

Appendix E (refined)

System of Transformation Rules: From OCL to Nereus

Refined consolidation of the original Appendix E. 27 rules; explicit handling of type tests, casts, transitive closures, and multi-variable iterators.

Preliminaries

This appendix describes a system of transformation rules from OCL constraints attached to MOF metamodels to Nereus axioms. The system consolidates the work of the original Appendix E by removing redundancy and explicitly covering type tests, casts, transitive closures, and multi-variable iterators.

Notation

- $\llbracket E \rrbracket$ denotes the recursive application of the system to OCL expression E .
- `self` in OCL maps to a Nereus variable `c` introduced in the `AXIOMS` clause and declared with the type of the OCL context.
- $[x \mapsto y]$ denotes substitution of x by y in the result of $\llbracket \cdot \rrbracket$.
- Pattern parts in italics (e.g. *roleName*) are placeholders to be substituted with the concrete identifier.
- Generated predicates: for each class C in the source MOF metamodel, a predicate `isC: AnySort  $\rightarrow$  Boolean` is automatically generated (see Conventions below).

Conventions

C1 — Uniform `Set` return for role navigation. All association ends in Nereus produce `Set[T]`, regardless of multiplicity. Chained navigations are lifted with `collect` (rule R16).

C2 — Class-membership predicates. For each MOF class C , the transformer generates a predicate `isC`:

- If C has no subclasses in the metamodel, `isC` is defined by axioms over the constructors of C .
- If C has direct subclasses $C_1, \dots, C_n$, then `isC(x)  $\equiv$  isC1(x) or ... or isCn(x)` (mechanically generated, no manual axioms).

C3 — Static type inference required. Rules R20 (cast) and R27 (type-specific operation) require knowing the static type of subexpressions, obtained from the MOF metamodel.

C4 — `@pre` is not supported. Postconditions using `@pre` cause the transformer to emit a warning and skip translation of that constraint. Manual translation is required.

C5 — Context variable. Each OCL context introduces a single fresh Nereus variable (we use `c` throughout) bound to the receiver of the appropriate type.

Reading the rules

Each rule has the form: OCL pattern in one column, Nereus pattern in the other. The Nereus pattern is constructed from the OCL pattern by applying $\llbracket \cdot \rrbracket$ recursively to subexpressions. Some rules have an explicit Condition column.

Rules

A. SIGNATURES AND ASSERTIONS

#	OCL	Nereus
R1	$T::op(p_1: T_1, \dots, p_n: T_n): RT$	$op: T * T_1 * \dots * T_n \rightarrow RT$
R2	$T::op(p_1: T_1, \dots, p_n: T_n): RT$ pre: P	$op: T(c) * T_1(p_1) * \dots * T_n(p_n) \rightarrow RT$ pre: $\llbracket P \rrbracket[self \mapsto c]$
R3	$T::op(p_1: T_1, \dots, p_n: T_n): RT$ post: result = E	AXIOMS $c: T; p_1: T_1; \dots; p_n: T_n$ $op(c, p_1, \dots, p_n) = \llbracket E \rrbracket[self \mapsto c]$
R4	$T::op(p_1: T_1, \dots, p_n: T_n): RT$ post: P (where P is not of the form result = E)	AXIOMS $c: T; p_1: T_1; \dots; p_n: T_n$ $\llbracket P \rrbracket[self \mapsto c, result \mapsto op(c, p_1, \dots, p_n)]$
R5	context T inv: P	AXIOMS $c: T; \dots$ $\llbracket P \rrbracket[self \mapsto c]$

B. BASIC EXPRESSIONS

#	OCL	Nereus
R6	v (variable)	v
R7	$self$	c (the context variable)
R8	$e.attr$ (where $attr$ is an attribute, not a role)	$attr(\llbracket e \rrbracket)$
R9	$e.op(a_1, \dots, a_n)$ (where op is an operation, not a role)	$op(\llbracket e \rrbracket, \llbracket a_1 \rrbracket, \dots, \llbracket a_n \rrbracket)$
R10	$e_1 \oplus e_2$ where $\oplus \in \{+, -, *, /, =, <>, <, >, <=, >=, \text{and}, \text{or}, \text{xor}, \text{implies}\}$	$\llbracket e_1 \rrbracket \oplus \llbracket e_2 \rrbracket$ with $<>$ translated to $\text{not}(\dots = \dots)$
R11	$\ominus e$ where $\ominus \in \{\text{not}, -\}$	$\ominus \llbracket e \rrbracket$
R12	if E_0 then E_1 else E_2 endif	if $\llbracket E_0 \rrbracket$ then $\llbracket E_1 \rrbracket$ else $\llbracket E_2 \rrbracket$ endif
R13	let $v: T = E_1$ in E_2	LET $v = \llbracket E_1 \rrbracket$ IN $\llbracket E_2 \rrbracket$ END-LET
R14	$Pkg::Name$	$Pkg::Name$

C. ASSOCIATION NAVIGATIONS

#	OCL	Nereus
R15	$e.roleName$ (where $roleName$ is an association end of $AssocName$)	$get_roleName(a, \llbracket e \rrbracket)$ with $a: AssocName$ declared in AXIOMS
R16	$e.role1.role2$ (chained navigation; result of $e.role1$ is $Set[T]$ by C1)	$collect(v)(get_role1(a_1, \llbracket e \rrbracket), [get_role2(a_2, v)])$ with v fresh; applied recursively for longer chains

D. COLLECTION LITERALS

#	OCL	Nereus
R17	$Set\{\} / Set\{e_1, \dots, e_n\}$	$create() \text{ as } Set / addSet(\dots addSet(create() \text{ as } Set, \llbracket e_1 \rrbracket) \dots, \llbracket e_n \rrbracket)$

#	OCL	Nereus
	Bag{} / Bag{e ₁ , ..., e _n }	createBag() / includingBag(...includingBag(createBag(), [[e ₁]])..., [[e _n]])
	Sequence{} / Sequence{e ₁ , ..., e _n }	create() as Sequence / add(...add(create() as Sequence, [[e ₁]])..., [[e _n]])
	OrderedSet{} / OrderedSet{e ₁ , ..., e _n }	create() as OrderedSet / addOrderedSet(...(create() as OrderedSet, [[e ₁]])..., [[e _n]])

E. TYPE TESTS AND CAST

#	OCL	Nereus	Condition
R18	e.oclIsKindOf(C)	isC([[e]])	always
R19	e.oclIsTypeOf(C)	isC([[e]])	C has no direct subclasses
	e.oclIsTypeOf(C)	isC([[e]]) and not (isC ₁ ([[e]]) or ... or isC _n ([[e]]))	C has direct subclasses C ₁ , ..., C _n
R20	e.oclAsType(T)	[[e]] as T	T is a strict subtype of the static type of e
	e.oclAsType(T)	[[e]]	T is a supertype (or equal to) the static type of e

F. ITERATORS

#	OCL	Nereus
R21	c -> op(v: T body), op ∈ {forAll, exists}	op(v)([[c]], [[body]])
	c -> op(v body) (no type annotation)	op(v)([[c]], [[body]])
	c -> op(body) (no bound variable)	op(v)([[c]], [[body]]) with v fresh, not occurring in body
R22	c -> op(v: T body), op ∈ {select, reject}	op(v)([[c]], [[body]])
R23	c -> collect(v: T expr)	collect(v)([[c]], [[expr]])
	c.prop (shorthand for c -> collect(prop)) where c is a collection	collect(v)([[c]], [prop(v)]) with v fresh of type elem(c)
R24	c -> iterate(v; acc: A = init body)	iterate(v)([[c]], [[body]], [[init]])
R25	c -> closure(v: T expr)	closure(v)([[c]], [[expr]])
	c -> closure(prop) (shorthand)	closure(v)([[c]], [prop(v)]) with v fresh

G. MULTI-VARIABLE ITERATORS

#	OCL	Nereus
R26	c -> op(v ₁ , v ₂ , ..., v _n body), op ∈ {forAll, exists}	op(v ₁)([[c]], [op(v ₂)([[c]], [... op(v _n)([[c]], [[body]] ...])])

Cartesian products over distinct collections (c₁, c₂ -> forAll(v₁, v₂ | ...)) are not covered and must be rewritten manually as nested iterators over the respective collections.

H. TYPE-SPECIFIC COLLECTION OPERATIONS

#	OCL	Nereus
R27	$c \rightarrow op(args)$ (where op has a type-specific name in the Nereus library)	$op_i(\llbracket c \rrbracket, \llbracket args \rrbracket)$ with T the static type of c and op_i from the name-mapping table below

Name-mapping table (OCL $\rightarrow$ Nereus, per receiver type)

OCL	Set	Bag	Sequence	OrderedSet
union	unionSet	unionBag	—	—
intersection	intersectionSet	intersectionBag	—	—
- (difference)	difference	—	—	—
including	addSet	includingBag	includingSequence	addOrderedSet
excluding	excludingSet	excludingBag	excludingSequence	excludingOrderedSet
symmetricDifference	symmetricDifference	—	—	—
at	—	—	atSequence	atOrderedSet
first	—	—	firstSequence	firstOrderedSet
last	—	—	lastSequence	lastOrderedSet
indexOf	—	—	indexOfSequence	indexOfOrderedSet
subSequence / subOrderedSet	—	—	subSequence	subOrderedSet
prepend	—	—	prependSequence	prependOrderedSet
insertAt	—	—	insertAtSequence	insertAtOrderedSet
asSet	identity (elided)	BagAsSet	sequenceAsSet	—
asBag	setAsBag	identity (elided)	sequenceAsBag	—
asSequence	setAsSequence	BagAsSequence	identity (elided)	—
asOrderedSet	setAsOrderedSet	BagAsOrderedSet	sequenceAsOrderedSet	identity (elided)

Generic operations (no suffix needed; defined in the `Collection` class): `size`, `isEmpty`, `notEmpty`, `includes`, `excludes`, `includesAll`, `excludesAll`, `count`, `forAll`, `exists`, `select`, `reject`, `collect`, `iterate`, `closure`.

When an identity conversion is encountered (e.g., `Set.asSet`), the term is elided: $\llbracket c.asSet \rrbracket \equiv \llbracket c \rrbracket$.

Worked example: Person&Meeting

Given the OCL constraints attached to the `Person&Meeting` metamodel (Figure 27 of the report):

```

context Person::numMeeting(): Nat
post: result = self.meetings -> size

context Person::numConfirmedMeeting(): Nat
post: result = self.meetings -> select(isConfirmed) -> size

context Meeting::duration(): Time
post: result = timeDifference(self.end, self.start)

context Meeting::checkDate(): Bool
post: result = self.participants -> collect(meetings) ->
  forAll(m | m <> self and m.isConfirmed implies
    (after(self.end, m.start) or after(m.end, self.start)))

context Meeting::isConfirmed()
post: result = self.checkdate() and self.numConfirmedParticipants >= 2

```

The system produces:

numMeetings (by R3, R15, R9)

```

AXIOMS c: Person; pa: <<Participates>>
  numMeetings(c) = size(get_meetings(pa, c))

```

numConfirmedMeetings (by R3, R15, R22, R9)

```

AXIOMS c: Person; pa: <<Participates>>
  numConfirmedMeetings(c) =
    size(select(m)(get_meetings(pa, c), [isConfirmed(m)]))

```

duration (by R3, R8, R9)

```

AXIOMS c: Meeting
  duration(c) = timeDifference(end(c), start(c))

```

checkDate (by R3, R15, R23-shorthand, R21, R10)

```

AXIOMS c: Meeting; pa: <<Participates>>
  checkDate(c) =
    forAll(m)(
      collect(p)(get_participants(pa, c), [get_meetings(pa, p)]),
      [(not (m = c) and isConfirmed(m)) implies
        (after(end(c), start(m)) or after(end(m), start(c)))]
    )

```

isConfirmed (by R3, R9, R8)

```

AXIOMS c: Meeting
  isConfirmed(c) = checkDate(c) and (numConfirmedParticipants(c) >= 2)

```

This exactly matches the manually-written axioms in Section 6.2.10 of the book, modulo the typo `select(p)` → `collect(p)` in the book.

Worked example: TRACEM (selected constraints)

Given the OCL of Figure 7 (TRACEM metamodel), the system produces:

Derived returnedObject of MethodCall (by R3, R16, R22, R18, R23-shorthand)

```
AXIOMS c: MethodCall;
  a1: <<Call_ExecutionTrace>>;
  a2: <<ExecutionTrace_Statement>>;
  a6: <<Return_Object>>
returnedObject(c) =
  collect(r')(
    select(r)(
      collect(s)(get_subtrace(a1, c), [get_statements(a2, s)]),
      [isReturn(r)]),
    [get_returnedObject(a6, r')])
```

Note that `oclIsTypeOf(r, return)` from the original OCL is translated via R19 (case “no subclasses”) to `isReturn(r)`, since `Return` is a leaf class.

Invariant of Compound (by R5, R15, R22, R21, R18)

```
AXIOMS c: Compound; a: <<Compound_ExecutionSentence>>
  forAll(l)(
    select(s)(get_statements(a, c), [isLocation(s)]),
    [isLocalVariable(l)])
```

The translation surfaces that `oclIsTypeOf(s, Location)` in the original OCL was likely a metamodel inconsistency (Location is abstract), and the rule resolves it as the union of `Location`’s leaf subclasses through the derived `isLocation` predicate.

Invariant of MethodCall (parameter binding) (by R5, R15, R21, R16, R22, R18)

```
AXIOMS c: MethodCall;
  a1: <<Call_ExecutionTrace>>;
  a2: <<ExecutionTrace_Statement>>;
  a7: <<MethodCall_Object_actualParameters>>
forAll(ap)(
  get_actualParameters(a7, c),
  [exists(fp)(
    select(t)(
      collect(s)(get_subtrace(a1, c), [get_statements(a2, s)]),
      [isFormalParameter(t)]),
    [allocateObject(fp) = ap]])])
```

The pattern `collect(oclIsTypeOf(FormalParameter))` of the original OCL is read as `select(t | t.oclIsTypeOf(FormalParameter))` because `oclIsTypeOf` returns a `Boolean` and `collect` requires a non-Boolean projection. The transformer either applies this normalization automatically or reports a warning to the metadesigner.

Limitations and warnings emitted by the transformer

1. **@pre not supported (C4).** Occurrence of `@pre` in any subexpression of a postcondition triggers a warning of the form:

```
Warning: use of '@pre' in postcondition of T::op is not supported.
Translation of this constraint is skipped. Rewrite manually using
the pre-state variable (the operation receiver) and the post-state
expression op(c, ...).
```

2. **Cartesian products over distinct collections (R26 limitation).** OCL `c1, c2 -> forAll(v1, v2 | ...)` requires manual rewriting.
 3. **Mismatched collect/select.** If the body of a `collect` returns `Boolean`, the transformer warns and suggests reading it as `select`. Symmetrically, a `select` body that does not return `Boolean` causes a warning.
 4. **oclIsTypeOf on abstract classes.** Always evaluates to `False` after translation, regardless of the metamodel. A warning is emitted to alert the metadesigner.
 5. **Library naming gaps (R27 table).** If an OCL operation appears on a receiver type for which no entry exists in the name-mapping table, the transformer falls back to literal translation `op(⟦c⟧, ⟦args⟧)` and warns about a possible name mismatch.
-

Summary of changes from the original Appendix E

- The 52 rules of the original Appendix E are consolidated into **27 rules**.
- Variants of the same iterator (with/without type, with/without variable) are unified under a single rule with three cases.
- Per-collection-type variants of `iterate`, `forAll`, `exists`, `select`, `reject`, `collect` are subsumed by a single rule plus the type-specific operations table.
- New rules cover `oclIsKindOf` (R18), `oclIsTypeOf` (R19), `oclAsType` (R20), `closure` (R25), and multi-variable iterators (R26).
- The convention C1 (uniform `Set` for navigation) and C2 (generated `isC` predicates) are made explicit, eliminating ambiguity in chained navigations and type tests.
- `@pre` is explicitly deferred (C4) rather than half-handled with redundant rules.