1 \le i \le n} : \texttt{Boolean}\end{array}} \quad \text{if } \zeta \in C_{\Omega}$$

$$(\mathrm{Const}^{\tau}) \quad \vdash_{\Omega} l \triangleright l : \mathit{type}_{\Omega}(l)$$

$$(\mathrm{Self}^{\tau}) \quad \Gamma \vdash_{\Omega} \texttt{self} \triangleright \texttt{self} : \Gamma(\texttt{self})$$

$$(\mathrm{Var}^{\tau}) \quad \Gamma \vdash_{\Omega} x \triangleright x : \Gamma(x)$$

$$(\mathrm{Type}^{\tau}) \quad \vdash_{\Omega} \tau \triangleright \tau : \texttt{OclType}$$

$$(\mathrm{Coll}_1^{\tau}) \quad \frac{(\Gamma \vdash_{\Omega} e_i \triangleright \tilde{e}_i : \tau_i)_{1 \le i \le n}}{\begin{array}{l}\Gamma \vdash_{\Omega} \sigma\texttt{\{}e_1\texttt{, } \ldots\texttt{, } e_n\texttt{\}} \triangleright \\ \qquad \sigma\texttt{\{}\tilde{e}_1\texttt{, } \ldots\texttt{, } \tilde{e}_n\texttt{\}} : \sigma(\alpha)\end{array}} \quad \text{where } \alpha = \bigsqcup_{\Omega}\{\mathit{flatten}_{\Omega}(\tau_i) \mid 1 \le i \le n\}$$

$$(\mathrm{Coll}_2^{\tau}) \quad \frac{(\Gamma \vdash_{\Omega} e_i \triangleright \tilde{e}_i : \texttt{Integer})_{1 \le i \le 2}}{\Gamma \vdash_{\Omega} \texttt{Sequence\{}e_1\ \texttt{..}\ e_2\texttt{\}} \triangleright \texttt{Sequence\{}\tilde{e}_1\ \texttt{..}\ \tilde{e}_2\texttt{\}} : \texttt{Sequence(Integer)}}$$

$$(\mathrm{Cond}^{\tau}) \quad \frac{\begin{array}{c}\Gamma \vdash_{\Omega} e \triangleright \tilde{e} : \texttt{Boolean} \\ (\Gamma \vdash_{\Omega} e_i \triangleright \tilde{e}_i : \tau_i)_{1 \le i \le 2}\end{array}}{\begin{array}{l}\Gamma \vdash_{\Omega} \texttt{if}\ e\ \texttt{then}\ e_1\ \texttt{else}\ e_2\ \texttt{endif} \triangleright \\ \qquad \texttt{if}\ \tilde{e}\ \texttt{then}\ \tilde{e}_1\ \texttt{else}\ \tilde{e}_2\ \texttt{endif} : \tau\end{array}} \quad \text{where } \tau = \bigsqcup_{\Omega}\{\tau_1, \tau_2\}$$

$$(\mathrm{Let}^{\tau}) \quad \frac{\begin{array}{c}\Gamma \vdash_{\Omega} e \triangleright \tilde{e} : \tau' \\ \Gamma, x : \tau \vdash_{\Omega} e' \triangleright \tilde{e}' : \tau''\end{array}}{\Gamma \vdash_{\Omega} \texttt{let}\ x\ \texttt{:}\ \tau\ \texttt{=}\ e\ \texttt{in}\ e' \triangleright \texttt{let}\ x\ \texttt{:}\ \tau\ \texttt{=}\ \tilde{e}\ \texttt{in}\ \tilde{e}' : \tau''} \quad \text{if } \tau' \le_{\Omega} \tau$$

$$(\mathrm{Cast}^{\tau}) \quad \frac{\Gamma \vdash_{\Omega} e \triangleright \tilde{e} : \tau}{\Gamma \vdash_{\Omega} e\texttt{.oclAsType(}\tau'\texttt{)} \triangleright \tilde{e}\texttt{.oclAsType(}\tau'\texttt{)} : \tau'} \quad \begin{array}{l}\text{if } \tau \le_{\Omega} \tau' \text{ or } \tau' \le_{\Omega} \tau \\ \text{and } \tau' \neq \texttt{Void}\end{array}$$

$$(\mathrm{Par}^{\tau}) \quad \frac{(\Gamma \vdash_{\Omega} e_i \triangleright \tilde{e}_i : \texttt{Boolean})_{1 \le i \le 2}}{\Gamma \vdash_{\Omega} e_1\texttt{.}o\texttt{(}e_2\texttt{)} \triangleright \tilde{e}_1\texttt{.}o\texttt{(}\tilde{e}_2\texttt{)} : \texttt{Boolean}} \quad \text{if } o = \texttt{and} \text{ or } o = \texttt{or}$$

$$(\mathrm{Iter}^{\tau}) \quad \frac{\begin{array}{c}\Gamma \vdash_{\Omega} e \triangleright \tilde{e} : \overline{\sigma}(\alpha') \\ \Gamma \vdash_{\Omega} e' \triangleright \tilde{e}' : \tau' \\ \Gamma, x : \alpha, x' : \tau \vdash_{\Omega} e'' \triangleright \tilde{e}'' : \tau''\end{array}}{\begin{array}{l}\Gamma \vdash_{\Omega} e\texttt{->iterate(}x\ \texttt{:}\ \alpha\texttt{;}\ x'\ \texttt{:}\ \tau\ \texttt{=}\ e'\ \texttt{|}\ e''\texttt{)} \triangleright \\ \qquad \tilde{e}\texttt{->iterate(}x\ \texttt{:}\ \alpha\ \texttt{;}\ x'\ \texttt{:}\ \tau\ \texttt{=}\ \tilde{e}'\ \texttt{|}\ \tilde{e}''\texttt{)} : \tau\end{array}} \quad \begin{array}{l}\text{if } \alpha' \le_{\Omega} \alpha \text{ and} \\ \tau', \tau'' \le_{\Omega} \tau\end{array}$$

Table 3. Type inference system I

$$(\text{Feat}_1^\tau)\quad \frac{\Gamma \vdash_\Omega e \triangleright \tilde{e} : \tau}{\Gamma \vdash_\Omega e.a \triangleright \tilde{e}.a_{\tau'} : \tau''} \quad \text{if } fd_\Omega(a,\tau) = \tau'.a : \tau''$$

$$(\text{Feat}_2^\tau)\quad \frac{\Gamma \vdash_\Omega e \triangleright \tilde{e} : \tau \quad (\Gamma \vdash_\Omega e_i \triangleright \tilde{e}_i : \tau_i)_{1\le i\le n}}{\Gamma \vdash_\Omega e.o(e_1, \ldots, e_n) \triangleright \tilde{e}.o_{\tau''}(\tilde{e}_1, \ldots, \tilde{e}_n) : \tau''} \quad \text{if } fd_\Omega(o,\tau,(\tau_i)_{1\le i\le n}) = \tau'.o : (\tau_i')_{1\le i\le n} \to \tau''$$

$$(\text{Prop}_1^\tau)\quad \frac{\Gamma \vdash_\Omega e \triangleright \tilde{e} : \overline{\sigma}(\alpha)}{\Gamma \vdash_\Omega e\texttt{->}a \triangleright \tilde{e}\texttt{->}a_{\tau'} : \tau''} \quad \text{if } fd_\Omega(a,\overline{\sigma}(\alpha)) = \tau'.a : \tau''$$

$$(\text{Prop}_2^\tau)\quad \frac{\Gamma \vdash_\Omega e \triangleright \tilde{e} : \overline{\sigma}(\alpha) \quad (\Gamma \vdash_\Omega e_i \triangleright \tilde{e}_i : \tau_i)_{1\le i\le n}}{\Gamma \vdash_\Omega e\texttt{->}o(e_1, \ldots, e_n) \triangleright \tilde{e}\texttt{->}o_{\tau''}(\tilde{e}_1, \ldots, \tilde{e}_n) : \tau''} \quad \text{if } fd_\Omega(o,\overline{\sigma}(\alpha),(\tau_i)_{1\le i\le n}) = \tau'.o : (\tau_i')_{1\le i\le n} \to \tau''$$

$$(\text{Sing}_1^\tau)\quad \frac{\Gamma \vdash_\Omega e \triangleright \tilde{e} : \alpha' \quad \Gamma \vdash_\Omega e' \triangleright \tilde{e}' : \tau' \quad \Gamma, x : \alpha, x' : \tau \vdash_\Omega e'' \triangleright \tilde{e}'' : \tau''}{\Gamma \vdash_\Omega e\texttt{->iterate(}\, x : \alpha\texttt{;}\ x' : \tau = e' \mid e''\texttt{)} \triangleright \texttt{Set\{}\tilde{e}\texttt{\}->iterate(}\, x : \alpha\texttt{;}\ x' : \tau = \tilde{e}' \mid \tilde{e}''\texttt{)} : \tau} \quad \text{if } \alpha' \le_\Omega \alpha \text{ and } \tau', \tau'' \le_\Omega \tau$$

$$(\text{Sing}_2^\tau)\quad \frac{\Gamma \vdash_\Omega e \triangleright \tilde{e} : \alpha \quad (\Gamma \vdash_\Omega e_i \triangleright \tilde{e}_i : \tau_i)_{1\le i\le n}}{\Gamma \vdash_\Omega e\texttt{->}o(e_1, \ldots, e_n) \triangleright \texttt{Set\{}\tilde{e}\texttt{\}->}o_\tau(\tilde{e}_1, \ldots, \tilde{e}_n) : \tau} \quad \text{if } fd_\Omega(o,\texttt{Set}(\alpha),(\tau_i)_{1\le i\le n}) = \tau'.o : (\tau_i')_{1\le i\le n} \to \tau$$

$$(\text{Short}_1^\tau)\quad \frac{\Gamma \vdash_\Omega e \triangleright \tilde{e} : \overline{\sigma}(\alpha)}{\Gamma \vdash_\Omega e.a \triangleright \tilde{e}\texttt{->iterate(i : }\alpha\texttt{; a : }\sigma(\alpha')\texttt{ = }\sigma\texttt{\{\} | a->including}_{\sigma(\alpha')}\texttt{(i.}a_\tau\texttt{)) : }\sigma(\alpha')}$$

if $fd_\Omega(a,\alpha) = \tau.a : \alpha'$, and $\overline{\sigma} \neq \texttt{Sequence}$ and $\sigma = \texttt{Bag}$ or $\sigma = \overline{\sigma} = \texttt{Sequence}$

$$(\text{Short}_2^\tau)\quad \frac{\Gamma \vdash_\Omega e \triangleright \tilde{e} : \overline{\sigma}(\alpha)}{\Gamma \vdash_\Omega e.a \triangleright \tilde{e}\texttt{->iterate(i : }\alpha\texttt{; a : }\sigma(\alpha')\texttt{ = }\sigma\texttt{\{\} | a->union}_{\sigma(\alpha')}\texttt{(i.}a_\tau\texttt{)) : }\sigma(\alpha')}$$

if $fd_\Omega(a,\alpha) = \tau.a : \sigma(\alpha')$

$$(\text{Weak}^\tau)\quad \frac{\Gamma \vdash_\Omega e \triangleright \tilde{e} : \tau}{\Gamma', \Gamma \vdash_\Omega e \triangleright \tilde{e} : \tau}$$

Table 4. Type inference system II

$l \in \mathit{Literal}$; $\alpha \in A_\Omega$, $\zeta \in C_\Omega$, $\sigma \in S$, $\overline{\sigma} \in \overline{S}$, $\tau \in T_\Omega$; $x \in \mathit{VarName}$; $a, o \in \mathit{Name}$; $e \in \mathit{Expression}$, $\tilde{e} \in \mathit{A\text{-}Expression}$. —

The rules follow the OCL specification [8, Ch. 7] as closely as possible. The rule (Coll_1^τ) in Table 3 provides a unique type for the empty concrete collections (in contrast to the type system by Clark [3]). The rule (Coll_2^τ) is the so-called "flattening" rule for nested collections; for a motivation see [8, p. 7-20]. The type of a conditional expression, as given by the (Cond^τ) rule, differs from what is stated in [8, p. 7-35]: there, independently of the type of e_2, the type of e_1 is assumed to be the type of the whole expression. For the casting rule (Cast^τ) in Table 3 see [8, pp. 7-10, 7-16, 7-29]; note, however, that this rule does not allow for arbitrary expressions resulting in a type as the argument for `oclAsType`, since this would imply term-dependent types as, for example, in

```
5.oclAsType(if 1.=(2) then Real else Integer endif) .
```

The rules (Sing_1^τ–Sing_2^τ) in Table 4 are the so-called "singleton" rules; see [8, p. 7-13], allowing to apply collection properties to non-collection expressions. The (Short_1^τ–Short_2^τ) rules in Table 4 define the shorthand notation for parameter-less properties on members of collections (see [8, p. 7-24]) combined with flattening (cf. [8, p. 7-20]); note that $\overline{\sigma}$ can indeed be `Collection`, if e is for instance the expression

```
if 1.=(2) then Set{ -1, 2 } else Bag{ -1, -1, 2 } endif .
```

This shorthand notation could be extended to parameterised properties of members of collections. However, this extension leads to a non-deterministic type system; e.g.,

```
Set{ 1, 2 }.oclIsTypeOf(Integer)
```

may have the incomparable types `Set(Boolean)` or `Boolean`. The only structural rule is the weakening rule (Weak^τ), that allows to extend type environments. In particular, there is no subsumption rule which would interfere with the overriding of properties and features (cf. e.g. [4]). —

The type inference system entails unique annotations and types:

Lemma 1. *Let Ω be a UML static structure, Γ a type environment, and t a Term. If $\Gamma \vdash_\Omega t \triangleright \tilde{t} : \tau$ and $\Gamma \vdash_\Omega t \triangleright \tilde{t}' : \tau'$ for some A-Term's $\tilde{t}$ and $\tilde{t}'$ and types τ and τ' then $\tilde{t} = \tilde{t}'$ and $\tau = \tau'$.*

Proof. All rules in Tables 3–4 are deterministic, i.e., entail unique annotated terms and types. The only overlapping rules are on the one hand (Self^τ), (Var^τ), and (Type^τ) and, on the other hand, the rule (Feat_1^τ) and the rules (Short_1^τ–Short_2^τ); however, no conflicts can occur, since, for the first pairs, the syntactical conventions apply, and, for the second pairs, there are no parameter-less properties for collection types. □

We call an OCL term t *well-typed* over a UML static structure Ω and a type environment Γ if there are $\tilde{t}$ and τ such that $\Gamma \vdash_\Omega t \triangleright \tilde{t} : \tau$; the term t is well-typed over Ω if it is *well-typed* over Ω and the empty type environment $\emptyset$.

3 Operational Semantics

We define a big-step operational semantics for OCL terms; our semantics resembles the system of Clark [3]. This operational semantics is type-sound with respect to the type system of the previous section.

3.1 Semantic Domains

Given a UML static structure Ω, we assume a primitive semantic domain Object_Ω for all instances conforming to the classifiers in Ω. This domain is equipped with a function

$$\mathit{actual}_\Omega : \mathit{Object}_\Omega \to C_\Omega$$

yielding for a given object its *actual type*.

The *values* of OCL terms are defined by OCL ground terms for Ω:

$$\begin{aligned}
\mathit{Value}_\Omega ::= {} & \mathit{SimpleValue}_\Omega \mid \\
& \texttt{Set \{}\ \mathit{SimpleValueList}_\Omega\ \texttt{\}} \mid \\
& \texttt{Bag \{}\ \mathit{SimpleValueList}_\Omega\ \texttt{\}} \mid \\
& \texttt{Sequence \{}\ \mathit{SimpleValueList}_\Omega\ \texttt{\}} \\
\mathit{SimpleValueList}_\Omega ::= {} & [\mathit{SimpleValue}_\Omega\ \{\texttt{,}\ \mathit{SimpleValue}_\Omega\}] \\
\mathit{SimpleValue}_\Omega ::= {} & \mathit{Literal} \mid \mathit{Object}_\Omega \mid T_\Omega
\end{aligned}$$

We assume suitably axiomatised arithmetical, boolean, &c. functions and relations on these ground terms such that, e.g., $\texttt{1} + \texttt{1} = \texttt{2}$, $\texttt{false} \wedge \texttt{true} = \texttt{false}$, $\texttt{Set\{ 1, 2 \}} = \texttt{Set\{ 2, 1 \}}$, $\texttt{1} \leq \texttt{2}$, $\texttt{Integer} \leq_\Omega \texttt{Real}$, &c.

For collection values, we use a map

$$\mathit{flatten}_\Omega : \mathit{Value}_\Omega \to \mathit{SimpleValue}_\Omega^*$$

defined similarly to the type function in Sect. 2.1, by $\mathit{flatten}_\Omega(\sigma\texttt{\{}v_1\texttt{, \ldots, }v_n\texttt{\}}) = v_1 \cdots v_n$, and $\mathit{flatten}_\Omega(v) = v$, if $v \in \mathit{SimpleValue}_\Omega$. Collection values are constructed by a function

$$\mathit{make}_\Omega : S \times \mathit{Value}_\Omega^* \to \mathit{Value}_\Omega$$

such that $\mathit{make}_\Omega(\sigma, v_1 \cdots v_n) = \sigma\texttt{\{}v_1\texttt{, \ldots, }v_n\texttt{\}}$ if $\sigma \in S$ and $v_i \in \mathit{SimpleValue}_\Omega$ for all $1 \leq i \leq n$, and, if $v_i \in \mathit{Value}_\Omega \setminus \mathit{SimpleValue}_\Omega$ for some $1 \leq i \leq n$ then $\mathit{make}_\Omega(\sigma, v_1 \cdots v_n) = \mathit{make}_\Omega(\sigma, \mathit{flatten}_\Omega(v_1) \cdots \mathit{flatten}_\Omega(v_n))$; if $\sigma = \texttt{Set}$, repetitions in $\mathit{flatten}(v_1) \cdots \mathit{flatten}(v_n)$ are discarded. If $n = 0$, we write $\mathit{make}_\Omega(S, \emptyset)$; more generally, we also denote $\mathit{make}_\Omega(\sigma, v_1 \cdots v_n)$ by $\mathit{make}_\Omega(\sigma, \{v_1, \ldots, v_n\})$. For sequences, we additionally use a map

$$\mathit{make}_\Omega : \mathit{IntegerLiteral} \times \mathit{IntegerLiteral} \to \mathit{Value}_\Omega$$

such that, for $n_1, n_2 \in \mathit{IntegerLiteral}$, $\mathit{make}_\Omega(n_1, n_2) = \texttt{Sequence\{}n_1\texttt{, }n_1 + \texttt{1, \ldots, }n_2\texttt{\}}$, if $n_1 \leq n_2$, and $\mathit{make}_\Omega(n_1, n_2) = \texttt{Sequence\{\}}$, otherwise; we more conveniently write $\mathit{make}_\Omega(\texttt{Sequence}, n_1 \mathinner{..} n_2)$ for $\mathit{make}_\Omega(n_1, n_2)$.

A collection value v has a sequence value *representation* v', written as $v \rightsquigarrow v'$, if $make_\Omega(\texttt{Sequence}, flatten_\Omega(v'')) = v'$ for some $v'' = v$. In general, a collection value has several different sequence value representations; e.g. `Set{ 1, 2 }` $\rightsquigarrow$ `Sequence{ 1, 2 }`, but also `Set{ 1, 2 }` $\rightsquigarrow$ `Sequence{ 2, 1 }`.

We define a relation between values and types

$$:_\Omega \subseteq Value_\Omega \times T_\Omega$$

extending $actual_\Omega$ above and $type_\Omega$ of Sect. 2.1 by subsumption, such that

1. For $l \in Literal$, $l :_\Omega type(l)$
2. For $\tau \in T_\Omega$, $\tau :_\Omega \texttt{OclType}$
3. For $o \in Object_\Omega$, $o :_\Omega actual_\Omega(o)$
4. $\sigma\texttt{\{\}} :_\Omega \sigma\texttt{(Void)}$
5. If $v_i :_\Omega \alpha$ for all $1 \leq i \leq n$, then $\sigma\texttt{\{}v_1\texttt{, } \ldots \texttt{, } v_n\texttt{\}} :_\Omega \sigma\texttt{(}\alpha\texttt{)}$
6. If $v :_\Omega \tau$ and $\tau \leq_\Omega \tau'$, then $v :_\Omega \tau'$

In particular, by rule (5), we have that if $v :_\Omega \tau$ and $flatten(v) = v_1 \cdots v_n$ then $v_i :_\Omega flatten(v_i)$ for all $1 \leq i \leq n$. Note, however, that there is no $v \in Value_\Omega$ with $v :_\Omega \texttt{Void}$ and, moreover, that there is no type $\tau \in T_\Omega$ such that `Set{ 1, Boolean }` $:_\Omega \tau$.

3.2 Systems

An OCL term is evaluated in the context of a set of instances conforming to the underlying UML static structure. For a given static structure Ω, we call a set of instances $\omega \subseteq Object_\Omega$ a *system* conforming to Ω, if with

$$Value^\omega_\Omega = \{v \in Value_\Omega \mid v \in Object_\Omega \supset v \in \omega\}$$
$$Result^\omega_\Omega = Value^\omega_\Omega \cup \{\bot\}$$

the following holds:

1. For every $a \in Name$ with $fd_\Omega(a, \tau) = \tau'.a : \tau''$ there is a function

   $$attr^\omega_\Omega(a, \tau') : Value^\omega_\Omega \rightarrow Result^\omega_\Omega$$

   such that if $v \in Value^\omega_\Omega$ and $v :_\Omega \tau$ and $attr^\omega_\Omega(a, \tau')(v) = v' \in Value^\omega_\Omega$ then $v' :_\Omega \tau''$.
2. For every $o \in Name$ with $fd_\Omega(o, \tau, (\tau_i)_{1 \leq i \leq n}) = \tau'.o : (\tau'_i)_{1 \leq i \leq n} \rightarrow \tau''$ there is a function

   $$meth^\omega_\Omega(o, \tau'') : Value^\omega_\Omega \times (Value^\omega_\Omega)^* \rightarrow Result^\omega_\Omega$$

   such that if $v, v_1, \ldots, v_n \in Value^\omega_\Omega$ and $v :_\Omega \tau$ and $v_i :_\Omega \tau_i$ for all $1 \leq i \leq n$ and $meth^\omega_\Omega(o, \tau'')(v, (v_i)_{1 \leq i \leq n}) = v' \in Value^\omega_\Omega$ then $v' :_\Omega \tau''$.

$$attr^{\omega}_{\Omega}(\texttt{supertypes}_{\texttt{OclType}}, \tau) = make_{\Omega}(\texttt{Set}, \{\tau' \mid \tau \leq_{\Omega} \tau'\})$$
$$attr^{\omega}_{\Omega}(\texttt{allInstances}_{\texttt{OclType}}, \texttt{Boolean}) = \texttt{Set\{ true, false \}}$$
$$attr^{\omega}_{\Omega}(\texttt{allInstances}_{\texttt{OclType}}, \texttt{Set(Boolean)}) = \texttt{Set\{ true, false \}}$$
$$attr^{\omega}_{\Omega}(\texttt{allInstances}_{\texttt{OclType}}, \zeta) = make_{\Omega}(\texttt{Set}, \omega(\zeta))$$
$$attr^{\omega}_{\Omega}(\texttt{allInstances}_{\texttt{OclType}}, \texttt{Set}(\zeta)) = make_{\Omega}(\texttt{Set}, \omega(\zeta))$$
$$attr^{\omega}_{\Omega}(\texttt{first}_{\texttt{Sequence}(\alpha)}, \texttt{Sequence}(v_1, \ldots, v_n)) = v_1$$
$$meth^{\omega}_{\Omega}(=_{\texttt{Boolean}}, v, v') = (v = v')$$
$$meth^{\omega}_{\Omega}(\texttt{oclIsKindOf}_{\texttt{Boolean}}, v, \tau) = v :_{\Omega} \tau$$
$$meth^{\omega}_{\Omega}(+_{\texttt{Integer}}, v, v') = v + v'$$
$$meth^{\omega}_{\Omega}(+_{\texttt{Real}}, v, v') = v + v'$$
$$meth^{\omega}_{\Omega}(\texttt{including}_{\sigma(\alpha)}, v, v') = make_{\Omega}(\sigma, v\ v')$$
$$meth^{\omega}_{\Omega}(\texttt{union}_{\sigma(\alpha)}, v, v') = make_{\Omega}(\sigma, v\ v')$$

Table 5. Semantics of sample built-in OCL properties

The domain $Value^{\omega}_{\Omega}$ contains all values of OCL terms for a system ω and $Result^{\omega}_{\Omega}$ extends this domain by an undefined result $\bot$. Condition (1) requires type and system preserving functions retrieving the values of structural features or parameter-less properties, condition (2) requires analogous functions executing behavioural features and properties with parameters.

Table 5 contains some sample axioms for the retrieval of OCL properties and features. In particular, the definitions for `allInstances` take into account the necessary flattening of the results. We more conveniently write $attr^{\omega}_{\Omega}(a_{\tau}, v)$ for $attr^{\omega}_{\Omega}(a, \tau)(v)$, and, similarly, we abbreviate $meth^{\omega}_{\Omega}(o, \tau)(v, (v_i)_{1 \leq i \leq n})$ by $meth^{\omega}_{\Omega}(o_{\tau}, v, (v_i)_{1 \leq i \leq n})$. Moreover, for a classifier $\zeta \in C_{\Omega}$, we write $\omega(\zeta)$ for the set $\{v \in \omega \mid v :_{\Omega} \zeta\}$.

3.3 Operational Rules

The operational semantics evaluates annotated OCL terms in the context of a system and some variable assignments.

A *variable environment* is a finite sequence γ of variable assignments of the form $x_1 \mapsto v_1, \ldots, x_n \mapsto v_n$ with $x_i \in VarName \cup \{\texttt{self}\}$ and $v_i \in Value_{\Omega}$ for all $1 \leq i \leq n$; for such a variable environment we write $\mathrm{dom}(\gamma) = \{x_1, \ldots, x_n\}$ and $\gamma(x_i) = v_i$ if $x_i \neq x_j$ for all $i < j \leq n$. The empty variable environment is denoted by $\emptyset$, concatenation of variable environments γ and γ' by γ, γ'.

A *variable environment* over a UML static structure Ω and a system ω conforming to Ω is a variable environment γ with $\gamma(x) \in Value^{\omega}_{\Omega}$ for all $x \in \mathrm{dom}(\gamma)$. The domain of variable environments over Ω and ω is denoted by $VarEnv^{\omega}_{\Omega}$. —

The operational semantics consists of judgements of the form

$$\gamma \vdash^{\omega}_{\Omega} \tilde{t} \downarrow \overline{v}$$

$$(\text{Ctxt}^{\downarrow}) \quad \frac{(\gamma, \texttt{self} \mapsto v \vdash^{\omega}_{\Omega} \tilde{e}_i \downarrow v_{i,v})_{1 \le i \le n, v \in \omega(\zeta)}}{\gamma \vdash^{\omega}_{\Omega} \texttt{context}\ \zeta\ (\texttt{inv:}\ \tilde{e}_i)_{1 \le i \le n} \downarrow \bigwedge_{i,v} v_{i,v}}$$

$$(\text{Const}^{\downarrow}) \quad \gamma \vdash^{\omega}_{\Omega} l \downarrow l$$

$$(\text{Self}^{\downarrow}) \quad \gamma \vdash^{\omega}_{\Omega} \texttt{self} \downarrow \gamma(\texttt{self})$$

$$(\text{Var}^{\downarrow}) \quad \gamma \vdash^{\omega}_{\Omega} x \downarrow \gamma(x)$$

$$(\text{Type}^{\downarrow}) \quad \gamma \vdash^{\omega}_{\Omega} \tau \downarrow \tau$$

$$(\text{Coll}^{\downarrow}_1) \quad \frac{(\gamma \vdash^{\omega}_{\Omega} \tilde{e}_i \downarrow v_i)_{1 \le i \le n}}{\gamma \vdash^{\omega}_{\Omega} \sigma\texttt{\{}\tilde{e}_1\texttt{, } \ldots\texttt{, } \tilde{e}_n\texttt{\}} \downarrow \mathit{make}_{\Omega}(\sigma, v_1 \cdots v_n)}$$

$$(\text{Coll}^{\downarrow}_2) \quad \frac{(\gamma \vdash^{\omega}_{\Omega} \tilde{e}_i \downarrow v_i)_{1 \le i \le 2}}{\gamma \vdash^{\omega}_{\Omega} \texttt{Sequence\{}\tilde{e}_1\ \texttt{..}\ \tilde{e}_2\texttt{\}} \downarrow \mathit{make}_{\Omega}(\texttt{Sequence}, v_1 \,..\, v_2)}$$

$$(\text{Cond}^{\downarrow}_1) \quad \frac{\gamma \vdash^{\omega}_{\Omega} \tilde{e} \downarrow \texttt{true} \quad (\gamma \vdash^{\omega}_{\Omega} \tilde{e}_i \downarrow v_i)_{1 \le i \le 2}}{\gamma \vdash^{\omega}_{\Omega} \texttt{if}\ \tilde{e}\ \texttt{then}\ \tilde{e}_1\ \texttt{else}\ \tilde{e}_2\ \texttt{endif} \downarrow v_1}$$

$$(\text{Cond}^{\downarrow}_2) \quad \frac{\gamma \vdash^{\omega}_{\Omega} \tilde{e} \downarrow \texttt{false} \quad (\gamma \vdash^{\omega}_{\Omega} \tilde{e}_i \downarrow v_i)_{1 \le i \le 2}}{\gamma \vdash^{\omega}_{\Omega} \texttt{if}\ \tilde{e}\ \texttt{then}\ \tilde{e}_1\ \texttt{else}\ \tilde{e}_2\ \texttt{endif} \downarrow v_2}$$

$$(\text{Let}^{\downarrow}) \quad \frac{\gamma \vdash^{\omega}_{\Omega} \tilde{e} \downarrow v \quad \gamma, x \mapsto v \vdash^{\omega}_{\Omega} \tilde{e}' \downarrow v'}{\gamma \vdash^{\omega}_{\Omega} \texttt{let}\ x\ \texttt{:}\ \tau\ \texttt{=}\ \tilde{e}\ \texttt{in}\ \tilde{e}' \downarrow v'}$$

$$(\text{Cast}^{\downarrow}_1) \quad \frac{\gamma \vdash^{\omega}_{\Omega} \tilde{e} \downarrow v}{\gamma \vdash^{\omega}_{\Omega} \tilde{e}\texttt{.oclAsType(}\tau\texttt{)} \downarrow v} \quad \text{if } v :_{\Omega} \tau$$

$$(\text{Cast}^{\downarrow}_2) \quad \frac{\gamma \vdash^{\omega}_{\Omega} \tilde{e} \downarrow v}{\gamma \vdash^{\omega}_{\Omega} \tilde{e}\texttt{.oclAsType(}\tau\texttt{)} \downarrow \bot} \quad \text{if } v \not:_{\Omega} \tau$$

Table 6. Operational semantics I

$$(\text{Par}_1^{\downarrow}) \quad \frac{(\gamma \vdash^{\omega}_{\Omega} \tilde{e}_i \downarrow v_i)_{1\leq i\leq 2}}{\gamma \vdash^{\omega}_{\Omega} \tilde{e}_1\texttt{.and(}\tilde{e}_2\texttt{)} \downarrow v_1 \wedge v_2} \qquad (\text{Par}_4^{\downarrow}) \quad \frac{(\gamma \vdash^{\omega}_{\Omega} \tilde{e}_i \downarrow v_i)_{1\leq i\leq 2}}{\gamma \vdash^{\omega}_{\Omega} \tilde{e}_1\texttt{.or(}\tilde{e}_2\texttt{)} \downarrow v_1 \vee v_2}$$

$$(\text{Par}_2^{\downarrow}) \quad \frac{\gamma \vdash^{\omega}_{\Omega} \tilde{e}_1 \downarrow \texttt{false}}{\gamma \vdash^{\omega}_{\Omega} \tilde{e}_1\texttt{.and(}\tilde{e}_2\texttt{)} \downarrow \texttt{false}} \qquad (\text{Par}_5^{\downarrow}) \quad \frac{\gamma \vdash^{\omega}_{\Omega} \tilde{e}_1 \downarrow \texttt{true}}{\gamma \vdash^{\omega}_{\Omega} \tilde{e}_1\texttt{.or(}\tilde{e}_2\texttt{)} \downarrow \texttt{true}}$$

$$(\text{Par}_3^{\downarrow}) \quad \frac{\gamma \vdash^{\omega}_{\Omega} \tilde{e}_2 \downarrow \texttt{false}}{\gamma \vdash^{\omega}_{\Omega} \tilde{e}_1\texttt{.and(}\tilde{e}_2\texttt{)} \downarrow \texttt{false}} \qquad (\text{Par}_6^{\downarrow}) \quad \frac{\gamma \vdash^{\omega}_{\Omega} \tilde{e}_2 \downarrow \texttt{true}}{\gamma \vdash^{\omega}_{\Omega} \tilde{e}_1\texttt{.or(}\tilde{e}_2\texttt{)} \downarrow \texttt{true}}$$

$$(\text{Par}_7^{\downarrow}) \quad \frac{(\gamma \vdash^{\omega}_{\Omega} \tilde{e}_i \downarrow \overline{v}_i)_{1\leq i\leq 2}}{\gamma \vdash^{\omega}_{\Omega} \tilde{e}_1\texttt{.and(}\tilde{e}_2\texttt{)} \downarrow \bot} \quad \begin{array}{l}\text{if } \overline{v}_1 \neq \texttt{false} \text{ and } \overline{v}_2 = \bot \text{ or} \\ \overline{v}_1 = \bot \text{ and } \overline{v}_2 \neq \texttt{false}\end{array}$$

$$(\text{Par}_8^{\downarrow}) \quad \frac{(\gamma \vdash^{\omega}_{\Omega} \tilde{e}_i \downarrow \overline{v}_i)_{1\leq i\leq 2}}{\gamma \vdash^{\omega}_{\Omega} \tilde{e}_1\texttt{.or(}\tilde{e}_2\texttt{)} \downarrow \bot} \quad \begin{array}{l}\text{if } \overline{v}_1 \neq \texttt{true} \text{ and } \overline{v}_2 = \bot \text{ or} \\ \overline{v}_1 = \bot \text{ and } \overline{v}_2 \neq \texttt{true}\end{array}$$

$$(\text{Iter}^{\downarrow}) \quad \frac{\begin{array}{c}\gamma \vdash^{\omega}_{\Omega} \tilde{e} \downarrow v \\ \gamma \vdash^{\omega}_{\Omega} \tilde{e}' \downarrow v'_0 \\ (\gamma, x \mapsto v_i, x' \mapsto v'_{i-1} \vdash^{\omega}_{\Omega} \tilde{e}'' \downarrow v'_i)_{1\leq i\leq n}\end{array}}{\gamma \vdash^{\omega}_{\Omega} \tilde{e}\texttt{->iterate(}x\texttt{ : }\alpha\texttt{; }x'\texttt{ : }\tau\texttt{ = }\tilde{e}'\texttt{ | }\tilde{e}''\texttt{)} \downarrow v'_n}$$

$$\text{if } v \rightsquigarrow \texttt{Sequence\{}v_1\texttt{, } \ldots \texttt{, } v_n\texttt{\}}$$

$$(\text{Feat}_1^{\downarrow}) \quad \frac{\gamma \vdash^{\omega}_{\Omega} \tilde{e} \downarrow v}{\gamma \vdash^{\omega}_{\Omega} \tilde{e}.a_{\tau} \downarrow \overline{v}'} \quad \text{where } attr^{\omega}_{\Omega}(a_{\tau}, v) = \overline{v}'$$

$$(\text{Feat}_2^{\downarrow}) \quad \frac{\begin{array}{c}\gamma \vdash^{\omega}_{\Omega} \tilde{e} \downarrow v \\ (\gamma \vdash^{\omega}_{\Omega} \tilde{e}_i \downarrow v_i)_{1\leq i\leq n}\end{array}}{\gamma \vdash^{\omega}_{\Omega} \tilde{e}.o_{\tau}\texttt{(}\tilde{e}_1\texttt{, } \ldots \texttt{, } \tilde{e}_n\texttt{)} \downarrow \overline{v}'} \quad \text{where } meth^{\omega}_{\Omega}(o_{\tau}, v, (v_i)_{1\leq i\leq n}) = \overline{v}'$$

$$(\text{Prop}_1^{\downarrow}) \quad \frac{\gamma \vdash^{\omega}_{\Omega} \tilde{e} \downarrow v}{\gamma \vdash^{\omega}_{\Omega} \tilde{e}\texttt{->}a_{\tau} \downarrow \overline{v}'} \quad \text{where } attr^{\omega}_{\Omega}(a_{\tau}, v) = \overline{v}'$$

$$(\text{Prop}_2^{\downarrow}) \quad \frac{\begin{array}{c}\gamma \vdash^{\omega}_{\Omega} \tilde{e} \downarrow v \\ (\gamma \vdash^{\omega}_{\Omega} \tilde{e}_i \downarrow v_i)_{1\leq i\leq n}\end{array}}{\gamma \vdash^{\omega}_{\Omega} \tilde{e}\texttt{->}o_{\tau}\texttt{(}\tilde{e}_1\texttt{, } \ldots \texttt{, } \tilde{e}_n\texttt{)} \downarrow \overline{v}'} \quad \text{where } meth^{\omega}_{\Omega}(o_{\tau}, v, (v_i)_{1\leq i\leq n}) = \overline{v}'$$

Table 7. Operational semantics II

where Ω is a UML static structure, ω is a system conforming to Ω, γ is a variable environment in $\mathit{VarEnv}^{\omega}_{\Omega}$, $\tilde{t}$ is an *A-Term*, and $\overline{v}$ is a $\mathit{Result}^{\omega}_{\Omega}$.

The judgement relation $\vdash^{\omega}_{\Omega}$ is defined by the rules in Tables 6–7; a rule may only be applied if all its constituents are well-defined. The meta-variables range as follows: $l \in \mathit{Literal}$; $\alpha \in A_{\Omega}$, $\zeta \in C_{\Omega}$, $\sigma \in S$, $\tau \in T_{\Omega}$; $x \in \mathit{VarName}$; $a, o \in \mathit{Name}$; $v \in \mathit{Value}^{\omega}_{\Omega}$, $\overline{v} \in \mathit{Result}^{\omega}_{\Omega}$; $\tilde{e} \in \mathit{A\text{-}Expression}$.

We additionally adopt the following general *strictness convention* that applies to all rules with the single exception of the rules ($\mathrm{Par}^{\downarrow}_{1}$–$\mathrm{Par}^{\downarrow}_{8}$) in Table 7: if $\bot$ occurs as a result in a judgement of a premise of some rule, the whole term evaluates to $\bot$. —

The operational rules are presented in close correspondence to the typing rules in Tables 3–4. All rules, except the rules ($\mathrm{Par}^{\downarrow}_{1}$–$\mathrm{Par}^{\downarrow}_{6}$) in Table 7 require of all sub-terms to be fully evaluated and to result in a value in Value_{Ω} in order to deliver a value for a term. In particular, this makes for a strict conditional; on the other hand, the (Par) rules yield parallel `Boolean` properties `and` and `or`; see [8, Sect. 7.4.9]. The only rules that introduce the undefined result $\bot$ are the ($\mathrm{Cast}^{\downarrow}_{2}$) rule in Table 6 (cf. [8, p. 7-29]) and, possibly, ($\mathrm{Feat}^{\downarrow}_{1}$–$\mathrm{Feat}^{\downarrow}_{2}$) and ($\mathrm{Prop}^{\downarrow}_{1}$–$\mathrm{Prop}^{\downarrow}_{2}$) in Table 6.

3.4 Subject reduction

The operational semantics is inherently non-deterministic. In particular, the well-typed expression

```
Set{ 1, 1.2 }->iterate(i : OclAny;
                       a : Sequence(OclAny) = Sequence{} |
                       a->append(i))->first.oclAsType(Integer)
```

may evaluate to `1`, if `Set{ 1, 1.2 }` is represented by `Sequence{ 1, 1.2 }`; or it may evaluate to $\bot$, if `Set{ 1, 1.2 }` is represented by `Sequence{ 1.2, 1 }`.

Thus the operational semantics is not type sound in the strict sense, i.e., converging well-typed terms do not always yield a result of the expected type. However, if the operational semantics reduces an OCL term of inferred type τ to some value then this value is indeed of type τ, i.e., the operational semantics in Sect. 3.3 satisfies the subject reduction property with respect to the type inference system in Sect. 2.4.

In order to state and prove this result, we say that, for a UML static structure Ω, a variable environment γ *conforms to* a type environment Γ if $\mathrm{dom}(\gamma) \supseteq \mathrm{dom}(\Gamma)$ and $\gamma(x) :_{\Omega} \Gamma(x)$ for all $x \in \mathrm{dom}(\gamma)$.

Proposition 1. *Let Ω be a UML static structure and ω a system conforming to Ω; let Γ be a type environment and γ a variable environment conforming to Γ; let t be a Term and $\tilde{t}$ an A-Term; let $\tau \in T_{\Omega}$ and $v \in \mathit{Value}^{\omega}_{\Omega}$. If $\Gamma \vdash_{\Omega} t \triangleright \tilde{t} : \tau$ and $\gamma \vdash^{\omega}_{\Omega} \tilde{t} \downarrow v$, then $v :_{\Omega} \tau$.*

Proof. We perform an induction over the proof tree height n for $\Gamma \vdash_{\Omega} t \triangleright \tilde{t} : \tau$.

If $n = 0$, nothing has to be proven. Thus, let $n > 0$ and let the claim be proven for all proof tree heights $n' < n$. We proceed by case analysis on the last proof step for $\Gamma \vdash_\Omega t \triangleright \tilde{t} : \tau$:

(Ctxt$^\tau$): Then $\tilde{t} = \texttt{context}\ \zeta\ (\texttt{inv:}\ \tilde{e}_i)_{1 \le i \le n}$ and $\tau = \texttt{Boolean}$ and $\Gamma, \texttt{self} : \zeta \vdash_\Omega e_i \triangleright \tilde{e}_i : \texttt{Boolean}$ for all $1 \le i \le n$. The only applicable operational rule for such a $\tilde{t}$ is (Ctxt$^\downarrow$) and, thus, if we have $\gamma \vdash^\omega_\Omega \texttt{context}\ \zeta\ (\texttt{inv:}\ \tilde{e}_i)_{1 \le i \le n} \downarrow v$ then necessarily $\gamma, \texttt{self} \mapsto v' \vdash^\omega_\Omega \tilde{e}_i \downarrow v_{i,v'}$ for all $1 \le i \le n$ and $v' \in \omega(\zeta)$ with $v = \bigwedge_{i,v'} v_{i,v'}$. Since $v_{i,v'} \in \mathit{Value}^\omega_\Omega$ for all i and v' and there is no value v'' with $v'' :_\Omega \texttt{Void}$ and $\texttt{Void}$ is the only sub-type of $\texttt{Boolean}$, we have $v_{i,v'} :_\Omega \texttt{Boolean}$ by the induction hypothesis and, a fortiori, $v :_\Omega \texttt{Boolean} = \tau$.

(Const$^\tau$): Then $\tilde{t} = l$ and $\tau = \mathit{type}_\Omega(l)$. The only applicable operational rule for such a $\tilde{t}$ is (Const$^\downarrow$) and, thus, if we have $\gamma \vdash^\omega_\Omega l \downarrow v$ then $v = l$; but, then, the claim is clear, for $l :_\Omega \mathit{type}_\Omega(l) = \tau$.

(Self$^\tau$): Then $\tilde{t} = \texttt{self}$ and $\tau = \Gamma(\texttt{self})$. The only applicable operational rule for such a $\tilde{t}$ is (Self$^\downarrow$) and, thus, if we have $\gamma \vdash^\omega_\Omega \texttt{self} \downarrow v$, then $v = \gamma(\texttt{self})$; but, then, the claim is clear, for γ conforms to Γ and thus we have $\gamma(\texttt{self}) :_\Omega \Gamma(\texttt{self}) = \tau$.

(Var$^\tau$): Then $\tilde{t} = x$ with $x \in \mathit{VarName}$ and $x \notin T_\Omega$, and $\tau = \Gamma(x)$. The only applicable operational rule for such a $\tilde{t}$ is (Var$^\downarrow$) and, thus, if we have $\gamma \vdash^\omega_\Omega x \downarrow v$ then $v = \gamma(x)$; but, then, the claim is clear, for γ conforms to Γ and thus $\gamma(x) :_\Omega \Gamma(x) = \tau$.

(Type$^\tau$): Then $\tilde{t} = \tau'$ with $\tau' \in T_\Omega$ and $\tau = \texttt{OclType}$. The only applicable operational rule for such a $\tilde{t}$ is (Type$^\downarrow$) and, thus, if we have $\gamma \vdash^\omega_\Omega \tau' \downarrow v$ then $v = \tau'$; but, then, the claim is clear, since $\tau' :_\Omega \texttt{OclType} = \tau$.

(Coll$^\tau_1$): Then $\tilde{t} = \sigma\texttt{\{}\tilde{e}_1\texttt{, ..., }\tilde{e}_n\texttt{\}}$ and $\tau = \sigma\texttt{(}\alpha\texttt{)}$ with $\alpha = \bigsqcup_\Omega \{\mathit{flatten}_\Omega(\tau_i) \mid 1 \le i \le n\}$, and $\Gamma \vdash_\Omega e_i \triangleright \tilde{e}_i : \tau_i$ for $1 \le i \le n$. The only applicable operational rule for such a $\tilde{t}$ is (Coll$^\downarrow_1$) and, thus, if we have $\gamma \vdash^\omega_\Omega \sigma\texttt{\{}\tilde{e}_1\texttt{, ..., }\tilde{e}_n\texttt{\}} \downarrow v$ then necessarily $\gamma \vdash^\omega_\Omega \tilde{e}_i \downarrow v_i$ for $1 \le i \le n$ with $v = \mathit{make}_\Omega(\sigma, \emptyset) = \sigma\texttt{\{\}}$ if $n = 0$, and $v = \mathit{make}_\Omega(\sigma, v_1 \cdots v_n) = \mathit{make}_\Omega(\sigma, v_{11} \cdots v_{1m_1} \cdots v_{n1} \cdots v_{nm_n})$ with $v_{i1} \cdots v_{im_i} = \mathit{flatten}_\Omega(v_i)$ otherwise. If $n = 0$ then $v :_\Omega \sigma\texttt{(Void)}$ and thus $v :_\Omega \sigma\texttt{(}\alpha\texttt{)}$. However, if $n > 0$, by the induction hypothesis, $v_i :_\Omega \tau_i$ and thus $v_{ij} :_\Omega \mathit{flatten}_\Omega(\tau_i)$ for all $1 \le i \le n$ and $1 \le j \le m_i$. Hence we have $v :_\Omega \sigma\texttt{(}\alpha\texttt{)} = \tau$.

(Coll$^\tau_2$): Then $\tilde{t} = \texttt{Sequence\{}\tilde{e}_1\ \texttt{..}\ \tilde{e}_2\texttt{\}}$ and $\tau = \texttt{Sequence(Integer)}$ and $\Gamma \vdash_\Omega e_i \triangleright \tilde{e}_i : \texttt{Integer}$ for $1 \le i \le 2$. The only applicable operational rule for such a $\tilde{t}$ is (Coll$^\downarrow_2$) and, thus, if we have $\gamma \vdash^\omega_\Omega \texttt{Sequence\{}\tilde{e}_1\ \texttt{..}\ \tilde{e}_2\texttt{\}} \downarrow v$ then necessarily $\gamma \vdash^\omega_\Omega \tilde{e}_i \downarrow v_i$ for $1 \le i \le 2$ with $v = \mathit{make}_\Omega(\sigma, v_1\ \texttt{..}\ v_2)$. By the induction hypothesis, we have $v_i :_\Omega \texttt{Integer}$ for $1 \le i \le 2$, and thus, because there is no value v' with $v' :_\Omega \texttt{Void}$ and $\texttt{Void}$ is the only sub-type of $\texttt{Sequence(Integer)}$, we have $v :_\Omega \texttt{Sequence(Integer)} = \tau$.

(Cond$^\tau$): Then $\tilde{t} = \texttt{if}\ \tilde{e}\ \texttt{then}\ \tilde{e}_1\ \texttt{else}\ \tilde{e}_2\ \texttt{endif}$ and $\tau = \bigsqcup_\Omega \{\tau_1, \tau_2\}$ with $\Gamma \vdash e_i \triangleright \tilde{e}_i : \tau_i$ for $1 \le i \le 2$ and $\Gamma \vdash e \triangleright \tilde{e} : \texttt{Boolean}$. The only applicable operational rules for such a $\tilde{t}$ are (Cond$^\downarrow_1$) and (Cond$^\downarrow_2$) and, thus, if we have $\gamma \vdash^\omega_\Omega \texttt{if}\ \tilde{e}\ \texttt{then}\ \tilde{e}_1\ \texttt{else}\ \tilde{e}_2\ \texttt{endif} \downarrow v$ then necessarily $\gamma \vdash^\omega_\Omega \tilde{e}_i \downarrow v_i$ for $1 \le i \le 2$

and $\gamma \vdash^{\omega}_{\Omega} \tilde{e} \downarrow b$ with $b = \texttt{true}$ and $v = v_1$ or $b = \texttt{false}$ and $v = v_2$. By the induction hypothesis, $v_i :_{\Omega} \tau_i \leq_{\Omega} \tau$ for $1 \leq i \leq 2$ and thus, in either case, $v :_{\Omega} \tau$.

(Let$^{\tau}$): Then $\tilde{t} = \texttt{let}\ x : \tau' = \tilde{e}\ \texttt{in}\ \tilde{e}'$ and $\tau = \tau'''$ with $\Gamma \vdash_{\Omega} e \triangleright \tilde{e} : \tau''$ and $\Gamma, x : \tau' \vdash e' \triangleright \tilde{e}' : \tau'''$ and $\tau'' \leq_{\Omega} \tau'$. The only applicable operational rule for such a $\tilde{t}$ is (Let$^{\downarrow}$) and, thus, if we have $\gamma \vdash^{\omega}_{\Omega} \texttt{let}\ x : \tau' = \tilde{e}\ \texttt{in}\ \tilde{e}' \downarrow v$ then necessarily $\gamma \vdash^{\omega}_{\Omega} \tilde{e} \downarrow v'$ and $\gamma, x \mapsto v' \vdash^{\omega}_{\Omega} \tilde{e}' \downarrow v$. By the induction hypothesis, $v' :_{\Omega} \tau''$ and, since hence $\gamma, x \mapsto v'$ conforms to $\Gamma, x : \tau''$, also $v :_{\Omega} \tau''' = \tau$.

(Cast$^{\tau}$): Then $\tilde{t} = e.\texttt{oclAsType(}\tau'\texttt{)}$ and $\tau = \tau'$ with $\Gamma \vdash_{\Omega} e \triangleright \tilde{e} : \tau''$ and $\tau'' \leq_{\Omega} \tau'$ or $\tau' \leq_{\Omega} \tau''$ and $\tau' \neq \texttt{Void}$. The only applicable rules for such a $\tilde{t}$ are (Cast$^{\downarrow}_1$) and (Cast$^{\downarrow}_2$) and, thus, if we have $\gamma \vdash^{\omega}_{\Omega} e.\texttt{oclAsType(}\tau'\texttt{)} \downarrow v$ then necessarily $\gamma \vdash^{\omega}_{\Omega} \tilde{e} \downarrow v$ with $v :_{\Omega} \tau' = \tau$.

(Par$^{\tau}$): Then $\tilde{t} = \tilde{e}_1.o(\tilde{e}_2)$ with $o \in \{\texttt{and}, \texttt{or}\}$ and $\tau = \texttt{Boolean}$ with $\Gamma \vdash_{\Omega} e_1 \triangleright \tilde{e}_i : \texttt{Boolean}$ for $1 \leq i \leq 2$. The only applicable rules for such a $\tilde{t}$ are (Par$^{\downarrow}_1$–Par$^{\downarrow}_6$).

If we have $\gamma \vdash^{\omega}_{\Omega} \tilde{e}_1.o(\tilde{e}_2) \downarrow v$ and (Par$^{\downarrow}_1$) or (Par$^{\downarrow}_4$) has been applied then necessarily $\gamma \vdash^{\omega}_{\Omega} \tilde{e}_i \downarrow v_i$ for $1 \leq i \leq 2$ and $v_1 \wedge v_2 = v$ or $v_1 \vee v_2 = v$. By the induction hypothesis $v_i :_{\Omega} \texttt{Boolean}$ and hence, since there is no value v' with $v' :_{\Omega} \texttt{Void}$ we have $v :_{\Omega} \texttt{Boolean} = \tau$.

If we have $\gamma \vdash^{\omega}_{\Omega} \tilde{e}_1.o(\tilde{e}_2) \downarrow v$ and (Par$^{\downarrow}_2$) or (Par$^{\downarrow}_3$) has been applied then necessarily $o = \texttt{and}$ and $\gamma \vdash^{\omega}_{\Omega} \tilde{e}_1 \downarrow \texttt{false}$ or $\gamma \vdash^{\omega}_{\Omega} \tilde{e}_2 \downarrow \texttt{false}$ and $v = \texttt{false}$. But then $v :_{\Omega} \texttt{Boolean} = \tau$.

If we have $\gamma \vdash^{\omega}_{\Omega} \tilde{e}_1.o(\tilde{e}_2) \downarrow v$ and (Par$^{\downarrow}_5$) or (Par$^{\downarrow}_6$) has been applied then necessarily $o = \texttt{or}$ and $\gamma \vdash^{\omega}_{\Omega} \tilde{e}_1 \downarrow \texttt{true}$ or $\gamma \vdash^{\omega}_{\Omega} \tilde{e}_2 \downarrow \texttt{true}$ and $v = \texttt{true}$. But then $v :_{\Omega} \texttt{Boolean} = \tau$.

(Iter$^{\tau}$): Then $\tilde{t} = \tilde{e}\texttt{->iterate(}x : \alpha\texttt{;}\ x' : \tau' = \tilde{e}'\ \texttt{|}\ \tilde{e}''\texttt{)}$ and $\tau = \tau'$ with $\Gamma \vdash_{\Omega} e \triangleright \tilde{e} : \overline{\sigma}(\alpha')$ and $\Gamma \vdash_{\Omega} e' \triangleright \tilde{e}' : \tau''$ and $\Gamma, x : \alpha, x' : \tau' \vdash_{\Omega} e'' \triangleright \tilde{e}'' : \tau'''$ and $\alpha' \leq_{\Omega} \alpha$ and $\tau'', \tau''' \leq_{\Omega} \tau'$. The only applicable rule for such a $\tilde{t}$ is (Iter$^{\downarrow}$) and, thus, if we have $\gamma \vdash^{\omega}_{\Omega} \tilde{e}\texttt{->iterate(}x : \alpha\texttt{;}\ x' : \tau' = \tilde{e}'\ \texttt{|}\ \tilde{e}''\texttt{)} \downarrow v$ then necessarily $\gamma \vdash^{\omega}_{\Omega} \tilde{e} \downarrow v'$ and $\gamma \vdash^{\omega}_{\Omega} \tilde{e}' \downarrow v''_0$ and $\gamma, x \mapsto v'_i, x' \mapsto v''_{i-1} \vdash^{\omega}_{\Omega} \tilde{e}'' \downarrow v''_i$ for $1 \leq i \leq n$ where $v' \rightsquigarrow \texttt{Sequence\{}v'_1\texttt{,}\ \ldots\texttt{,}\ v'_n\texttt{\}}$ with $v = v''_n$. By the induction hypothesis $v' :_{\Omega} \overline{\sigma}(\alpha')$ and $v''_0 :_{\Omega} \tau'' \leq_{\Omega} \tau'$ and hence $v'_i :_{\Omega} \alpha' \leq_{\Omega} \alpha$ and, inductively, $v''_i :_{\Omega} \tau''' \leq_{\Omega} \tau'$ for $1 \leq i \leq n$, since hence $\gamma, x \mapsto v'_i, x' \mapsto v''_{i-1}$ conforms to $\Gamma, x : \alpha, x' : \tau'$. In particular, $v :_{\Omega} \tau' = \tau$.

(Feat$^{\tau}_1$): Then $\tilde{t} = \tilde{e}.a_{\tau''}$ and $\tau = \tau'''$ with $\Gamma \vdash_{\Omega} e \triangleright \tilde{e} : \tau'$ and $fd_{\Omega}(a, \tau') = \tau''.a : \tau'''$. The only applicable rule for such a $\tilde{t}$ is (Feat$^{\downarrow}_1$) and, thus, if we have $\gamma \vdash^{\omega}_{\Omega} \tilde{e}.a_{\tau''} \downarrow v$ then necessarily $\gamma \vdash^{\omega}_{\Omega} \tilde{e} \downarrow v'$ and $attr^{\omega}_{\Omega}(a_{\tau''}, v') = v$. By the induction hypothesis $v' :_{\Omega} \tau'$ and hence, by condition (1) of Sect. 3.2, we have $v :_{\Omega} \tau''' = \tau$.

(Feat$^{\tau}_2$): Then $\tilde{t} = \tilde{e}.o_{\tau''}\texttt{(}\tilde{e}_1\texttt{,}\ \ldots\texttt{,}\ \tilde{e}_n\texttt{)}$ and $\tau = \tau'''$ with $\Gamma \vdash_{\Omega} e \triangleright \tilde{e} : \tau'$ and $\Gamma \vdash_{\Omega} e_i \triangleright \tilde{e}_i : \tau_i$ and $fd_{\Omega}(o, \tau', (\tau_i)_{1 \leq i \leq n}) = \tau''.o : (\tau'_i)_{1 \leq i \leq n} \rightarrow \tau'''$. The only applicable rule for such a $\tilde{t}$ is (Feat$^{\downarrow}_2$) and, thus, if we have $\gamma \vdash^{\omega}_{\Omega} \tilde{e}.o_{\tau''}\texttt{(}\tilde{e}_1\texttt{,}\ \ldots\texttt{,}\ \tilde{e}_n\texttt{)} \downarrow v$ then necessarily $\gamma \vdash^{\omega}_{\Omega} \tilde{e} \downarrow v'$ and $\gamma \vdash^{\omega}_{\Omega} \tilde{e}_i \downarrow v_i$ for $1 \leq i \leq n$ and $meth^{\omega}_{\Omega}(o_{\tau''}, v', (v_i)_{1 \leq i \leq n}) = v$. By the induction hypothesis $v' :_{\Omega} \tau'$ and $v_i :_{\Omega} \tau_i$ and hence, by condition (2) of Sect. 3.2, we have $v :_{\Omega} \tau''' = \tau$.

(Prop$_1^\tau$): Then $\tilde{t} = \tilde{e}\texttt{->}a_{\tau''}$ and $\tau = \tau'''$ with $\Gamma \vdash_\Omega e \triangleright \tilde{e} : \overline{\sigma}(\alpha)$ and $fd_\Omega(a, \overline{\sigma}(\alpha)) = \tau''.a : \tau'''$. The only applicable rule for such a $\tilde{t}$ is (Prop$_1^\downarrow$) and, thus, if we have $\gamma \vdash^\omega_\Omega \tilde{e}\texttt{->}a_{\tau''} \downarrow v$ then necessarily $\gamma \vdash^\omega_\Omega \tilde{e} \downarrow v'$ and $attr^\omega_\Omega(a_{\tau''}, v') = v$. By the induction hypothesis $v' :_\Omega \overline{\sigma}(\alpha)$ and hence, by condition (1) of Sect. 3.2, we have $v :_\Omega \tau''' = \tau$.

(Prop$_2^\tau$): Then $\tilde{t} = \tilde{e}\texttt{->}o_{\tau''}\texttt{(}\tilde{e}_1\texttt{, ... , }\tilde{e}_n\texttt{)}$ and $\tau = \tau'''$ with $\Gamma \vdash_\Omega e \triangleright \tilde{e} : \overline{\sigma}(\alpha)$ and $\Gamma \vdash_\Omega e_i \triangleright \tilde{e}_i : \tau_i$ and $fd_\Omega(o, \overline{\sigma}(\alpha), (\tau_i)_{1\le i\le n}) = \tau''.o : (\tau'_i)_{1\le i\le n} \to \tau'''$. The only applicable rule for such a $\tilde{t}$ is (Prop$_2^\downarrow$) and, thus, if we have $\gamma \vdash^\omega_\Omega \tilde{e}\texttt{->}o_{\tau''}\texttt{(}\tilde{e}_1\texttt{, ... , }\tilde{e}_n\texttt{)} \downarrow v$ then necessarily $\gamma \vdash^\omega_\Omega \tilde{e} \downarrow v'$ and $\gamma \vdash^\omega_\Omega \tilde{e}_i \downarrow v_i$ for $1 \le i \le n$ and $meth^\omega_\Omega(o_{\tau''}, v', (v_i)_{1\le i\le n}) = v$. By the induction hypothesis $v' :_\Omega \overline{\sigma}(\alpha)$ and $v_i :_\Omega \tau_i$ and hence, by condition (2) of Sect. 3.2, we have $v :_\Omega \tau''' = \tau$.

(Sing$_1^\tau$): Then $\tilde{t} = \texttt{Set\{}\tilde{e}\texttt{\}->iterate(}x : \alpha\texttt{; }x' : \tau' = \tilde{e}' \texttt{ | } \tilde{e}''\texttt{)}$ and $\tau = \tau'$ with $\Gamma \vdash_\Omega e \triangleright \tilde{e} : \alpha'$ and $\Gamma \vdash_\Omega e' \triangleright \tilde{e}' : \tau''$ and $\Gamma, x : \alpha, x' : \tau' \vdash_\Omega e'' \triangleright \tilde{e}'' : \tau'''$ and $\alpha' \le_\Omega \alpha$ and $\tau'', \tau''' \le_\Omega \tau'$. The only applicable rule for such a $\tilde{t}$ is (Iter$^\downarrow$) and, thus, if we have $\gamma \vdash^\omega_\Omega \texttt{Set\{}\tilde{e}\texttt{\}->iterate(}x : \alpha\texttt{; }x' : \tau' = \tilde{e}' \texttt{ | } \tilde{e}''\texttt{)} \downarrow v$, then necessarily $\gamma \vdash^\omega_\Omega \texttt{Set\{}\tilde{e}\texttt{\}} \downarrow v'$ and $\gamma \vdash^\omega_\Omega \tilde{e}' \downarrow v''_0$ and $\gamma, x \mapsto v'_i, x' \mapsto v''_{i-1} \vdash^\omega_\Omega \tilde{e}'' \downarrow v''_i$ for $1 \le i \le n$ where $v' \rightsquigarrow \texttt{Sequence\{}v'_1\texttt{, ... , }v'_n\texttt{\}}$ with $v = v''_n$; moreover, by (Coll$_1^\downarrow$), from $\gamma \vdash^\omega_\Omega \texttt{Set\{}\tilde{e}\texttt{\}} \downarrow v'$ we have $\gamma \vdash^\omega_\Omega \tilde{e} \downarrow v''$. By the induction hypothesis $v'' :_\Omega \alpha'$ and thus $v' :_\Omega \texttt{Set(}\alpha'\texttt{)}$ and $v''_0 :_\Omega \tau'' \le_\Omega \tau'$ and hence $v'_i :_\Omega \alpha' \le_\Omega \alpha$ and, inductively, $v''_i :_\Omega \tau''' \le_\Omega \tau'$ for $1 \le i \le n$, since hence $\gamma, x \mapsto v'_i, x' \mapsto v''_{i-1}$ conforms to $\Gamma, x : \alpha, x' : \tau'$. In particular, $v :_\Omega \tau' = \tau$.

(Sing$_2^\tau$): Then $\tilde{t} = \texttt{Set\{}\tilde{e}\texttt{\}->}o_{\tau''}\texttt{(}\tilde{e}_1\texttt{, ... , }\tilde{e}_n\texttt{)}$ and $\tau = \tau'''$ with $\Gamma \vdash_\Omega e \triangleright \tilde{e} : \alpha$ and $\Gamma \vdash_\Omega e_i \triangleright \tilde{e}_i : \tau_i$ and $fd_\Omega(o, \texttt{Set(}\alpha\texttt{)}, (\tau_i)_{1\le i\le n}) = \tau''.o : (\tau'_i)_{1\le i\le n} \to \tau'''$. The only applicable rule for such a $\tilde{t}$ is (Prop$_2^\downarrow$) and, thus, if we have $\gamma \vdash^\omega_\Omega \texttt{Set\{}\tilde{e}\texttt{\}->}o_{\tau''}\texttt{(}\tilde{e}_1\texttt{, ... , }\tilde{e}_n\texttt{)} \downarrow v$ then necessarily $\gamma \vdash^\omega_\Omega \texttt{Set\{}\tilde{e}\texttt{\}} \downarrow v'$ and $\gamma \vdash^\omega_\Omega \tilde{e}_i \downarrow v_i$ for $1 \le i \le n$ and $meth^\omega_\Omega(o_{\tau''}, v', (v_i)_{1\le i\le n}) = v$; moreover, by (Coll$_1^\downarrow$), from $\gamma \vdash^\omega_\Omega \texttt{Set\{}\tilde{e}\texttt{\}} \downarrow v'$ we have $\gamma \vdash^\omega_\Omega \tilde{e} \downarrow v''$. By the induction hypothesis $v'' :_\Omega \alpha$ and thus $v' :_\Omega \texttt{Set(}\alpha\texttt{)}$ and $v_i :_\Omega \tau_i$ and hence, by condition (2) of Sect. 3.2, we have $v :_\Omega \tau''' = \tau$.

(Short$_1^\tau$): Then $\tilde{t} = \tilde{e}\texttt{->iterate(i : }\alpha\texttt{; a : }\sigma(\alpha')\texttt{ = }\tilde{e}' \texttt{ | } \tilde{e}''\texttt{)}$ and $\tau = \sigma(\alpha')$ with $\tilde{e}' = \sigma\texttt{\{\}}$ and $\tilde{e}'' = \texttt{a->including}_{\sigma(\alpha')}\texttt{(i.}a_{\tau'}\texttt{)}$ and $\Gamma \vdash_\Omega e \triangleright \tilde{e} : \overline{\sigma}(\alpha)$ and $fd_\Omega(a, \alpha) = \tau'.a : \alpha'$ and $\overline{\sigma} \ne \texttt{Sequence}$, and $\sigma = \texttt{Bag}$ or $\overline{\sigma} = \sigma = \texttt{Sequence}$. The only applicable rule for such a $\tilde{t}$ is (Iter$^\downarrow$) and, thus, if we have $\gamma \vdash^\omega_\Omega \tilde{t} \downarrow v$ then necessarily $\gamma \vdash^\omega_\Omega \tilde{e} \downarrow v'$ and $\gamma \vdash^\omega_\Omega \tilde{e}' \downarrow v''_0$ and $\gamma, \texttt{i} \mapsto v'_i, \texttt{a} \mapsto v''_{i-1} \vdash^\omega_\Omega \tilde{e}'' \downarrow v''_i$ for $1 \le i \le n$ where $v' \rightsquigarrow \texttt{Sequence\{}v'_1\texttt{, ... , }v'_n\texttt{\}}$ with $v = v''_n$. By the induction hypothesis $v' :_\Omega \overline{\sigma}(\alpha)$. Moreover, $\sigma\texttt{\{\}} :_\Omega \sigma\texttt{(Void)} \le_\Omega \sigma(\alpha')$ and $\Gamma, \texttt{i} : \alpha, \texttt{a} : \sigma(\alpha') \vdash_\Omega e'' \triangleright \tilde{e}'' : \sigma(\alpha')$ for $e'' = \texttt{a->including(i.}a\texttt{)}$ by the assumptions in Table 2. Thus by the induction hypothesis, inductively, $v''_i :_\Omega \sigma(\alpha')$ for $1 \le i \le n$, since hence $\gamma, \texttt{i} \mapsto v'_i, \texttt{a} \mapsto v''_{i-1}$ conforms to $\Gamma, \texttt{i} : \alpha, \texttt{a} : \sigma(\alpha')$. In particular, $v :_\Omega \sigma(\alpha) = \tau$.

(Short$_2^\tau$): Then $\tilde{t} = \tilde{e}\texttt{->iterate(i : }\alpha\texttt{; a : }\sigma(\alpha')\texttt{ = }\tilde{e}' \texttt{ | } \tilde{e}''\texttt{)}$ and $\tau = \sigma(\alpha')$ with $\tilde{e}' = \sigma\texttt{\{\}}$ and $\tilde{e}'' = \texttt{a->union}_{\sigma(\alpha')}\texttt{(i.}a_{\tau'}\texttt{)}$ and $\Gamma \vdash_\Omega e \triangleright \tilde{e} : \overline{\sigma}(\alpha)$ and $fd_\Omega(a, \alpha) = \tau'.a : \sigma(\alpha')$. The only applicable rule for such a $\tilde{t}$ is (Iter$^\downarrow$) and, thus,

if we have $\gamma \vdash^{\omega}_{\Omega} \tilde{t} \downarrow v$ then necessarily $\gamma \vdash^{\omega}_{\Omega} \tilde{e} \downarrow v'$ and $\gamma \vdash^{\omega}_{\Omega} \tilde{e}' \downarrow v''_0$ and $\gamma, \mathtt{i} \mapsto v'_i, \mathtt{a} \mapsto v''_{i-1} \vdash^{\omega}_{\Omega} \tilde{e}'' \downarrow v''_i$ for $1 \leq i \leq n$ where $v' \rightsquigarrow \mathtt{Sequence\{}v'_1\mathtt{,}\ \ldots\ \mathtt{,}\ v'_n\mathtt{\}}$ with $v = v''_n$. By the induction hypothesis $v' :_{\Omega} \overline{\sigma}(\alpha)$. Moreover, $\sigma\mathtt{\{\}} :_{\Omega} \sigma\mathtt{(Void)} \leq_{\Omega} \sigma(\alpha')$ and $\Gamma, \mathtt{i} : \alpha, \mathtt{a} : \sigma(\alpha') \vdash_{\Omega} e'' \triangleright \tilde{e}'' : \sigma(\alpha')$ for $e'' = \mathtt{a\text{->}union(i}.a\mathtt{)}$ by the assumptions in Table 2. Thus by the induction hypothesis, inductively, $v''_i :_{\Omega} \sigma(\alpha')$ for $1 \leq i \leq n$, since hence $\gamma, \mathtt{i} \mapsto v'_i, \mathtt{a} \mapsto v''_{i-1}$ conforms to $\Gamma, \mathtt{i} : \alpha, \mathtt{a} : \sigma(\alpha')$. In particular, $v :_{\Omega} \sigma(\alpha) = \tau$.

(Weak$^{\tau}$): Then $\Gamma = \Gamma', \Gamma''$ and $\tilde{t} = \tilde{e}$ and $\tau = \tau'$ and $\Gamma'' \vdash_{\Omega} e \triangleright \tilde{e} : \tau'$. If $\gamma \vdash^{\omega}_{\Omega} \tilde{e} \downarrow v$ and γ conforms to Γ then, a fortiori, γ conforms to Γ'' and thus, by the induction hypothesis, $v :_{\Omega} \tau' = \tau$. □

On the other hand, a well-typed OCL term always yields some result, that is, the operational semantics will not get stuck.

Lemma 2. *Let Ω be a UML static structure and ω a system conforming to Ω; let Γ be a type environment and γ a variable environment conforming to Γ; let t be a Term and $\tilde{t}$ an A-Term; let $\tau \in T_{\Omega}$. If $\Gamma \vdash_{\Omega} t \triangleright \tilde{t} : \tau$ then there is a result $\overline{v} \in Result^{\omega}_{\Omega}$ such that $\gamma \vdash^{\omega}_{\Omega} \tilde{t} \downarrow \overline{v}$.*

Proof. By induction on the type derivation and using the strictness convention for the operational rules. □

4 Denotational Semantics

The big-step operational semantics for OCL terms described in the previous section gives rise to a set-based denotational semantics assigning a finite set of results to each OCL term. All operational rules applying to the same OCL term are gathered into a single clause defining the denotation of this term; in particular, each clause has to take into account the potential non-determinism of the operational evaluation process and undefined results.

The denotational semantics for OCL terms is given by a function

$$[\![-]\!]^{\omega}_{\Omega} : A\text{-}Term \rightarrow VarEnv^{\omega}_{\Omega} \rightarrow \wp Result^{\omega}_{\Omega}$$

where Ω is a UML static structure and ω a system conforming to Ω.

Besides the semantic functions in Sect. 3.1, we use some additional functions in order to define $[\![-]\!]^{\omega}_{\Omega}$: The families of functions

$$lift_{f^{(n)}}, \overline{lift}_{g^{(n)}} : (\wp Result^{\omega}_{\Omega})^n \rightarrow \wp Result^{\omega}_{\Omega}$$

defined for all n-ary functions $f : {Value^{\omega}_{\Omega}}^n \rightarrow \wp Result^{\omega}_{\Omega}$ and $g : {Result^{\omega}_{\Omega}}^n \rightarrow \wp Result^{\omega}_{\Omega}$ by

$$\begin{aligned} lift_{f^{(n)}}(\overline{V}_1, \ldots, \overline{V}_n) = (&\textstyle\bigcup_{v_1 \in \overline{V}_1 \cap Value^{\omega}_{\Omega}, \ldots, v_n \in \overline{V}_n \cap Value^{\omega}_{\Omega}} f(v_1, \ldots, v_n)) \cup \\ &((\overline{V}_1 \cup \cdots \cup \overline{V}_n) \cap \{\bot\}) \ , \\ \overline{lift}_{g^{(n)}}(\overline{V}_1, \ldots, \overline{V}_n) = &\textstyle\bigcup_{\overline{v}_1 \in \overline{V}_1, \ldots, v_n \in \overline{V}_n} g(v_1, \ldots, v_n) \ , \end{aligned}$$

$$
\begin{array}{rl}
 & [\![\texttt{context}\ \zeta\ (\texttt{inv:}\ \tilde{e}_i)_{1\le i\le n}]\!]^{\omega}_{\Omega}\,\gamma = \\
(\mathrm{Ctxt}^{\in}) & \quad \bigcup_{v\in\omega(\zeta)}((\bigcup v_1\cdots v_n\,.\,\{\bigwedge_{1\le i\le n} v_i\}) \\
 & \qquad\qquad ([\![\tilde{e}_1]\!]^{\omega}_{\Omega}\,(\gamma,\texttt{self}\mapsto v))\cdots([\![\tilde{e}_n]\!]^{\omega}_{\Omega}\,(\gamma,\texttt{self}\mapsto v)))
\end{array}
$$

$$(\mathrm{Const}^{\in})\quad [\![l]\!]^{\omega}_{\Omega}\,\gamma = \{l\}$$

$$(\mathrm{Self}^{\in})\quad [\![\texttt{self}]\!]^{\omega}_{\Omega}\,\gamma = \{\overline{\gamma}(\texttt{self})\}$$

$$(\mathrm{Var}^{\in})\quad [\![x]\!]^{\omega}_{\Omega}\,\gamma = \{\overline{\gamma}(x)\}$$

$$(\mathrm{Type}^{\in})\quad [\![\tau]\!]^{\omega}_{\Omega}\,\gamma = \{\tau\}$$

$$
\begin{array}{rl}
(\mathrm{Coll}_1^{\in}) & [\![\sigma\texttt{\{}\tilde{e}_1\texttt{, }\ldots\texttt{, }\tilde{e}_n\texttt{\}}]\!]^{\omega}_{\Omega}\,\gamma = \\
 & \quad (\bigcup v_1\cdots v_n\,.\,\{make_{\Omega}(\sigma, v_1\cdots v_n)\})\,([\![\tilde{e}_1]\!]^{\omega}_{\Omega}\,\gamma)\cdots([\![\tilde{e}_n]\!]^{\omega}_{\Omega}\,\gamma)
\end{array}
$$

$$
\begin{array}{rl}
(\mathrm{Coll}_2^{\in}) & [\![\texttt{Sequence\{}\tilde{e}_1\ \texttt{..}\ \tilde{e}_2\texttt{\}}]\!]^{\omega}_{\Omega}\,\gamma = \\
 & \quad (\bigcup v_1\,v_2\,.\,\{make_{\Omega}(\texttt{Sequence}, v_1\,..\,v_2)\})\,([\![\tilde{e}_1]\!]^{\omega}_{\Omega}\,\gamma)\,([\![\tilde{e}_2]\!]^{\omega}_{\Omega}\,\gamma)
\end{array}
$$

$$
\begin{array}{rl}
 & [\![\texttt{if}\ \tilde{e}\ \texttt{then}\ \tilde{e}_1\ \texttt{else}\ \tilde{e}_2\ \texttt{endif})]\!]^{\omega}_{\Omega}\,\gamma = \\
(\mathrm{Cond}^{\in}) & \quad (\bigcup v\,v_1\,v_2\,.\,\{v' \mid (v=\texttt{true}\wedge v'=v_1)\vee(v=\texttt{false}\wedge v'=v_2)\}) \\
 & \qquad ([\![\tilde{e}]\!]^{\omega}_{\Omega}\,\gamma)\,([\![\tilde{e}_1]\!]^{\omega}_{\Omega}\,\gamma)\,([\![\tilde{e}_2]\!]^{\omega}_{\Omega}\,\gamma)
\end{array}
$$

$$(\mathrm{Let}^{\in})\quad [\![\texttt{let}\ x\ \texttt{:}\ \tau\ \texttt{=}\ \tilde{e}\ \texttt{in}\ \tilde{e}')]\!]^{\omega}_{\Omega}\,\gamma = (\bigcup v\,.\,[\![\tilde{e}']\!]^{\omega}_{\Omega}\,(\gamma, x\mapsto v))\,([\![\tilde{e}]\!]^{\omega}_{\Omega}\,\gamma)$$

$$
\begin{array}{rl}
(\mathrm{Cast}^{\in}) & [\![\tilde{e}\texttt{.oclAsType(}\tau\texttt{)}]\!]^{\omega}_{\Omega}\,\gamma = \\
 & \quad (\bigcup v\,.\,\{\overline{v}' \mid (v :_{\Omega}\tau\wedge\overline{v}'=v)\vee(v\not:_{\Omega}\tau\wedge\overline{v}'=\bot)\})\,([\![\tilde{e}]\!]^{\omega}_{\Omega}\,\gamma)
\end{array}
$$

$$(\mathrm{Par}_1^{\in})\quad [\![\tilde{e}_1\texttt{.and(}\tilde{e}_2\texttt{)}]\!]^{\omega}_{\Omega}\,\gamma = (\overline{\bigcup}\,\overline{v}_1\,\overline{v}_2\,.\,\{\overline{v}_1\,\overline{\wedge}\,\overline{v}_2\})\,([\![\tilde{e}_1]\!]^{\omega}_{\Omega}\,\gamma)\,([\![\tilde{e}_2]\!]^{\omega}_{\Omega}\,\gamma)$$

$$(\mathrm{Par}_2^{\in})\quad [\![\tilde{e}_1\texttt{.or(}\tilde{e}_2\texttt{)}]\!]^{\omega}_{\Omega}\,\gamma = (\overline{\bigcup}\,\overline{v}_1\,\overline{v}_2\,.\,\{\overline{v}_1\,\overline{\vee}\,\overline{v}_2\})\,([\![\tilde{e}_1]\!]^{\omega}_{\Omega}\,\gamma)\,([\![\tilde{e}_2]\!]^{\omega}_{\Omega}\,\gamma)$$

$$
\begin{array}{rl}
 & [\![\tilde{e}\texttt{->iterate(}x\ \texttt{:}\ \alpha\texttt{;}\ x'\ \texttt{:}\ \tau\ \texttt{=}\ \tilde{e}'\ \texttt{|}\ \tilde{e}''\texttt{)}]\!]^{\omega}_{\Omega}\,\gamma = \\
(\mathrm{Iter}^{\in}) & \quad (\bigcup v\,v_0'\,.\,\bigcup_{v\rightsquigarrow\texttt{Sequence\{}v_1\texttt{, }\ldots\texttt{, }v_n\texttt{\}}} iterate(x, v_1\cdots v_n, x', v_0')\,[\![\tilde{e}'']\!]^{\omega}_{\Omega}\,\gamma) \\
 & \qquad ([\![\tilde{e}]\!]^{\omega}_{\Omega}\,\gamma)\,([\![\tilde{e}']\!]^{\omega}_{\Omega}\,\gamma)
\end{array}
$$

$$(\mathrm{Feat}_1^{\in})\quad [\![\tilde{e}\,.\,a_\tau]\!]^{\omega}_{\Omega}\,\gamma = (\bigcup v\,.\,\{attr^{\omega}_{\Omega}(a_\tau, v)\})\,([\![\tilde{e}]\!]^{\omega}_{\Omega}\,\gamma)$$

$$
\begin{array}{rl}
 & [\![\tilde{e}\,.\,o_\tau\texttt{(}\tilde{e}_1\texttt{, }\ldots\texttt{, }\tilde{e}_n\texttt{)}]\!]^{\omega}_{\Omega}\,\gamma = \\
(\mathrm{Feat}_2^{\in}) & \quad (\bigcup v\,v_1\cdots v_n\,.\,\{meth^{\omega}_{\Omega}(o_\tau, v, (v_i)_{1\le i\le n})\}) \\
 & \qquad ([\![\tilde{e}]\!]^{\omega}_{\Omega}\,\gamma)\,([\![\tilde{e}_1]\!]^{\omega}_{\Omega}\,\gamma)\cdots([\![\tilde{e}_n]\!]^{\omega}_{\Omega}\,\gamma)
\end{array}
$$

$$(\mathrm{Prop}_1^{\in})\quad [\![\tilde{e}\texttt{->}a_\tau]\!]^{\omega}_{\Omega}\,\gamma = (\bigcup v\,.\,\{attr^{\omega}_{\Omega}(a_\tau, v)\})\,([\![\tilde{e}]\!]^{\omega}_{\Omega}\,\gamma)$$

$$
\begin{array}{rl}
 & [\![\tilde{e}\texttt{->}o_\tau\texttt{(}\tilde{e}_1\texttt{, }\ldots\texttt{, }\tilde{e}_n\texttt{)}]\!]^{\omega}_{\Omega}\,\gamma = \\
(\mathrm{Prop}_2^{\in}) & \quad (\bigcup v\,v_1\cdots v_n\,.\,\{meth^{\omega}_{\Omega}(o_\tau, v, (v_i)_{1\le i\le n})\}) \\
 & \qquad ([\![\tilde{e}]\!]^{\omega}_{\Omega}\,\gamma)\,([\![\tilde{e}_1]\!]^{\omega}_{\Omega}\,\gamma)\cdots([\![\tilde{e}_n]\!]^{\omega}_{\Omega}\,\gamma)
\end{array}
$$

Table 8. Denotational semantics

respectively, lift functions on the semantic domains $Value_\Omega^{\omega\,n}$ and $Result_\Omega^{\omega\,n}$ to functions on $\wp Result_\Omega^\omega$, propagating undefined results. We write $(\bigcup v_1 \cdots v_n \,.\, f)\ \overline{V}_1 \cdots \overline{V}_n$ instead of $lift_{f^{(n)}}(\overline{V}_1, \ldots, \overline{V}_n)$ and also $(\overline{\bigcup} v_1 \cdots v_n \,.\, g)\,(\overline{V}_1 \cdots \overline{V}_n)$ instead of $\overline{lift}_{g^{(n)}}(\overline{V}_1, \ldots, \overline{V}_n)$.

Furthermore, the family of functions

$$iterate_f : VarName \times Value_\Omega^{\omega\,*} \times VarName \times Value_\Omega^\omega \times VarEnv_\Omega^\omega \to \wp Result_\Omega^\omega$$

defined for all functions $f : VarEnv_\Omega^\omega \to \wp Result_\Omega^\omega$ by

$$iterate_f(x, \emptyset, x', v, \gamma) = \{v\}$$
$$iterate_f(x, v_1 \cdots v_n, x', v, \gamma) = \bigcup_{v' \in f(\gamma, x \mapsto v_1, x' \mapsto v)} iterate_f(x, v_2 \cdots v_n, x', v', \gamma)$$

iterates a function on variable environments over a sequence of values and accumulates previous results. We write $iterate(x, v_1 \cdots v_n, x', v)\, f\, \gamma$ instead of $iterate_f(x, v_1 \cdots v_n, x', v, \gamma)$.

Moreover, the functions

$$\overline{\wedge}, \overline{\vee} : (BooleanLiteral \cup \{\bot\})^2 \to BooleanLiteral \cup \{\bot\}$$

defined by

$$\overline{b}_1 \,\overline{\wedge}\, \overline{b}_2 = \begin{cases} \overline{b}_1 \wedge \overline{b}_2, & \text{if } \overline{b}_1, \overline{b}_2 \in BooleanLiteral \\ \texttt{false}, & \text{if } \overline{b}_1 = \texttt{false} \text{ or } \overline{b}_2 = \texttt{false} \\ \bot, & \text{otherwise} \end{cases}$$

$$\overline{b}_1 \,\overline{\vee}\, \overline{b}_2 = \begin{cases} \overline{b}_1 \vee \overline{b}_2, & \text{if } \overline{b}_1, \overline{b}_2 \in BooleanLiteral \\ \texttt{true}, & \text{if } \overline{b}_1 = \texttt{true} \text{ or } \overline{b}_2 = \texttt{true} \\ \bot, & \text{otherwise} \end{cases}$$

represent "parallel" boolean connectives.

Finally, for a variable environment $\gamma \in VarEnv_\Omega^\omega$ we define

$$\overline{\gamma}(x) = \begin{cases} \gamma(x), & \text{if } x \in \mathrm{dom}(\gamma) \\ \bot, & \text{if } x \notin \mathrm{dom}(\gamma) \end{cases}$$

The function $[\![-]\!]_\Omega^\omega$ is defined in Table 8. The meta-variables range as follows: $l \in Literal$; $\alpha \in A_\Omega$, $\zeta \in C_\Omega$, $\sigma \in S$, $\tau \in T_\Omega$; $x \in VarName$; $a, o \in Name$; $v \in Value_\Omega^\omega$, $\overline{v} \in Result_\Omega^\omega$; $\gamma \in VarEnv_\Omega^\omega$; $\tilde{e} \in A\text{-}Expression$. —

By its very construction, the denotational and the operational semantics coincide on well-typed OCL terms:

Proposition 2. *Let Ω be a UML static structure and ω a system conforming to Ω; let Γ be a type environment and γ a variable environment conforming to Γ; let t be a Term and $\tilde{t}$ an A-Term; let $\tau \in T_\Omega$ and $\overline{v} \in Result_\Omega^\omega$. If $\Gamma \vdash_\Omega t \triangleright \tilde{t} : \tau$, then $\gamma \vdash_\Omega^\omega \tilde{t} \downarrow \overline{v}$ if, and only if $\overline{v} \in [\![\tilde{t}]\!]_\Omega^\omega\, \gamma$.*

Proof. We proceed by induction over the term structure of $\tilde{t}$.

If $\tilde{t}$ is of the form $l \in \mathit{Literal}$ or `self` or $x \in \mathit{VarName}$ or $\tau \in T_\Omega$, the claim is obvious.

Let $\tilde{t} = \sigma\{\tilde{e}_1, \ldots, \tilde{e}_n\}$. The only applicable operational rule for such a $\tilde{t}$ is $(\mathrm{Coll}_1^\downarrow)$, the only applicable denotational clause $(\mathrm{Coll}_1^\in)$. By $(\mathrm{Coll}_1^\downarrow)$, $\gamma \vdash^\omega_\Omega \tilde{t} \downarrow v$ with $v \in \mathit{Value}^\omega_\Omega$, if, and only if $\gamma \vdash \tilde{e}_i \downarrow v_i$ with $v_i \in \mathit{Value}^\omega_\Omega$ for all $1 \leq i \leq n$, if, and only if $v_i \in [\![\tilde{e}_i]\!]^\omega_\Omega \gamma$ for all $1 \leq i \leq n$ by the induction hypothesis, and hence if, and only if $v \in [\![\tilde{t}]\!]^\omega_\Omega \gamma$ by the definition of $(\mathrm{Coll}\in_1)$. Conversely, $\gamma \vdash^\omega_\Omega \tilde{t} \downarrow \bot$ if, and only if $\gamma \vdash^\omega_\Omega \tilde{e}_i \downarrow \bot$ for some $1 \leq i \leq n$, if, and only if $\bot \in [\![\tilde{e}_i]\!]^\omega_\Omega \gamma$, if, and only if $\bot \in [\![\tilde{t}]\!]^\omega_\Omega \gamma$ by the definition of $\bigcup$.

If $\tilde{t} = \texttt{Sequence\{}\tilde{e}_1 \texttt{ .. } \tilde{e}_2\texttt{\}}$ the same reasoning applies as for the case $\tilde{t} = \sigma\{\tilde{e}_1, \ldots, \tilde{e}_n\}$.

Let $\tilde{t} = \texttt{if } \tilde{e} \texttt{ then } \tilde{e}_1 \texttt{ else } \tilde{e}_2 \texttt{ endif}$. The only applicable operational rules for such a $\tilde{t}$ are $(\mathrm{Cond}_1^\downarrow)$ and $(\mathrm{Cond}_2^\downarrow)$, the only applicable denotational clause $(\mathrm{Cond}^\in)$. By $(\mathrm{Cond}_1^\downarrow)$ and $(\mathrm{Cond}_2^\downarrow)$, $\gamma \vdash^\omega_\Omega \tilde{t} \downarrow v'$ with $v' \in \mathit{Value}^\omega_\Omega$, if, and only if $\gamma \vdash \tilde{e} \downarrow v$ and $\gamma \vdash \tilde{e}_i \downarrow v_i$ with $v \in \mathit{BooleanLiteral}$ and $v_i \in \mathit{Value}^\omega_\Omega$ for $1 \leq i \leq 2$, if, and only if $v \in [\![\tilde{e}]\!]^\omega_\Omega \gamma$ and $v_i \in [\![\tilde{e}_i]\!]^\omega_\Omega \gamma$ for $1 \leq i \leq 2$ by the induction hypothesis, and hence if, and only if $v \in [\![\tilde{t}]\!]^\omega_\Omega \gamma$ by the definition of $(\mathrm{Cond}^\in)$. Conversely, $\gamma \vdash^\omega_\Omega \tilde{t} \downarrow \bot$ if, and only if $\gamma \vdash^\omega_\Omega \tilde{e} \downarrow \bot$ or $\gamma \vdash^\omega_\Omega \tilde{e}_i \downarrow \bot$ for some $1 \leq i \leq 2$, if, and only if $\bot \in [\![\tilde{e}]\!]^\omega_\Omega \gamma$ or $\bot \in [\![\tilde{e}_i]\!]^\omega_\Omega \gamma$, if, and only if $\bot \in [\![\tilde{t}]\!]^\omega_\Omega \gamma$ by the definition of $\bigcup$.

Let $\tilde{t} = \texttt{let } x \texttt{ : } \tau \texttt{ = } \tilde{e} \texttt{ in } \tilde{e}'$. The only applicable operational rules for such a $\tilde{t}$ is $(\mathrm{Let}^\downarrow)$, the only applicable denotational clause $(\mathrm{Let}^\in)$. By $(\mathrm{Let}^\downarrow)$, $\gamma \vdash^\omega_\Omega \tilde{t} \downarrow v''$ with $v'' \in \mathit{Value}^\omega_\Omega$, if, and only if $\gamma \vdash^\omega_\Omega \tilde{e} \downarrow v$ and $\gamma, x \mapsto v \vdash^\omega_\Omega \tilde{e}' \downarrow v'$ with $v, v' \in \mathit{Value}^\omega_\Omega$. By the induction hypothesis, $\gamma \vdash \tilde{e} \downarrow v$ if, and only if $v \in [\![\tilde{e}]\!]^\omega_\Omega \gamma$. Since $\Gamma \vdash_\Omega t \triangleright \tilde{t} : \tau'$ the variable environment $\gamma, x \mapsto v$ conforms to $\Gamma, x : \tau$, and thus we have $\gamma, x \mapsto v \vdash^\omega_\Omega \tilde{e}' \downarrow v'$ if, and only if $v' \in [\![\tilde{t}]\!]^\omega_\Omega (\gamma, x \mapsto v)$ by the induction hypothesis, and hence, $\gamma \vdash^\omega_\Omega \tilde{t} \downarrow v''$ if, and only if $v'' \in [\![\tilde{t}]\!]^\omega_\Omega \gamma$ by the definition of $(\mathrm{Let}^\in)$. Conversely, $\gamma \vdash^\omega_\Omega \tilde{t} \downarrow \bot$ if, and only if $\gamma \vdash^\omega_\Omega \tilde{e} \downarrow \bot$ or $\gamma \vdash^\omega_\Omega \tilde{e} \downarrow v$ and $\gamma, x \mapsto v \vdash^\omega_\Omega \tilde{e}' \downarrow \bot$ for some $v \in \mathit{Value}^\omega_\Omega$, if, and only if $\bot \in [\![\tilde{t}]\!]^\omega_\Omega \gamma$ or $\bot \in [\![\tilde{e}']\!]^\omega_\Omega (\gamma, x \mapsto v)$, if, and only if $\bot \in [\![\tilde{t}]\!]^\omega_\Omega \gamma$ by the definition of $\bigcup$.

If $\tilde{t} = \tilde{e}\texttt{.oclAsType(}\tau\texttt{)}$ the same reasoning applies as for the case $\tilde{t} = \texttt{if } \tilde{e} \texttt{ then } \tilde{e}_1 \texttt{ else } \tilde{e}_2 \texttt{ endif}$.

Let $\tilde{t} = \tilde{e}_1\texttt{.and(}\tilde{e}_2\texttt{)}$. The only applicable operational rules for such a $\tilde{t}$ are $(\mathrm{Par}_1^\downarrow$–$\mathrm{Par}_3^\downarrow)$ and $(\mathrm{Par}_7^\downarrow)$, the only applicable denotational clause $(\mathrm{Par}_1^\in)$. By $(\mathrm{Par}_1^\downarrow$–$\mathrm{Par}_3^\downarrow)$, $\gamma \vdash^\omega_\Omega \tilde{t} \downarrow v$ with $\mathit{Value}^\omega_\Omega$ if, and only if $\gamma \vdash^\omega_\Omega \tilde{e}_i \downarrow v_i$ with $v_1, v_2 \in \mathit{Value}^\omega_\Omega$ or $\gamma \vdash^\omega_\Omega \tilde{e}_1 \downarrow \texttt{false}$ or $\gamma \vdash^\omega_\Omega \tilde{e}_2 \downarrow \texttt{false}$. But, for $1 \leq i \leq 2$, $\gamma \vdash^\omega_\Omega \tilde{e}_i \downarrow \overline{v}_i$ with $\overline{v}_i \in \mathit{Result}^\omega_\Omega$ if, and only if $\overline{v}_i \in [\![\tilde{e}_i]\!]^\omega_\Omega \gamma$, by the induction hypothesis. Thus, $\gamma \vdash^\omega_\Omega \tilde{t} \downarrow v$ with $v \in \mathit{Value}^\omega_\Omega$ if, and only if $v \in [\![\tilde{t}]\!]^\omega_\Omega \gamma$ by the definition of $(\mathrm{Par}_1^\in)$ and Lemma 2. Conversely, $\gamma \vdash^\omega_\Omega \tilde{t} \downarrow \bot$ if, and only if $\gamma \vdash^\omega_\Omega \tilde{e}_i \downarrow \overline{v}_i$ with $\overline{v}_1, \overline{v}_2 \in \mathit{Value}^\omega_\Omega$ and $\overline{v}_1 \neq \texttt{false}$ and $\overline{v}_2 = \bot$ or $\overline{v}_1 = \bot$ and $\overline{v}_2 \neq \texttt{false}$, if, and only if $\bot \in [\![\tilde{t}]\!]^\omega_\Omega \gamma$ by the definition of $\overline{\wedge}$ and $\overline{\bigcup}$.

If $\tilde{t} = \tilde{e}_1\texttt{.or(}\tilde{e}_2\texttt{)}$ the same reasoning applies as for the case $\tilde{t} = \tilde{e}_1\texttt{.and(}\tilde{e}_2\texttt{)}$, by the definition of $\overline{\nabla}$.

Let $\tilde{t} = \tilde{e}\texttt{->iterate(}x : \alpha\texttt{; } x' : \tau' \texttt{ = } \tilde{e}' \texttt{ | } \tilde{e}''\texttt{)}$. The only applicable operational rules for such a $\tilde{t}$ is (Iter$^{\downarrow}$), the only applicable denotational clause (Iter$^{\in}$). By (Iter$^{\downarrow}$), $\gamma \vdash^{\omega}_{\Omega} \tilde{e}\texttt{->iterate(}x : \alpha\texttt{; } x' : \tau' \texttt{ = } \tilde{e}' \texttt{ | } \tilde{e}''\texttt{)} \downarrow v$ with $v \in \mathit{Value}^{\omega}_{\Omega}$ if, and only if $\gamma \vdash^{\omega}_{\Omega} \tilde{e} \downarrow v'$ with $v' \in \mathit{Value}^{\omega}_{\Omega}$ and $\gamma \vdash^{\omega}_{\Omega} \tilde{e}' \downarrow v''_0$ and $\gamma, x \mapsto v'_i, x' \mapsto v''_{i-1} \vdash^{\omega}_{\Omega} \tilde{e}'' \downarrow v''_i$ with $v''_i \in \mathit{Value}^{\omega}_{\Omega}$ for $0 \leq i \leq n$ where $v' \rightsquigarrow \texttt{Sequence\{}v'_1\texttt{, } \ldots\texttt{, } v'_n\texttt{\}}$. By the induction hypothesis, $\omega; \gamma \vdash^{\omega}_{\Omega} \tilde{e} \downarrow v'$ if, and only if $v' \in [\![\tilde{e}]\!]^{\omega}_{\Omega}\,\gamma$ and $\omega; \gamma \vdash^{\omega}_{\Omega} \tilde{e}' \downarrow v''_0$ if, and only if $v''_0 \in [\![\tilde{e}']\!]^{\omega}_{\Omega}\,\gamma$. Moreover, since $\Gamma \vdash_{\Omega} t \triangleright \tilde{t} : \tau$, inductively, all variable environments $\gamma, x \mapsto v'_i, x' \mapsto v''_{i-1}$ conform to $\gamma, x : \alpha, x' : \tau$, and thus we have that $\gamma, x \mapsto v'_i, x' \mapsto v''_{i-1} \vdash^{\omega}_{\Omega} \tilde{e}'' \downarrow v''_i$ if, and only if $v''_i \in [\![\tilde{e}'']\!]^{\omega}_{\Omega}\,(\gamma, x \mapsto v'_i, x' \mapsto v''_{i-1}$. Thus, $\gamma \vdash^{\omega}_{\Omega} \tilde{t} \downarrow v$ if, and only if $v \in [\![\tilde{t}]\!]^{\omega}_{\Omega}\,\gamma$ by the definition of (Iter$^{\in}$). Conversely, $\gamma \vdash^{\omega}_{\Omega} \tilde{t} \downarrow \bot$ if, and only if $\gamma \vdash^{\omega}_{\Omega} \tilde{e} \downarrow \bot$ or $\gamma \vdash^{\omega}_{\Omega} \tilde{e}' \downarrow \bot$ or $\gamma \vdash^{\omega}_{\Omega} \tilde{e} \downarrow v'$ and $\gamma \vdash^{\omega}_{\Omega} \tilde{e}' \downarrow v''_0$ with $v', v''_0 \in \mathit{Value}^{\omega}_{\Omega}$ and $\gamma, x \mapsto v'_i, x' \mapsto v''_{i-1} \vdash^{\omega}_{\Omega} \tilde{e}'' \downarrow v''_i$ for $1 \leq i \leq k$ and $\gamma, x \mapsto v'_k, x' \mapsto v''_{k-1} \vdash^{\omega}_{\Omega} \tilde{e}'' \downarrow \bot$ and $1 \leq k \leq n$ where $v' \rightsquigarrow \texttt{Sequence\{}v'_1\texttt{, } \ldots\texttt{, } v'_n\texttt{\}}$. But hence $\gamma \vdash^{\omega}_{\Omega} \tilde{t} \downarrow \bot$ if, and only if $\bot \in [\![\tilde{t}]\!]^{\omega}_{\Omega}\,\gamma$ by the induction hypothesis and the definition of *iterate*.

If $\tilde{t}$ is of the form $\tilde{e}\texttt{.}a_\tau$ or $\tilde{e}\texttt{.}o_\tau\texttt{(}\tilde{e}_1\texttt{, } \ldots\texttt{, } \tilde{e}_n\texttt{)}$ or $\tilde{e}\texttt{->}a_\tau$ or $\tilde{e}\texttt{->}o_\tau\texttt{(}\tilde{e}_1\texttt{, } \ldots\texttt{, } \tilde{e}_n\texttt{)}$, the same reasoning applies as for the case $\tilde{t} = \sigma\texttt{\{}\tilde{e}_1\texttt{, } \ldots\texttt{, } \tilde{e}_n\texttt{\}}$.

Let $\tilde{t} = \texttt{context } \zeta \texttt{ (inv: } \tilde{e}_i)_{1 \leq i \leq n}$. The only applicable operational rule for such a $\tilde{t}$ is (Ctxt$^{\downarrow}$), the only applicable denotational clause (Ctxt$^{\in}$). By (Ctxt$^{\downarrow}$), $\gamma \vdash^{\omega}_{\Omega} \tilde{t} \downarrow v'$ with $v' \in \mathit{Value}^{\omega}_{\Omega}$ if, and only if $\gamma, \texttt{self} \mapsto v \vdash^{\omega}_{\Omega} \tilde{e}_i \downarrow v_{i,v}$ with $v_{i,v} \in \mathit{Value}^{\omega}_{\Omega}$ for all $v \in \omega(\zeta)$ and $1 \leq i \leq n$. Since $\Gamma \vdash t \triangleright \tilde{t} : \tau$, the variable environment $\gamma, \texttt{self} \mapsto v$ conforms to $\Gamma, \texttt{self} : \zeta$ for all $v \in \omega(\zeta)$, and thus we have, by the induction hypothesis, that $\gamma, \texttt{self} \mapsto v \vdash^{\omega}_{\Omega} \tilde{e}_i \downarrow v_{i,v}$ if, and only if $v_{v,i} \in [\![\tilde{e}_i]\!]^{\omega}_{\Omega}\,(\gamma, \texttt{self} \mapsto v)$ for all $1 \leq i \leq n$, and hence if, and only if $v' \in [\![\tilde{t}]\!]^{\omega}_{\Omega}\,\gamma$ by the definition of (Ctxt$^{\in}$). Conversely, $\gamma \vdash^{\omega}_{\Omega} \tilde{t} \downarrow \bot$ if, and only if $\gamma, \texttt{self} \mapsto v \vdash \tilde{e}_i \downarrow \bot$ for some $v \in \omega(\zeta)$ and some $1 \leq i \leq n$, if, and only if $\bot \in [\![\tilde{e}_i]\!]^{\omega}_{\Omega}\,(\gamma, \texttt{self} \mapsto v)$, if, and only if $\bot \in [\![\tilde{t}]\!]^{\omega}_{\Omega}\,\gamma$ by the definition of $\bigcup$. □

5 Expressiveness

Mandel and Cengarle [6] have argued that all primitive recursive functions can be encoded as OCL expressions. We prove the reverse direction of this observation: that the evaluation of a well-typed OCL term over a UML static structure and a system conforming to this static structure and only showing primitive recursive methods and attributes is primitive recursive. In particular, the evaluation of a well-typed OCL term over the empty UML static structure is primitive recursive.

Since every primitive recursive function can be represented by an OCL expression, a function evaluating any OCL expression cannot be primitive recursive. However, for every OCL term, the defining clauses for the denotational semantics in Sect. 4 allow to derive a function that evaluates this term and is

primitive recursive. For example, the OCL expression

```
Sequence{ 1, 2 }->iterate(i : Integer; a : Integer = 0 | a.+(i))
```

yields the function

$$
\begin{aligned}
&[\![\texttt{Sequence\{ 1, 2 \}->iterate(i : Integer;}\\
&\qquad\qquad \texttt{a : Integer = 0 | a.+}_{\texttt{Integer}}\texttt{(i))}]\!]^{\omega}_{\Omega}\,\gamma =\\
&(\bigcup v\, v'_0 \,.\, \bigcup_{v \rightsquigarrow \texttt{Sequence}\{v_1\,,\,\ldots\,,\,v_n\}} \mathit{iterate}(\texttt{i}, v_1 \cdots v_n, \texttt{a}, v'_0)\\
&\qquad [\![\texttt{a.+}_{\texttt{Integer}}\texttt{(i)}]\!]^{\omega}_{\Omega}\,\gamma)\,([\![\texttt{Sequence\{ 1, 2 \}}]\!]^{\omega}_{\Omega}\,\gamma)\,([\![\texttt{0}]\!]^{\omega}_{\Omega}\,\gamma) =\\
&(\bigcup v\, v'_0 \,.\, \bigcup_{v \rightsquigarrow \texttt{Sequence}\{v_1\,,\,\ldots\,,\,v_n\}} \mathit{iterate}(\texttt{i}, v_1 \cdots v_n, \texttt{a}, v'_0)\\
&\qquad ((\bigcup v\, v_1 \,.\, \{\mathit{meth}^{\omega}_{\Omega}(\texttt{+}_{\texttt{Integer}}, v, v_1)\})\,([\![\texttt{a}]\!]^{\omega}_{\Omega}\,\gamma)\,([\![\texttt{i}]\!]^{\omega}_{\Omega}\,\gamma))\,\gamma)\\
&\quad ((\bigcup v_1\, v_2 \,.\, \{\mathit{make}(\texttt{Sequence}, v_1\, v_2)\})\,([\![\texttt{1}]\!]^{\omega}_{\Omega}\,\gamma)\,([\![\texttt{2}]\!]^{\omega}_{\Omega}\,\gamma))\,([\![\texttt{0}]\!]^{\omega}_{\Omega}\,\gamma) =\\
&(\bigcup v\, v'_0 \,.\, \bigcup_{v \rightsquigarrow \texttt{Sequence}\{v_1\,,\,\ldots\,,\,v_n\}} \mathit{iterate}(\texttt{i}, v_1 \cdots v_n, \texttt{a}, v'_0)\\
&\qquad ((\bigcup v\, v_1 \,.\, \{\mathit{meth}^{\omega}_{\Omega}(\texttt{+}_{\texttt{Integer}}, v, v_1)\})\,\{\overline{\gamma}(\texttt{a})\}\,\{\overline{\gamma}(\texttt{i})\})\,\gamma)\\
&\quad ((\bigcup v_1\, v_2 \,.\, \{\mathit{make}(\texttt{Sequence}, v_1\, v_2)\})\,\{\texttt{1}\}\,\{\texttt{2}\})\,\{\texttt{0}\}\ .
\end{aligned}
$$

We call a system ω conforming to a UML static structure Ω *primitive recursive* if all functions $\mathit{attr}^{\omega}_{\Omega}(a,\tau) : \mathit{Value}^{\omega}_{\Omega} \to \mathit{Result}^{\omega}_{\Omega}$, $\mathit{meth}^{\omega}_{\Omega}(o,\tau) : \mathit{Value}^{\omega}_{\Omega} \times \mathit{Value}^{\omega}_{\Omega}{}^{*} \to \mathit{Result}^{\omega}_{\Omega}$, defined in ω and the relation $:_{\Omega} \subseteq \mathit{Value}^{\omega}_{\Omega} \times T_{\Omega}$ are primitive recursive.

Proposition 3. *Let Ω be a UML static structure and ω a primitive recursive system conforming to Ω; let t be a Term and $\tilde{t}$ be an A-Term; let $\tau \in T_{\Omega}$. If $\Gamma \vdash t \triangleright \tilde{t} : \tau$, then there is a primitive recursive function $\mathit{eval}(\tilde{t})^{\omega}_{\Omega} : \mathit{VarEnv}^{\omega}_{\Omega} \to \wp(\mathit{Result}^{\omega}_{\Omega})$ such that $\mathit{eval}(\tilde{t})^{\omega}_{\Omega}\,\gamma = [\![\tilde{t}]\!]^{\omega}_{\Omega}\,\gamma$.*

Proof. Using suitable encodings for $\mathit{VarEnv}^{\omega}_{\Omega}$, $\mathit{Result}^{\omega}_{\Omega}$, and finite sequences and sets of $\mathit{Result}^{\omega}_{\Omega}$, all auxiliary functions used in the definition of $[\![\tilde{t}]\!]^{\omega}_{\Omega}$ are primitive recursive. Thus, by induction over the term structure of $\tilde{t}$, all functions in $[\![\tilde{t}]\!]^{\omega}_{\Omega}$ are primitive recursive. Thus we may define $\mathit{eval}(\tilde{t})^{\omega}_{\Omega}\,\gamma = [\![\tilde{t}]\!]^{\omega}_{\Omega}\,\gamma$. □

Any system ω conforming to the empty UML static structure $\emptyset$ can obviously be chosen to be primitive recursive; thus, every OCL expression that is well-typed over the empty UML static structure, i.e., a pure OCL expression, denotes a primitive recursive function.

6 Conclusions

We have investigated the expressive power of the "Object Constraint Language." We have presented a type inference system, a big-step operational semantics, and denotational semantics for the OCL. This formal semantics has provided the basis for proving that the functions computed by pure OCL expressions are primitive recursive.

We have based our investigations on the OCL 1.3 specification [8, Ch. 7]. The forthcoming OCL 1.4 release may invalidate some of the findings discussed here. In particular, means for defining arbitrary recursive functions on the OCL level, and not on the UML static structure level only, may be provided, which would turn OCL into a Turing complete language.

References

1. Mark Bickford and David Guaspari. Lightweight Analysis of UML. Draft NAS1-20335/10, Odyssey Research Assoc., 1998. `http://cgi.omg.org/cgi-bin/doc?ad/98-10-01`.
2. Grady Booch, James Rumbaugh, and Ivar Jacobson. *The Unified Modeling Language User Guide*. Addison–Wesely, Reading, Mass., etc., 1998.
3. Tony Clark. Type Checking UML Static Diagrams. In Robert B. France and Bernhard Rumpe, editors, *Proc. 2nd Int. Conf. UML*, volume 1723 of *Lect. Notes Comp. Sci.*, pages 503–517. Springer, Berlin, 1999.
4. Sophia Drossopoulou and Susan Eisenbach. Describing the Semantics of Java and Proving Typing Soundness. In Jim Alves-Foss, editor, *Formal Syntax and Semantics of Java*, volume 1523 of *Lect. Notes Comp. Sci.*, pages 41–82. Springer, Berlin, 1999.
5. Ali Hami, John Howse, and Stuart Kent. Interpreting the Object Constraint Language. In *Proc. Asia Pacific Conf. Software Engineering*. IEEE Press, 1998.
6. Luis Mandel and María Victoria Cengarle. On the Expressive Power of OCL. In Jeannette M. Wing, Jim Woodcock, and Jim Davies, editors, *Proc. World Congress Formal Methods, Vol. 1*, volume 1708 of *Lect. Notes Comp. Sci.*, pages 854–874. Springer, Berlin, 1999.
7. John C. Mitchell. *Foundations for Programming Languages*. Foundations of Computing. MIT Press, Cambridge, Mass.–London, England, 1996.
8. Object Management Group. Unified Modeling Language Specification, Version 1.3. Technical report, Object Management Group, 1999. `http://cgi.omg.org/cgi-bin/doc?ad/99-06-08`.
9. Mark Richters and Martin Gogolla. On Formalizing the UML Object Constraint Language OCL. In Tok Wang Ling, Sudha Ram, and Mong Li Lee, editors, *Proc. 17th Int. Conf. Conceptual Modeling*, volume 1507 of *Lect. Notes Comp. Sci.*, pages 449–464. Springer, Berlin, 1998.
10. Jos Warmer and Anneke Kleppe. *The Object Constraint Language*. Addison–Wesely, Reading, Mass., &c., 1999.