Semantics of OCL Operation Specifications

Rolf Hennicker and Alexander Knapp and Hubert Baumeister

Ludwig-Maximilians-Universität München
Institut für Informatik, Oettingenstraße 67, 80 538 München, Germany
+(49) 89 2180 9184
{*hennicke,knapp,baumeist*}*@pst.informatik.uni-muenchen.de*

Abstract

The semantics of OCL operation specifications is discussed from a model theoretic perspective. It is argued that the semantics of operation specifications as defined in the OCL 2.0 proposal is not compatible with the view of operation specifications as contracts between a client and a supplier. As a solution, a semantics of OCL operation specifications based on standard model theory is presented. This semantics introduces the concept of a model over a UML class signature — which is a labelled transition system with output — together with a notion of the satisfiability of an OCL operation specification w.r.t. a model. The models respect the OCL features for methods with and without results, constructors, and queries. Regarding inheritance, the combination of several OCL operation specifications is introduced based on a lattice structure on models with respect to generalisation and refinement. Satisfiability is parametric in the underlying signature, and thus the notion can be transferred from UML class signatures to signatures including invariants.

1 Introduction

An important application area of the "Object Constraint Language" (OCL [13]) is the specification of preconditions and postconditions of operations occurring in UML static structure diagrams. Although much work has been done to formalise the semantics of the OCL expression language (see, e.g., [17,18,4]), much less effort is still spent on a detailed semantics of operation specifications which is needed for an unambiguous interpretation of precondition and postcondition constraints. Important questions that arise here concern the meaning of operation specifications in the context of local and global invariants, the combination of constraints, the inheritance of constraints, and the meaning of constructor and query specifications.

⋆ This research has been partially supported by the GLOWA-Danube project (07GWK04) sponsored by the German Federal Ministry of Education and Research and the EC 5th Framework project AGILE: Architectures for Mobility (IST-2001-32747).

Suggestions for the semantics of OCL operation specifications have been provided by Bickford and Guaspari [2], Richters and Gogolla [15], Hennicker, Hußmann, and Bidoit [9], Brucker and Wolff [3], and the OCL 2.0 specification itself [13] based on Richters [14]. Given a constraint of the form C`::`op`(`$\ldots$`)` `pre:` P `post:` Q, these approaches can be classified by two styles of interpretation, in the following called the "$P \Rightarrow Q$" style and the "$P \wedge Q$" style (see also [3]). The $P \Rightarrow Q$ style ([2,15,9]) basically requires that if the precondition P is satisfied in the state σ before the operation is performed then the postcondition is satisfied in the state σ' after the execution of the operation. If the precondition is not satisfied in σ then the operation yields an arbitrary result. On the other hand, the $P \wedge Q$ style ([3,13,14]) considers relations (or state transitions) between pre- and poststates which simply do not contain any pair (σ, σ') where the precondition is not satisfied in the prestate σ. (For the time being, we deliberately neglect the issue of termination and the semantic variants considered in [9].)

To discuss which approach is more appropriate, one needs a simple, intuitive background which we believe is provided by the notion of a *contract* as described by Meyer [12]. The contract principle assumes two actors, a client who uses (i.e. calls) an operation and an implementor who realises the operation. Both actors have responsibilities. The client has the responsibility to call an operation only in a state where its precondition is satisfied. On the other hand, an implementor can assume that the precondition is valid and must ensure that after execution of the operation its postcondition is satisfied. If both actors fulfill their responsibilities it is guaranteed that the system is executing correctly. Both views on a system, the client's view and the implementor's view should be reflected by the semantics of an operation specification which should characterise the correct system runs.

Indeed, it turns out that the $P \Rightarrow Q$ style fits perfectly with the implementor's view, but does not take into account the obligations of the client because (arbitrary) state transitions are possible if the precondition of the operation is not satisfied. Hence this kind of semantics subsumes also incorrect system runs which can never occur if the client fulfils his responsibilities. Such incorrect transitions are excluded in the $P \wedge Q$ style to the semantics of operation specifications, which is adopted by the OCL 2.0 specification. In this semantics, the following two specifications Spec_1 and Spec_2 are semantically equivalent since they describe the same state transitions (given by state pairs (σ, σ') such that σ satisfies P and (σ, σ') satisfies Q):

$$\mathit{Spec}_1 = \texttt{context}\ C\texttt{::}op\texttt{(}\ldots\texttt{)}\ \texttt{pre:}\ P\ \texttt{post:}\ Q$$
$$\mathit{Spec}_2 = \texttt{context}\ C\texttt{::}op\texttt{(}\ldots\texttt{)}\ \texttt{pre: true post:}\ P\texttt{@pre and}\ Q$$

where P`@pre` denotes the expression obtained from P by replacing each property name a occurring in P by a`@pre`.

However, when viewed as contracts, both specifications yield different obligations for the client and the implementor of the specifications. In the case of Spec_1 the client has to establish the precondition P while the implementor has to establish the postcondition Q under the assumption that P holds. In the case of Spec_2 the client has no obligation since the precondition is `true`. The implementor, however, must in any situation satisfy the postcondition of $Spec_2$ which may well be impossible if P is a proper precondition different from `true`. Then the implementor can not fulfil his responsibility which means that $Spec_2$ is not satisfiable. In practice, satisfiability is particularly important in the connection with invariants which impose further constraints on correct realisations. The semantics according to the $P \wedge Q$ style does not reflect the important notion of satisfiability because it is simple to construct specifications $Spec_1$ and $Spec_2$ such that $Spec_1$ is satisfiable but $Spec_2$ is not, but nevertheless both specifications are equivalent according to the $P \wedge Q$ style.

In this paper we present a semantics of OCL operation specifications that has the benefits of the $P \wedge Q$ style and incorporates the view of pre- and postcondition specifications as contracts, which allows us to distinguish $Spec_1$ and $Spec_2$. The solution is based on model theory and mathematical logic where satisfiability is a standard notion. Satisfiability of a formula φ means that there is a model of φ, i.e. a structure for the chosen logic which satisfies φ. Hence we have to investigate in the context of OCL appropriate notions of formula, model, and satisfaction relation: Formulas are OCL operation specifications over a UML class signature; models are a special class of labelled transition systems with output that respect the contract view. Furthermore, we investigate the concept of correct realisations of operation specifications and relate them to the notion of models (Sect. 3). The class of models shows a lattice structure with respect to a generalisation ordering which provides the means for combining operation specification, in particular in view of inheritance of operation specifications (Sect. 4). Incidentally, the satisfaction relation following from the notion of models can be taken to be parameterised in the class of signatures. Extending UML class signatures with invariant specifications, the notion of formulas and models can be reused to include invariants (Sect. 5).

Besides employing Meyer's contract paradigm, our approach combines several well-known principles and results for operation specifications from formal specification languages like Z, Object-Z, or VDM with OCL. Jones [10] makes the case of implicit preconditions that may render a postcondition not satisfiable for some states allowed in the explicit precondition, which has become known as the satisfiability principle (cf. also [11]). The notion of generalisation between models corresponds to the refinement relation as discussed by Derrick and Boiten [6].

2 Preliminaries

We briefly sketch the necessary prerequisites for interpreting OCL expressions. More detailed accounts of a formal semantics for OCL's navigational expression language have been given by, e.g., Gogolla and Richters [17], Schmitt [18], Cengarle and Knapp [4], and the OCL 2.0 proposal [13] itself. However, we pay special attention to undefined and `null` values and their semantics in connection with attributes and queries. This is to ensure that attribute and query valuations in states may only mention objects that are indeed part of the state. Moreover, we distinguish between attributes and queries in states, treating states as being uniquely defined by their instances and attribute valuations, but assigning queries a derived status.

OCL expressions are built inductively over a class signature and variables. An OCL expression is evaluated over an environment, binding the expression's free variables; a prestate and a poststate that are used to retrieve the values of attributes and opposite association ends; and a query interpretation.

Syntax. A *class signature* Σ is defined over a UML static structure diagram. It contains sort symbols T for all classes T of the UML diagram, the predefined OCL basic types like `Integer`, the OCL collection types like `Set(`T`)`, and the OCL tuple types. The OCL type conformance rules are represented by a partial order $\leq$, the *type subsumption* relation, on the sort symbols. Finally, a class signature contains *operations*. These operations comprise the predefined OCL standard operations, like `_->includes(_)` : `Collection(`T`)` $\times T \rightarrow$ `Boolean`, and operations induced by attributes, opposite association ends, and queries from the UML static structure diagram. For example, if a is an attribute of class C with type D and multiplicity `1`, then the class signature contains an operation `_.`$a : C \rightarrow D$.

Semantic domains. Each sort symbol T of a class signature Σ is mapped into a semantic domain of *values* $[\![T]\!]$. The semantic domain of values over a class signature Σ is denoted by Value_Σ and it is required that the domains $[\![T]\!]$ are included in Value_Σ. Every such domain $[\![T]\!]$ contains the special value $\perp$ for *undefined*. For sort symbols induced by classes, we assume infinite sets of *object identifiers* as semantic domains together with a constant null denoting `null`. We require that $T \leq T'$ implies $[\![T]\!] \subseteq [\![T']\!]$. In particular, this means that an object identifier for a class D is also an object identifier for class C if D is a subclass of C.

A *state* defines a finite set of existing instances, or objects, for each class type and the valuation of object attributes and association ends. The semantic domain of states over a given class signature Σ is written as State_Σ. Given a class type C, we write $[\![C]\!]_\sigma$ for the finite set of instances of C that exist in state σ together with the null constant; in particular $\perp \notin [\![C]\!]_\sigma$, but $\mathit{null} \in [\![C]\!]_\sigma$. For an OCL basic type T we set $[\![T]\!]_\sigma = [\![T]\!] \setminus \{\perp\}$. These definitions on class and OCL basic types are extended canonically to OCL collection and tuple types. An attribute `_.`$a : C \rightarrow T$ is evaluated on an object (identifier) $o \in [\![C]\!]$, written as $\sigma(o.a)$, yielding a value in $[\![T]\!]_\sigma$. We additionally require $\sigma(o.a) = \perp$ if, and only if $o \notin [\![C]\!]_\sigma \setminus \{\mathit{null}\}$.

Evaluation. We assume suitable interpretations $[\![\ldots]\!]$ for the OCL predefined operations and term formers, like `_->iterate(`$\ldots$`)` or `_->exists(`$\ldots$`)`. These operations do not depend on a state. Their interpretation has to be strict, i.e. whenever an argument is undefined then the result is undefined. The only exceptions to this general rule are the boolean connectives `_and_`, `_or_`, and `_implies_`, such that, for instance, $[\![$`_and_`$]\!](\perp, \mathit{false}) = \mathit{false}$ (see e.g. [17,4]).

An *environment* over a class signature Σ maps variables into values of the semantic domain corresponding to the variable's type, they are written as $[x \mapsto v, y \mapsto v']$ where x, y are variables and v, v' are values.

A *query interpretation* over a class signature Σ is a function that maps each query operation symbol to a function on a state and the query arguments yielding a value. If $q : C \times T_1 \times \cdots \times T_n \to T_0$ is an operation symbol in a class signature Σ induced by a query of the UML static structure diagram, then a query interpretation I maps q to a function in $\mathit{State}_\Sigma \times [\![C]\!] \times [\![T_1]\!] \times \cdots \times [\![T_n]\!] \to [\![T_0]\!]$ such that $I(q)(\sigma, o, v_1, \ldots, v_n) = \bot$ if $(o, v_1, \ldots, v_n) \notin (([\![C]\!]_\sigma \setminus \{\mathit{null}\}) \times [\![T_1]\!]_\sigma \times \cdots \times [\![T_n]\!]_\sigma)$ and $I(q)(\sigma, o, v_1, \ldots, v_n) \in [\![T_0]\!]_\sigma \cup \{\bot\}$ otherwise. By abuse of notation, we write $q^I(\sigma, o, v_1, \ldots, v_n)$ instead of $I(q)(\sigma, o, v_1, \ldots, v_n)$.

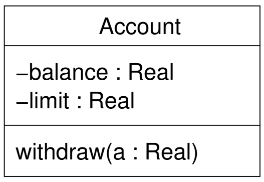
The *evaluation* of an OCL expression e over a class signature Σ is written as $[\![e]\!]_{\beta,\sigma,\sigma',I}$ where β is an environment, σ and σ' are states over Σ, and I is a query interpretation over Σ. Note that the selection of the appropriate query interpretation for a query call on an object depends on the dynamic type of an object. For example, if x is a variable of type Integer, $[\![\texttt{x = 0}]\!]_{[\mathsf{x}\mapsto 0],\sigma,\sigma',I} = \mathit{true}$ for all states σ, σ' and for all query interpretations I; and if q : $C \to$ Integer is a query, $[\![\texttt{self.q@pre() + self.q()}]\!]_{\beta,\sigma,\sigma',I} = \mathtt{q}^I(\sigma, \beta(\mathtt{self})) + \mathtt{q}^I(\sigma', \beta(\mathtt{self}))$.

3 Semantics of OCL Operation Specifications

The semantics of OCL operation specifications is modelled on Meyer's contract view of operation specifications [12]: The user, or client, of an operation has to meet the operation's precondition, the implementor, or supplier, of an operation has to meet the operation's postcondition, if the precondition has been satisfied. More concretely, we take operation calls to induce transitions between states of a system. In the semantics, a transition between states may only exist if both the client and the supplier of the operation specification corresponding to the operation call have met their respective duties, i.e., in the source state the precondition holds and in the target state the postcondition holds. In fact, the postcondition may refer both to the source and the target state. However, it may well be impossible to find a state satisfying the postcondition depending on the source state. We are thus lead to conditions on the satisfiability of OCL operation specifications when viewed as contracts: On the one hand, the client should be able to call the operation. On the other hand, the supplier must be able to reach a state where the postcondition is satisfied, whenever the precondition holds. The latter condition corresponds to the well-known *satisfiability* or *feasibility* principle for operation specifications as advocated by Jones [10].

Consider the (simple) example of an account specification in Fig. 1. In fact, the operation specification of withdraw violates the feasibility principle, as subtracting an arbitrary amount from balance may make it impossible to satisfy balance >= limit. According to the contract view, no transition system should be a model of this operation specification. When, however, the precondition of withdraw is strengthened to balance >= limit and balance-a >= limit, the operation specification becomes satisfiable.

We first define the particular notion of labelled transitions with output that forms the general semantic domain of the semantics of OCL operation specifications. Then we define the sets of states where a given precondition or postcondition is



```
context Account::withdraw(a : Real)
pre: balance >= limit
post: balance = balance@pre-a and balance >= limit
```

Fig. 1. UML/OCL specification for account example

satisfied. With these tools, we define the models of an operation specification and the notion of satisfiability of an operation specification. The models of an operation specification allow the precondition to be weakened and the postcondition to be strengthened. However, we prove that a canonical model can be chosen as the semantics of a satisfiable operation specification. This canonical model exactly reflects the contract view. Finally, we define the possible implementations of a satisfiable operation specification.

3.1 Labelled Transition Systems with Output

A labelled transition system with output over a class signature Σ describes transitions between states in a system. The states are given by State_Σ. The transitions represent the possible changes of state induced by a call to an operation. Each transition is labelled by the operation name, the callee, and the actual arguments. As operation calls may return a result, these outcomes are recorded in an additional field of the labelled transition system.

For OCL, actually, the labels may take different forms, depending on which kind of behavioural feature is called. We define the set Label_Σ to comprise the following: For each method (with or without a result) or query of class C with parameter types $T_1, \ldots, T_n$ the labels $o.\mathit{op}(v_1, \ldots, v_n)$ with $o \in [\![C]\!]$ and $v_1 \in [\![T_1]\!], \ldots, v_n \in [\![T_n]\!]$. For each constructor of class C with parameter types $T_1, \ldots, T_n$ the labels $C(v_1, \ldots, v_n)$ with $v_1 \in [\![T_1]\!], \ldots, v_n \in [\![T_n]\!]$. Analogously, result kinds may differ. We define the set Result_Σ to comprise Value_Σ and a special result $*$ that is used for methods without an explicit result.

Formally, a *labelled transition system with output* (*ltso*) over the class signature Σ is a subset of $\mathit{State}_\Sigma \times \mathit{Label}_\Sigma \times \mathit{State}_\Sigma \times \mathit{Result}_\Sigma$. The *domain* of a Σ-ltso S is given by $\mathrm{dom}(S) = \{(\sigma, l) \mid \exists(\sigma', r) \,.\, (\sigma, l, \sigma', r) \in S\}$. The *codomain* of a Σ-ltso S for a set $M \subseteq \mathit{State}_\Sigma \times \mathit{Label}_\Sigma$ is given by $S(M) = \{(\sigma', r) \mid \exists(\sigma, l) \in M \,.\, (\sigma, l, \sigma', r) \in S\}$. We also write $S(\sigma, l)$ for $S(\{(\sigma, l)\})$.

We define a *generalisation* relation $\sqsubseteq$ between Σ-ltsos S, S' as follows: $S \sqsubseteq S'$ if

(i) $\mathrm{dom}(S') \subseteq \mathrm{dom}(S)$

(ii) $S(\sigma, l) \subseteq S'(\sigma, l)$ for all $(\sigma, l) \in \mathrm{dom}(S')$.

The generalisation relation corresponds to the *refinement* relation for Z specifications (cf. [6]) and is a partial order on Σ-ltsos. Moreover, the ordering induces universal meet (conjunction) and join (disjunction) operations $\sqcap$ and $\sqcup$ for Σ-ltsos: The meet of a family of Σ-ltsos is a Σ-ltso that is less or equal than every single member and the largest Σ-ltso fulfilling this condition, the join of Σ-ltsos affords the dual construction. Meet and join of a family of Σ-ltsos $(S_j)_{j\in J}$ are explicitly given by

$$\begin{aligned}
\textstyle\bigsqcap_{j\in J} S_j = \{(\sigma, l, \sigma', r) \mid {} & (\sigma, l) \in \textstyle\bigcup_{j\in J} \mathrm{dom}(S_j), \\
& (\sigma', r) \in \textstyle\bigcap_{k\in\{k\in J \mid (\sigma,l)\in\mathrm{dom}(S_k)\}} S_k(\sigma, l)\}\ , \\
\textstyle\bigsqcup_{j\in J} S_j = \{(\sigma, l, \sigma', r) \mid {} & (\sigma, l) \in \textstyle\bigcap_{j\in J} \mathrm{dom}(S_j), \\
& (\sigma', r) \in \textstyle\bigcup_{k\in\{k\in J \mid (\sigma,l)\in\mathrm{dom}(S_k)\}} S_k(\sigma, l)\}\ .
\end{aligned}$$

As it stands, the notion of Σ-ltsos does not include the interpretation of queries. A query interpretation, in fact, presents a direct, functional way of expressing possible transitions in a system. The functional interpretation of query operations in Σ must therefore be *compatible* with the transitions of a Σ-ltso, when both are applicable. We thus define a *system* over a class signature Σ as a pair (S, I) of a Σ-ltso and a query interpretation I over Σ such that if $q^I(\sigma, o, v_1, \ldots, v_n) = v_0$ then $(\sigma, o.q(v_1, \ldots, v_n), \sigma, v_0) \in S$. The partial order $\sqsubseteq$ on Σ-ltsos is extended canonically to a partial order $\sqsubseteq$ on Σ-systems: $(S, I) \sqsubseteq (S', I')$, if $S \sqsubseteq S'$ and I and I' coincide. Note that for a fixed query interpretation I, meet and join of families of systems $(S_j, I)_{j\in J}$ are well-defined by $\bigsqcap_{j\in J}(S_j, I) = (\bigsqcap_{j\in J} S_j, I)$ and $\bigsqcup_{j\in J}(S_j, I) = (\bigsqcup_{j\in J} S_j, I)$, respectively.

3.2 Precondition and Postcondition Domains

The precondition and postcondition domains of an operation specification yield the set of states or pairs of states where the precondition or postcondition of the operation specification holds. The definition of the precondition and the postcondition domain is by case analysis on the form of the operation specification. The differences between OCL operation specifications for methods with result, methods without result, constructors, and queries are marginal, but warrant separate consideration for a complete treatment. In the following, we detail the precondition and the postcondition domain for operations specifications for methods with result; the remaining cases are summarised in Table 1.

An operation specification φ over a class signature Σ for a method op with result takes the general form

```
context $T$::$op$($x_1$ : $T_1$, ..., $x_n$ : $T_n$) : $T_0$
pre: $P$
post: $Q$
```

where the OCL expression P may contain as free variables $\mathrm{fv}(P)$ only `self` and $x_1, \ldots, x_n$, the OCL expression Q may contain as free variables $\mathrm{fv}(Q)$ only `self`,

Method without result $\varphi = \texttt{context}\ C\texttt{::}op\texttt{(}x_1\ \texttt{:}\ T_1\texttt{,}\ \ldots\texttt{,}\ x_n\ \texttt{:}\ T_n\texttt{)}$
$\texttt{pre:}\ P\ \ \texttt{post:}\ Q$

Side condition: $\mathrm{fv}(P) \subseteq \{\texttt{self}, x_1, \ldots, x_n\}$, $\mathrm{fv}(Q) \subseteq \{\texttt{self}, x_1, \ldots, x_n\}$

$$\begin{aligned}
\mathrm{pre}^I_\Sigma(\varphi) = \{(\sigma, o.op(v_1, \ldots, v_n)) \mid{} & \sigma \in \mathit{State}_\Sigma, o \in [\![C]\!]_\sigma \setminus \{\mathit{null}\}, \\
& v_1 \in [\![T_1]\!]_\sigma, \ldots, v_n \in [\![T_n]\!]_\sigma, \\
& [\![P]\!]_{[\texttt{self}\mapsto o, x_1 \mapsto v_1, \ldots, x_n \mapsto v_n], \sigma, \sigma, I} = \mathit{true}\}
\end{aligned}$$

$$\begin{aligned}
\mathrm{post}^I_\Sigma(\varphi) = \{(\sigma, o.op(v_1, \ldots, v_n), \sigma', *) \mid{} & \sigma, \sigma' \in \mathit{State}_\Sigma, o \in [\![C]\!]_\sigma \setminus \{\mathit{null}\}, \\
& v_1 \in [\![T_1]\!]_\sigma, \ldots, v_n \in [\![T_n]\!]_\sigma, \\
& [\![Q]\!]_{[\texttt{self}\mapsto o, x_1 \mapsto v_1, \ldots, x_n \mapsto v_n], \sigma, \sigma', I} = \mathit{true}\}
\end{aligned}$$

Constructor $\varphi = \texttt{context}\ C\texttt{(}x_1\ \texttt{:}\ T_1\texttt{,}\ \ldots\texttt{,}\ x_n\ \texttt{:}\ T_n\texttt{)}$
$\texttt{pre:}\ P\ \ \texttt{post:}\ Q$

Side condition: $\mathrm{fv}(P) \subseteq \{x_1, \ldots, x_n\}$, $\mathrm{fv}(Q) \subseteq \{\texttt{self}, x_1, \ldots, x_n\}$

$$\begin{aligned}
\mathrm{pre}^I_\Sigma(\varphi) = \{(\sigma, C(v_1, \ldots, v_n)) \mid{} & \sigma \in \mathit{State}_\Sigma, \\
& v_1 \in [\![T_1]\!]_\sigma, \ldots, v_n \in [\![T_n]\!]_\sigma, \\
& [\![P]\!]_{[x_1 \mapsto v_1, \ldots, x_n \mapsto v_n], \sigma, \sigma, I} = \mathit{true}\}
\end{aligned}$$

$$\begin{aligned}
\mathrm{post}^I_\Sigma(\varphi) = \{(\sigma, C(v_1, \ldots, v_n), \sigma', r) \mid{} & \sigma, \sigma' \in \mathit{State}_\Sigma, \\
& v_1 \in [\![T_1]\!]_\sigma, \ldots, v_n \in [\![T_n]\!]_\sigma, r \in [\![C]\!]_{\sigma'} \setminus \{\mathit{null}\}, r \notin [\![C]\!]_\sigma, \\
& [\![Q]\!]_{[\texttt{self}\mapsto r, x_1 \mapsto v_1, \ldots, x_n \mapsto v_n], \sigma, \sigma', I} = \mathit{true}\}
\end{aligned}$$

Query $\varphi = \texttt{context}\ C\texttt{::}op\texttt{(}x_1\ \texttt{:}\ T_1\texttt{,}\ \ldots\texttt{,}\ x_n\ \texttt{:}\ T_n\texttt{)}\ \texttt{:}\ T_0$
$\texttt{pre:}\ P\ \ \texttt{post:}\ Q$

Side condition: $\mathrm{fv}(P) \subseteq \{\texttt{self}, x_1, \ldots, x_n\}$,
$\mathrm{fv}(Q) \subseteq \{\texttt{self}, x_1, \ldots, x_n, \texttt{result}\}$

$$\begin{aligned}
\mathrm{pre}^I_\Sigma(\varphi) = \{(\sigma, o.op(v_1, \ldots, v_n)) \mid{} & \sigma \in \mathit{State}_\Sigma, o \in [\![C]\!]_\sigma \setminus \{\mathit{null}\}, \\
& v_1 \in [\![T_1]\!]_\sigma, \ldots, v_n \in [\![T_n]\!]_\sigma, \\
& [\![P]\!]_{[\texttt{self}\mapsto o, x_1 \mapsto v_1, \ldots, x_n \mapsto v_n], \sigma, \sigma, I} = \mathit{true}\}
\end{aligned}$$

$$\begin{aligned}
\mathrm{post}^I_\Sigma(\varphi) = \{(\sigma, o.op(v_1, \ldots, v_n), \sigma, r) \mid{} & \sigma \in \mathit{State}_\Sigma, o \in [\![C]\!]_\sigma \setminus \{\mathit{null}\}, \\
& v_1 \in [\![T_1]\!]_\sigma, \ldots, v_n \in [\![T_n]\!]_\sigma, r \in [\![T_0]\!]_\sigma, \\
& [\![Q]\!]_{[\texttt{self}\mapsto o, x_1 \mapsto v_1, \ldots, x_n \mapsto v_n, \texttt{result}\mapsto r], \sigma, \sigma, I} = \mathit{true}\}
\end{aligned}$$

Table 1
Precondition and postcondition domains

`result`, and $x_1, \ldots, x_n$. The reserved variable `result` is used to refer to the result value of a method call. (This is in contrast to constructors where `self` denotes the newly constructed object and thus the result of a constructor call.)

The precondition domain of φ w.r.t. a query interpretation I defines a set of states and labels that satisfy the precondition P of φ as follows:

$$\begin{aligned}\mathrm{pre}^I_\Sigma(\varphi) = \{(\sigma, o.op(v_1, \ldots, v_n)) \mid {} & \sigma \in \mathit{State}_\Sigma, o \in [\![T]\!]_\sigma \setminus \{\mathit{null}\},\\ & v_1 \in [\![T_1]\!]_\sigma, \ldots, v_n \in [\![T_n]\!]_\sigma,\\ & [\![P]\!]_{[\mathtt{self} \mapsto o, x_1 \mapsto v_1, \ldots, x_n \mapsto v_n], \sigma, \sigma, I} = \mathit{true}\}\end{aligned}$$

In particular, the precondition domain only takes into account such values that indeed exist in the current state σ (where for the sake of simplicity the value null is assumed to exist in every state). For evaluating the precondition P the single state σ is used as prestate and poststate.

The postcondition domain of φ w.r.t. I likewise defines a set of pairs of states, labels, and results that satisfy the postcondition of φ:

$$\begin{aligned}\mathrm{post}^I_\Sigma(\varphi) = \{(\sigma, o.op(v_1, \ldots, v_n), \sigma', r) \mid {} & \sigma, \sigma' \in \mathit{State}_\Sigma, o \in [\![T]\!]_\sigma \setminus \{\mathit{null}\},\\ & v_1 \in [\![T_1]\!]_\sigma, \ldots, v_n \in [\![T_n]\!]_\sigma, r \in [\![T_0]\!]_{\sigma'},\\ & [\![Q]\!]_{[\mathtt{self} \mapsto o, x_1 \mapsto v_1, \ldots, x_n \mapsto v_n, \mathtt{result} \mapsto r], \sigma, \sigma', I} = \mathit{true}\}\end{aligned}$$

This definition and its companions in Table 1 do not cover all possible parameter kinds for operation specifications that could occur in the UML. We restrict ourselves to in-parameters and a single return-parameter. The treatment of in-out-parameters and out-parameters would call for reference types and locations; several return-parameters can be handled by the OCL tuple types.

3.3 Models of OCL Operation Specifications

A model of an OCL operation specification over a class signature Σ is a Σ-system that respects the contract view of the operation specification: The precondition domain using the Σ-system's query interpretation must not be empty, in order to ensure that the operation is *usable* from the client's point of view. The precondition domain must be included in the domain of the Σ-system's ltso, in order to ensure that when the operation is called within its precondition domain the operation is *realisable* from the implementor's point of view. Finally, all possible transitions in the Σ-system's ltso from the precondition domain must be covered by the postcondition domain, in order to ensure that the postcondition indeed has been established when the operation has been executed.

Let Σ be a class signature and let φ be an operation specification over Σ. A Σ-system (S, I) is a Σ-*model* of φ, written as $(S, I) \models_\Sigma \varphi$, if

(i) $\mathrm{pre}^I_\Sigma(\varphi) \neq \emptyset$;

(ii) $\mathrm{pre}^I_\Sigma(\varphi) \subseteq \mathrm{dom}(S)$;

(iii) $\{(\sigma, l, \sigma', r) \in S \mid (\sigma, l) \in \mathrm{pre}^I_\Sigma(\varphi)\} \subseteq \mathrm{post}^I_\Sigma(\varphi)$.

In accordance with general model theory we call an operation specification φ Σ-*satisfiable*, if φ has a Σ-model.

Note that an operation specification is not only unsatisfiable due to unsatisfiable pre- or postconditions, but also when the postcondition implicitly restricts the domain of the operation specification. Consider

$$\varphi = \texttt{context}\ C\texttt{::}op\texttt{()}\ \texttt{pre:}\ P\ \texttt{post:}\ Q\ .$$

The implicit precondition domain of the operation specification φ is given by the set $\{(\sigma, l) \mid \exists(\sigma', r) . (\sigma, l, \sigma', r) \in \mathrm{post}^I_\Sigma(\varphi)\}$. A necessary condition for φ to be satisfiable is that the precondition domain $\mathrm{pre}^I_\Sigma(\varphi)$ is a subset of the implicit precondition domain. For instance, taking up the account example in Fig. 1, the operation specification for withdraw indeed is not satisfiable due to the implicit precondition balance - a >= limit.

3.4 Semantics of OCL Operation Specifications

The class of Σ-models satisfying an operation specification, if non-empty, enjoys the property that there is a *maximal* model with respect to a given query interpretation. This distinguished model can be used to define the semantics of satisfiable operation specifications and the equivalence of operation specifications.

In the maximal model, the application domain of an operation meets its precondition domain and the application range includes all possible outcomes of the operation as defined by its postcondition domain, representing full non-determinism.

Lemma 3.1 *Let Σ be a class signature and let φ be a satisfiable operation specification over Σ, i.e. there exists a Σ-system (S, I) such that $(S, I) \models_\Sigma \varphi$. Then*

$$\overline{S} = \{(\sigma, l, \sigma', r) \in \mathrm{post}^I_\Sigma(\varphi) \mid (\sigma, l) \in \mathrm{pre}^I_\Sigma(\varphi)\}$$

is the unique Σ-ltso such that $(\overline{S}, I) \models_\Sigma \varphi$ and $(S', I) \sqsubseteq (\overline{S}, I)$ for all Σ-systems (S', I) with $(S', I) \models_\Sigma \varphi$.

Proof. First of all, note that $\emptyset \neq \mathrm{pre}^I_\Sigma(\varphi)$ by the satisfiability of φ. Furthermore, $\mathrm{pre}^I_\Sigma(\varphi) = \mathrm{dom}(\overline{S})$ since for every $(\sigma, l) \in \mathrm{pre}^I_\Sigma(\varphi)$ there are σ' and r such that $(\sigma, l, \sigma', r) \in \mathrm{post}^I_\Sigma(\varphi)$, again by the satisfiability of φ, and $\mathrm{dom}(\overline{S}) \subseteq \mathrm{pre}^I_\Sigma(\varphi)$ by definition of $\overline{S}$. In particular, $(\overline{S}, I) \models_\Sigma \varphi$.

In order to prove that $(\overline{S}, I)$ is maximal, let (S', I) be some Σ-system with $(S', I) \models_\Sigma \varphi$. Then $\mathrm{dom}(\overline{S}) = \mathrm{pre}^I_\Sigma(\varphi) \subseteq \mathrm{dom}(S')$. Furthermore, if $(\sigma, l) \in \mathrm{dom}(\overline{S})$, then $S'(\sigma, l) \subseteq \mathrm{post}^I_\Sigma(\varphi)(\sigma, l) = \overline{S}(\sigma, l)$. Thus $(S', I) \sqsubseteq (\overline{S}, I)$. The uniqueness of $\overline{S}$ follows from $\sqsubseteq$ being a partial order on Σ-systems. $\Box$

In fact, the construction of the lemma coincides with $\bigsqcup\{(S, I) \mid (S, I) \models_\Sigma \varphi\}$ with respect to a fixed query interpretation I. However, the Σ-satisfiability of φ is crucial, as the disjunction over the empty family does not constitute a Σ-model of φ.

For a given query interpretation I, we thus define the *semantics* $[\![\varphi]\!]^I_\Sigma$ of a Σ-satisfiable operation specification φ as the maximal Σ-model (S, I) with $(S, I) \models_\Sigma \varphi$. Two operation specifications φ_1, φ_2 are *equivalent* with respect to a query interpretation I, written as $\varphi_1 \simeq^I_\Sigma \varphi_2$, if both are Σ-satisfiable and $[\![\varphi_1]\!]^I_\Sigma = [\![\varphi_2]\!]^I_\Sigma$. Note that the precondition domains of equivalent, satisfiable operation specifications coincide by construction of the maximal model.

Provided that the operation specification φ is satisfiable, our definition of the semantics of an operation specification coincides with the semantics of the OCL 2.0 proposal [13] and Brucker and Wolff [3]. However, if φ is not satisfiable, our semantics of φ is undefined while the OCL 2.0 semantics is still defined. Consider again the two operation specifications from the introduction (Sect. 1):

$$\varphi_1 = \texttt{context}\ C\texttt{::}\mathit{op}\texttt{()}\ \texttt{pre:}\ P\ \texttt{post:}\ Q \quad \text{and}$$
$$\varphi_2 = \texttt{context}\ C\texttt{::}\mathit{op}\texttt{()}\ \texttt{pre:}\ \texttt{true}\ \texttt{post:}\ P\texttt{@pre}\ \texttt{and}\ Q$$

On the one hand, if φ_1 is satisfiable then $[\![\varphi_1]\!]^I_\Sigma$ is defined and yields the same relation between pre- and poststates as in the $P \wedge Q$ style. On the other hand, satisfiability of φ_1 does not imply satisfiability of φ_2 as, in general, condition (ii) in Sect. 3.3 cannot be fulfilled, thus making $[\![\varphi_2]\!]^I_\Sigma$ undefined.

3.5 Correct Realisations of OCL Operation Specifications

An implementor of a class signature Σ provides for each operation op in Σ an interpretation of op in terms of a state transition function (possibly with result) for a non-query method and a function on states for a query operation. This gives rise to the notion of a Σ-interpretation extending the notion of a query interpretation over Σ given in Sect. 2. To answer the question, when an implementor of a class signature Σ has correctly implemented an operation specifications φ, i.e. provided a correct Σ-interpretation, we first define the Σ-system (S_Z, I_Z) induced by a Σ-interpretation Z. Then Z is considered a correct realization of φ if (S_Z, I_Z) is a Σ-model for φ.

A Σ-interpretation Z of a class signature Σ is given by a query interpretation I_Z of the query operations together with a set of state transition functions for the operations and constructors in Σ. Let $op : C \times T_1 \times \cdots \times T_n \to T$ be an operation with result in Σ. An interpretation of op is a function:

$$op^Z_C : \mathit{State}_\Sigma \times [\![C]\!] \times [\![T_1]\!] \times \cdots \times [\![T_n]\!] \to (\mathit{State}_\Sigma \times [\![T]\!]) \cup \{\bot\}$$

such that for all $\sigma \in \mathit{State}_\Sigma$, $o \in [\![C]\!]$, $v_1 \in [\![T_1]\!], \ldots, v_n \in [\![T_n]\!]$:

$$op^Z_C(\sigma, o, v_1, \ldots, v_n) = \bot$$

if $(o, v_1, \ldots, v_n) \notin [\![C]\!]_\sigma \times [\![T_1]\!]_\sigma \times \cdots \times [\![T_n]\!]_\sigma$, $o = \mathit{null}$, or $op^Z_C(\sigma, o, v_1, \ldots, v_n) \notin [\![T]\!]_{\sigma'} \cup \{\bot\}$ otherwise. Similarly, an operation without a result $(op : C \times T_1 \times$

$\cdots \times T_n)$ and a constructor $C : T_1 \times \cdots \times T_n \to C$ are interpreted as functions

$$op_C^Z : State_\Sigma \times [\![C]\!] \times [\![T_1]\!] \times \cdots \times [\![T_n]\!] \to State_\Sigma \cup \{\bot\}$$

and

$$C_C^Z : State_\Sigma \times [\![T_1]\!] \times \cdots \times [\![T_n]\!] \to (State_\Sigma \times [\![C]\!]) \cup \{\bot\}\ ,$$

respectively. Non-termination and exceptions are modelled by the value $\bot$.

A Σ-interpretation Z gives rise to the Σ-system (S_Z, I_Z) defined by:

$$(\sigma, o.op(v_1, \ldots, v_n), \sigma', r) \in S_Z \text{ iff } op_C^Z(\sigma, o, v_1, \ldots, v_n) = (\sigma', r) \neq \bot$$

for all operation symbols with result from Σ and for all $\sigma, \sigma' \in State_\Sigma$, $v_1 \in [\![T_1]\!]_\sigma$, $\ldots$, $v_n \in [\![T_n]\!]_\sigma$, and $r \in [\![T]\!]_{\sigma'}$. The labels for operation symbols without result, constructors, and query operations are defined in a similar way (cf. Sect. 2 for query operations).

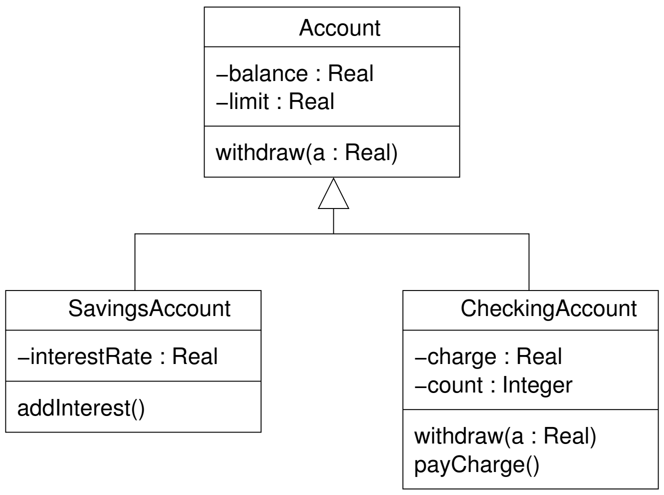
Note that there is a one-to-one correspondence between Σ-interpretations and deterministic Σ-systems. A deterministic Σ-system is a Σ-system where for each state σ all outgoing transitions have different labels.

A Σ-interpretation Z is a *correct realisation* of an operation specification φ if $(S_Z, I_Z) \models \varphi$ or, equivalently, if $(S_Z, I_Z) \sqsubseteq [\![\varphi]\!]_\Sigma^{I_Z}$ for the Σ-system (S_Z, I_Z) induced by Z.

Note that with our definition of a correct realisation, an implementation correctly realising an operation specification may be defined on more states and arguments than on the states and arguments for which the precondition holds. This is in accordance with the implementor's view of the contract principle and corresponds to the $P \Rightarrow Q$ style (as has been observed by Brucker and Wolff [3]). Hence our definition of a correct realisation differs from the corresponding definition in the OCL 2.0 proposal where the domain of a correct operation realisation has to coincide with the precondition domain of the operation specification.

4 Sets of OCL Operation Specifications and Inheritance

Operations may be specified not by a single OCL operation specification only, but by several operation specifications. The contract view can be extended to include sets of operation specifications for a single operation by the following reasoning: As every single operation specification forms a contract, the client may choose to fulfil the precondition of some operation specification out of the set in order to meet his duties and may expect the postcondition of the chosen contract. The supplier, however, is bound by all contracts simultaneously, i.e., when the operation is called in a state of the precondition domain of some contract, he has to establish all postconditions of the contracts whose preconditions the state satisfies. Inheritance of operations forms a special case of sets of operation specifications, if both the superclass and the subclass define an operation specification for an operation. The



```
context Account::withdraw(a : Real)
pre: balance-a >= limit
post: balance = balance@pre-a and balance >= limit

context CheckingAccount::withdraw(a : Real)
pre: balance-a >= limit
post: count = count@pre+1

context CheckingAccount::payCharge()
pre: balance-count*charge >= limit
post: balance = balance@pre-count*charge and count = 0
```

Fig. 2. UML/OCL specification for extended account example

operation specification of the superclass yields a contract for the operation in the subclass, too, since all instances of the subclass are also instances of the superclass.

Consider the extended example of an account specification in Fig. 2. When calling withdraw on an instance of CheckingAccount with an argument such that both, identical, preconditions are satisfied, both postconditions, changing the balance and increasing the withdraw counter have to be established. If we replaced the precondition of withdraw in CheckingAccount by `true`, the withdraw counter would have to be increased on every call of withdraw on an instance of CheckingAccount, independently of changing the balance. For calls of withdraw on instances of SavingsAccount only the contract inherited from Account is in force.

4.1 Models of Sets of OCL Operation Specifications

In line with the model-theoretic view of operation specifications, a set of operation specifications is considered as the conjunction of the single operation specifications. In particular, a set of operation specifications is considered satisfiable if all operation specifications in the set are satisfiable simultaneously.

Let Σ be a class signature, and let $\Phi = \{\varphi_1, \dots, \varphi_n\}$ be a set of operation specifications over Σ. A Σ-system (S, I) is a Σ-*model* of φ, written as $(S, I) \models_\Sigma \Phi$, if $(S, I) \models_\Sigma \varphi_i$ for all $1 \leq i \leq n$. Accordingly, Φ is called Σ-*satisfiable*, if Φ has a

Σ-model.

Necessarily, if a set of operation specifications is satisfiable, all elements of the set are satisfiable as there is a common model. The reverse direction, however, does not hold: Even if both

$$\varphi_1 = \texttt{context}\ C\texttt{::}\mathit{op}\texttt{()}\ \texttt{pre:}\ P\ \texttt{post:}\ Q \quad \text{and}$$
$$\varphi_2 = \texttt{context}\ C\texttt{::}\mathit{op}\texttt{()}\ \texttt{pre:}\ P\ \texttt{post:}\ \texttt{not}\ Q$$

are satisfiable, there is, in general, no common model for $\{\varphi_1, \varphi_2\}$ since there will be no postcondition state satisfying both Q and `not` Q.

4.2 Semantics of Sets of OCL Operation Specifications

As for a single satisfiable operation specification, a satisfiable set of operation specification has a maximal model that we will use as its semantics.

Lemma 4.1 *Let Σ be a class signature and let $\Phi = \{\varphi_1, \dots, \varphi_n\}$ be a set of operation specifications over Σ such that $(S, I) \models_\Sigma \Phi$ for a Σ-system (S, I). Then*

$$\begin{aligned}\overline{S} = \{(\sigma, l, \sigma', r) \mid (\sigma, l) \in {\textstyle\bigcup_{1 \le i \le n}} \mathrm{pre}^I_\Sigma(\varphi_i), \\ (\sigma', r) \in {\textstyle\bigcap} \{\mathrm{post}^I_\Sigma(\varphi_i)(\sigma, l) \mid (\sigma, l) \in \mathrm{pre}^I_\Sigma(\varphi_i), 1 \le i \le n\}\}\end{aligned}$$

is the unique Σ-ltso such that $(\overline{S}, I) \models_\Sigma \Phi$ and $(S', I) \sqsubseteq (\overline{S}, I)$ for all Σ-systems (S', I) with $(S', I) \models_\Sigma \Phi$.

Proof. We first show that $(\overline{S}, I) \models_\Sigma \varphi_i$ for all $1 \le i \le n$. Indeed, $\mathrm{pre}^I_\Sigma(\varphi_i) \neq \emptyset$ by the satisfiability of Φ and thus the satisfiability of φ_i. Furthermore, for every $(\sigma, l) \in \mathrm{pre}^I_\Sigma(\varphi_i)$ there is a σ' and an r such that $(\sigma', r) \in S(\sigma, l) \subseteq \mathrm{post}^I_\Sigma(\sigma, l)$. By the satisfiability of Φ, we have, in fact, $S(\sigma, l) \subseteq \bigcap_{j \in J_{(\sigma,l)}} \mathrm{post}^I_\Sigma(\varphi_j)(\sigma, l)$ with $J_{(\sigma,l)} = \{j \mid (\sigma, l) \in \mathrm{pre}^I_\Sigma(\varphi_j)\}$. Thus, $\bigcup_{1 \le i \le n} \mathrm{pre}^I_\Sigma(\varphi_i) = \mathrm{dom}(\overline{S})$. As $\bigcap_{j \in J_{(\sigma,l)}} \mathrm{post}^I_\Sigma(\varphi_j)(\sigma, l) = \overline{S}(\sigma, l)$ for all $(\sigma, l) \in \bigcup_{1 \le i \le n} \mathrm{pre}^I_\Sigma(\varphi_i)$, consequently $(\overline{S}, I) \models_\Sigma \Phi$.

With the same reasoning, but replacing (S, I) by an arbitrary Σ-system (S', I) with $(S', I) \models_\Sigma \Phi$, it follows that $\mathrm{dom}(\overline{S}) = \bigcup_{1 \le i \le n} \mathrm{pre}^I_\Sigma(\varphi) \subseteq \mathrm{dom}(S')$ and $S'(\sigma, l) \subseteq \bigcap_{j \in J_{(\sigma,l)}} \mathrm{post}^I_\Sigma(\varphi_j)(\sigma, l) = \overline{S}(\sigma, l)$ for all $(\sigma, l) \in \mathrm{dom}(\overline{S})$ and thus $(S', I) \sqsubseteq (\overline{S}, I)$. □

Extending the definition for a single operation specification, we define the *semantics* $[\![\Phi]\!]^I_\Sigma$ of a Σ-satisfiable set of operation specifications Φ with respect to a query interpretation I as the maximal Σ-model (S, I) with $(S, I) \models_\Sigma \Phi$. Two sets of operation specifications Φ_1, Φ_2 are *equivalent* with respect to a query interpretation I, written as $\Phi_1 \simeq^I_\Sigma \Phi_2$, if both are Σ-satisfiable and $[\![\Phi_1]\!]^I_\Sigma = [\![\Phi_2]\!]^I_\Sigma$.

By the construction of the lemma, the semantics of a Σ-satisfiable set of operation specifications is

$$[\![\{\varphi_1, \dots, \varphi_n\}]\!]^I_\Sigma = {\textstyle\bigsqcap_{1 \le i \le n}} [\![\varphi_i]\!]^I_\Sigma$$

for a fixed query interpretation I. The semantics thus induces a *normal form* φ for the Σ-satisfiable set of operation specifications $\{\varphi_1, \ldots, \varphi_n\}$ given by

$$\varphi_i = \texttt{context } C\texttt{::}op\texttt{() pre: } P_i \texttt{ post: } Q_i$$

such that $\{\varphi\} \simeq^I_\Sigma \{\varphi_1, \ldots, \varphi_n\}$: The preconditions are combined disjunctively, while the postconditions are combined conjunctively for those preconditions which are fulfilled simultaneously:

```
φ = context C::op()
    pre: P_1 or ... or P_n
    post: P_1@pre implies Q_1
          and ... and
          P_n@pre implies Q_n
```

For instance, in the extended account example in Fig. 2, the normal form for the operation withdraw in the context of CheckingAccount becomes:

```
context CheckingAccount::withdraw(a : Real)
pre: balance-a >= limit
post: (balance@pre-a >= limit@pre) implies
      (balance = balance@pre-a and balance >= limit and
       count = count@pre+1)
```

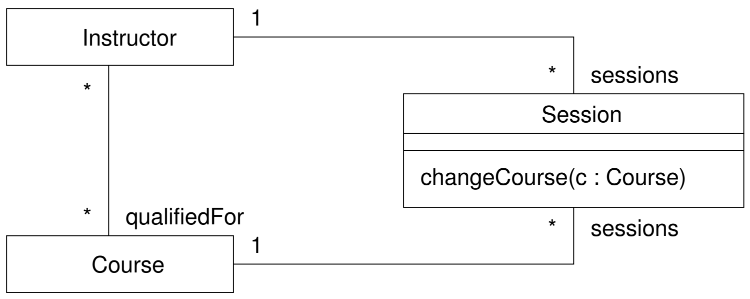
as the contract for withdraw in the context of Account is inherited to CheckingAccount.

5 Combining OCL Operation Specifications with Invariants

Class invariants constrain the possible states under which objects of a class can be viewed from other objects. These state constraints also influence the satisfaction of operation specifications, as both the preconditions and the postcondition are implicitly strengthened by the class invariants.

Consider the example of a seminar specification in Fig. 3, see also [7,8]. The invariant of Instructor requires an instructor to be qualified for all assigned courses. When calling changeCourse on a session that has already been assigned to an instructor but using a course which the instructor is not qualified for, the invariant would be violated, if only the course is changed. A realisation of changeCourse would not only have to modify its course but also the sessions of the instructor, removing this session. However, such a solution is hardly viable in the present system as the multiplicity 1 of the association end at Instructor requires each session to have an Instructor assigned. A more appropriate solution is to strengthen the precondition of changeCourse into `instructor.qualifiedFor->includes(c)`.

We take class invariants in a UML static structure diagram to extend the induced class signature Σ into a specification (Σ, A) with the class invariants as axioms A. This extension leads to a refined notion of precondition and postcondition domains, as only states have to be considered that satisfy all invariants in A. The notions



```
context Session::changeCourse(c : Course)
pre: true
post: course = c

context Instructor
inv: qualifiedFor->includesAll(sessions.course)
```

Fig. 3. UML/OCL specification for seminar example

of model and satisfiability defined for plain class signatures Σ can be transferred to specifications (Σ, A). All properties, like existence of a maximal model of a satisfiable operation specification or the combination of sets of operation properties can be replayed in the extended setting.

5.1 Semantics of OCL Operations Specifications with Invariants

An OCL invariant specification ψ over a class signature Σ takes the form

$$\texttt{context}\ C\ \texttt{inv:}\ V$$

where C is a class in Σ and V may contain only `self` as a free variable. We define the *invariant domain* of ψ with respect to a query interpretation I over Σ as

$$\mathrm{inv}^I_\Sigma(\psi) = \{\sigma \in \mathit{State}_\Sigma \mid \forall o \in [\![C]\!]_\sigma \setminus \{\mathit{null}\} \,.\, [\![V]\!]_{[\texttt{self}\mapsto o],\sigma,\sigma,I} = \mathit{true}\}\ .$$

This definition is extended to sets of invariant specifications A by $\mathrm{inv}^I_\Sigma(A) = \bigcap_{\psi\in A} \mathrm{inv}^I_\Sigma(\psi)$.

Given a class signature Σ, a set of invariant specifications A, and a query interpretation I over Σ, we define the precondition and the postcondition domain of an operation specification in the context of the invariant specifications A by

$$\mathrm{pre}^I_{(\Sigma,A)}(\varphi) = \mathrm{pre}^I_\Sigma(\varphi) \cap (\mathrm{inv}^I_\Sigma(A) \times \mathit{Label}_\Sigma)$$
$$\mathrm{post}^I_{(\Sigma,A)}(\varphi) = \mathrm{post}^I_\Sigma(\varphi) \cap (\mathrm{inv}^I_\Sigma(A) \times \mathit{Label}_\Sigma \times \mathrm{inv}^I_\Sigma(A) \times \mathit{Result}_\Sigma)$$

A Σ-system (S, I) is a (Σ, A)-*model* of φ, written as $(S, I) \models_{(\Sigma,A)} \varphi$, if

(i) $\mathrm{pre}^I_{(\Sigma,A)}(\varphi) \neq \emptyset$;

(ii) $\mathrm{pre}^I_{(\Sigma,A)}(\varphi) \subseteq \mathrm{dom}(S)$;

(iii) $\{(\sigma, l, \sigma', r) \in S \mid (\sigma, l) \in \mathrm{pre}^I_{(\Sigma,A)}(\varphi)\} \subseteq \mathrm{post}^I_{(\Sigma,A)}(\varphi)$.

An operation specification φ is (Σ, A)-*satisfiable*, if φ has a (Σ, A)-model.

All consequences from the notion of Σ-satisfiability remain valid for the extended notion of (Σ, A)-satisfiability. In particular, we define the *semantics* $[\![\varphi]\!]^{I}_{(\Sigma,A)}$ of a (Σ, A)-satisfiable operation specification φ with respect to a query interpretation I as the maximal Σ-system (S, I) with $(S, I) \models_{(\Sigma,A)} \varphi$.

5.2 Hierarchical Invariants

The effect of an invariant of some class on operation specifications requires to check that all operations, also of remote classes, respect all invariants. An alternative approach is discussed by Baumeister, Hennicker, Knapp, and Wirsing [8]: Navigating class invariants, i.e., class invariants using object properties that are not under exclusive control of the class are forbidden. Instead, navigating invariants have to be attached not to a class but to a higher-level container, called a component. Operations are classified into being private, component-private, and component-public. Only component-public operations are visible to the outside of the component, private operations are not visible outside their owning class. This classification and the lifting of navigating invariants to components simplifies the obligations for the different types of operations: Private operations need not respect any invariants, component-private operations have to respect the corresponding class invariants, while only component-public operations have to respect all class invariants and the component invariants. In the example above, the static structure would be enclosed in a subsystem Seminar lifting the invariant of Instructor to Seminar:

```
context Seminar
inv: Instructor.allInstances()->forAll(i : Instructor |
       i.qualifiedFor->includes(i.sessions.course))
```

If changeCourse is defined to be only component-private, it need not respect the component invariant of Seminar, as it can only be called inside the component.

This hierarchical approach to combining invariants with operation specifications can be modelled in the current setting by extending the signature of operations with private, component private, and component public tags for operations and splitting the invariants into class invariants and component invariants. The semantics of operation specifications is now defined by restricting the precondition and postcondition domains only to those invariants that have to be respected by the specific operation type.

6 Conclusions and Future Work

We have discussed the semantics of OCL operation specifications from the viewpoint of contracts interpreting an operation specification's precondition and postcondition as obligations for a client and a supplier. A model of an operation specification contract is a labelled transition system with output respecting the contract's obligations, an operation specification is satisfiable if it has a model. The semantics of a satisfiable operation specification is defined as the maximal model with respect to a general refinement relation. The refinement relation is also used to define correct realisations for operation specifications as deterministic labelled transition systems with output. The general model-theoretic treatment of operation specification entails a seamless integration of the semantics of sets of operation specifications, the semantics of operation-specification inheritance, and the combination of operation specifications with invariants.

The semantics of a single satisfiable operation specification coincides with the transition relation semantics defined in the OCL 2.0 proposal [13] and also by Brucker and Wolff [3]. However, pursuing the contract view, the notion of a model of a contract makes use of the explicit OCL precondition and leads to a more fine-grained distinction between operation specifications based on satisfiability. Moreover, the notion of labelled transition systems with output yields a global view on the totality of interacting and collaborating objects, whereas the transition relation semantics is geared towards a single operation. The implementation-oriented notion of correct realisations of operation specifications loosens the too restricted definition in the OCL 2.0 [13], allowing implementations to be defined on a wider domain than required by the precondition.

In our account of the semantics of OCL operation specifications, we have striven to be precise on at least the essential features and peculiarities of OCL. However, though the definitions distinguish between methods with and without result, constructors, and queries and respect queries as integral part of OCL, we have not included all different kinds of parameters, like in-out-parameters, and also have neglected feature overloading. Notwithstanding these omissions, an integration into a formal proof environment like HOL-OCL [3] may seem of interest. More importantly from a modelling perspective, we have not included a discussion of the proof obligations that result from operation specifications. Here, integration in the KeY environment [1] or the USE tool [16] remains future work.

Acknowledgement. We would like to thank Michel Bidoit for useful suggestions for improvements.

References

[1] Ahrendt, W., T. Baar, B. Beckert, M. Giese, R. Hähnle, W. Menzel, W. Mostowski and P. H. Schmitt, *The KeY System: Integrating Object-Oriented Design and Formal Methods*, in: M. Ojeda-Aciego, I. P. de Guzmán, G. Brewka and L. M. Pereira, editors, *Europ. Wsh. Logics in Artifical Intelligence*, Lect. Notes Artif. Intell. **1919** (2002), pp. 21–36.

[2] Bickford, M. and D. Guaspari, *Lightweight Analysis of UML*, Draft NAS1-20335/10, Odyssey Research Assoc. (1998), http://www.omg.org/cgi-bin/doc?ad/98-10-01.

[3] Brucker, A. D. and B. Wolff, *HOL-OCL: Experiences, Consequences and Design Choices*, in: J.-M. Jezequel, H. Hussmann and S. Cook, editors, *Proc. 5th Int. Conf. UML*, Lect. Notes Comp. Sci. **2460** (2002), pp. 196–210.

[4] Cengarle, M. V. and A. Knapp, *OCL 1.4/5 vs. 2.0 Expressions — Formal Semantics and Expressiveness*, Software and Systems Modelling (2003), to appear.

[5] Clark, T. and J. Warmer, editors, "Advances in Object Modelling with the OCL," Lect. Notes Comp. Sci. **2263**, Springer, Berlin, 2002.

[6] Derrick, J. and E. Boiten, "Refinement in Z and Object-Z — Foundations and Advanced Applications," Formal Approaches to Computing and Information Technology, Springer, London–&c., 2001.

[7] D'Souza, D. F. and A. C. Wills, "Object, Components, Frameworks with UML: The Catalysis Approach," Addison-Wesley, Reading, Mass., &c., 1998.

[8] Hennicker, R., H. Baumeister, A. Knapp and M. Wirsing, *Specifying Component Invariants with OCL*, in: K. Bauknecht, W. Brauer and T. Mück, editors, *Proc. GI/OCG-Jahrestagung*, books@ocg.at **157/I**, ÖGI (Austrian Computer Society), 2001, pp. 600–607.

[9] Hennicker, R., H. Hußmann and M. Bidoit, *On the Precise Meaning of OCL Constraints*, in: Clark and Warmer [5] pp. 70–85.

[10] Jones, C. B., "Systematic Software Construction Using VDM," Prentice Hall, Upper Saddle River, New Jersey, 1990.

[11] Lano, K., "Formal Object-Oriented Development," Formal Approaches to Computing and Information Technology, Springer, London–&c., 1995.

[12] Meyer, B., "Object-Oriented Software Construction," Prentice-Hall, Upper Saddle River, New Jersey, 1997.

[13] *Response to OMG RfP ad/00-09-03 "UML 2.0 OCL"*, 2nd revised submission, OMG (2003), `http://www.omg.org/cgi-bin/doc?ad/03-01-07`.

[14] Richters, M., "A Precise Approach to Validating UML Models and OCL Constraints," Ph.D. thesis, Universität Bremen (2001).

[15] Richters, M. and M. Gogolla, *A Semantics for OCL Pre- and Postconditions*, in: T. Clark and J. Warmer, editors, *Proc. UML'2000 Wsh. UML 2.0 — The Future of OCL*, York, 2000.

[16] Richters, M. and M. Gogolla, *Validating UML Models and OCL Constraints*, in: A. Evans, S. Kent and B. Selic, editors, *Proc. 3rd Int. Conf. UML*, Lect. Notes Comp. Sci. **1939** (2000), pp. 265–277.

[17] Richters, M. and M. Gogolla, *OCL — Syntax, Semantics and Tools*, in: Clark and Warmer [5] pp. 38–63.

[18] Schmitt, P. H., *A Model Theoretic Semantics of OCL*, in: B. Beckert, R. France, R. Hähnle and B. Jacobs, editors, *Proc. Wsh. Precise Modelling and Deduction for Object-Oriented Software Development*, Technical Report DII 07/01 (2001), pp. 43–57.