Glass-Box and Black-Box Views on Object-Oriented Specifications[*]

Michel Bidoit
Laboratoire Spécification et Vérification
CNRS & ENS de Cachan, France
bidoit@lsv.ens-cachan.fr

Rolf Hennicker, Alexander Knapp, Hubert Baumeister
Institut für Informatik
Ludwig-Maximilians-Universität München, Germany
{hennicke, knapp, baumeist}@pst.ifi.lmu.de

Abstract

We present a logical foundation for object-oriented specifications which supports a rigorous formal development of object-oriented systems. In this setting, we study two different views on a system, the implementor's view (glass-box view) and the user's view (black-box view) which both are founded on a model-theoretic semantics. We also discuss the hierarchical construction of specifications and realisations. Our approach is abstract in the sense that it can be instantiated by various concrete specification formalisms like OCL or JML.

1. Introduction

One of the main approaches to support formal object-oriented software development are assertion-based techniques which use invariants to describe properties of objects and pre-/postconditions to describe properties of operations. In this area, many formalisms have been developed which either are tailored to a particular programming language, like the assertion language of Eiffel [15] and the "Java Modeling Language" (JML [14]), or are programming language independent like Object-Z [9] and the "Object Constraint Language" (OCL [23]). There are also many kinds of semantic interpretations of invariants and pre-/postcondition constraints, for instance, axiomatic interpretations using Hoare-style triples [19, 17, 8, 1], translations into algebraic specifications [4, 6] or into (dynamic) first-order logic [3], coalgebraic interpretations [21], functional interpretations [12] and there are semantic approaches using labelled transition systems [10, 7, 2, 16].

[*] This research has been partially supported by the GLOWA-Danube project (07GWK04) sponsored by the German Federal Ministry of Education and Research, by the InOpSys project (WI 841/6-1) sponsored by Deutsche Forschungsgemeinschaft (DFG) and by the EC 5th Framework project AGILE: Architectures for Mobility (IST-2001-32747).

Even for a single formalism like OCL, two different styles of interpretations, in the following called the "implies"-style and the "and"-style (see also [7]), have been proposed. Given a pre-/postcondition constraint of the form C`::`op`(`$\ldots$`) pre:` P `post:` Q, the "implies"-style ([4, 20, 12]) 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 "and"-style ([7, 18]) 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.

The semantic variations point out that there is still no consensus on the meaning of the crucial specification constructs which is definitely needed to gain a larger acceptance of formal object-oriented methods in practice. Important issues that must be clarified concern, for instance, the following questions:

- In which states should an invariant hold?
- Can we consider invariants and pre-/postconditions in isolation or only in the context of each other?
- What is the meaning of a precondition? Is it a domain constraint for an operation or is it bound to a postcondition?
- When does an object-oriented specification satisfy a constraint, when is it satisfiable?

An Abstract Framework for Object-Oriented Specifications. In order to focus on the crucial aspects of object-oriented specifications we develop in this paper an abstract object-oriented framework which is independent of both, of a programming language and of a concrete specification formalism. The framework can be instantiated by any concrete language which provides some basic ingredients like the notions of a class with attributes together with some expression language for describing properties of states. The underlying idea of our framework is to provide a foundation of object-oriented specifications which is based on model theory and mathematical logic where, in particular, satisfiability is a standard notion. Satisfiability of a formula (logical sentence) φ means that there is a model, i.e. a structure for the chosen logic which satisfies φ. Hence we investigate in the context of object-oriented systems appropriate notions of sentence, model, and satisfaction: As sentences we use invariants and so-called bi-state sentences for specifying the behaviour of operations. Models are considered as abstract representations of non-deterministic object-oriented programs thus representing concrete realisations.

Formally, a model is defined as a non-deterministic labelled transition system (with output) which conforms to a given class signature. In particular, a model has to respect the domain constraints of the operations which means that whenever an operation is called in a state where the domain constraint of the operation is satisfied then there must be a transition leading to a well-defined successor state. Domain constraints represent preconditions in the sense of UML stereotyped ≪pre≫ constraints and the application requirements by Poetzsch-Heffter [19] and hence are not bound to a specific postcondition. An object-oriented specification $Sp = (\Sigma, Inv, Beh)$ consists of a class signature Σ, a set of invariants Inv and a set Beh of behaviour specifications for the operations. The semantics of Sp is given by the class of all glass-box Σ-models which satisfy the given invariants and behaviour specifications. Thus we use the loose semantics approach where the semantics of a specification consists of all its correct realisations. We also provide proof rules for verifying the correctness of a realisation which are variants of well-known strategies considered in the literature (like the invariant rule of Eiffel [15]). The soundness of these rules can now be formally justified in terms of our satisfaction relation.

Black-Box Views on Object-Oriented Specifications. Following the contract principle described by Meyer [15] there are two different views on an object-oriented specification, the implementor's and the user's point of view. The user has to ensure that the operations are only called when their domain constraints are satisfied. On the other hand, the implementor can assume that the user respects the domain constraints and must then ensure that the operations have the specified behaviour. In particular, a correct realisation may have an arbitrary behaviour if the domain constraints are not satisfied. The semantics of an object-oriented specification Sp considered so far has reflected all correct realisations of Sp, i.e. the implementor's point of view, and is therefore also called the glass-box semantics of Sp. To model the user's point of view we restrict the state space of a realisation to those states which are indeed reachable by correct operation calls of the user and we forget all transitions that a realisation performs outside the domain constraints of the operations (which anyway are irrelevant and cannot be observed by the user). In this way we obtain black-box models of object-oriented specifications. We show that the glass-box and the black-box views can be formally related by a satisfaction preserving function which maps glass-box to black-box models.

Hierarchical Object-Oriented Specifications. For developing large systems, specifications and realisations should be constructed in a modular way by composing single constituent parts. For this purpose we consider hierarchical object-oriented specifications $Sp_{\mathrm{H}} = (Sp, \hat{Sp})$ consisting of a subsystem specification Sp and a "body" specification $\hat{Sp}$. According to usual software engineering principles we require some basic properties like the independent realizability of the different constituent parts, the reusability of subsystem realisations and the preservation of the properties of local realisation pieces after their integration in the overall system. Taking into account these requirements we define the semantics of a hierarchical specification as a suitable class of functions which can be applied to any realisation M of a subsystem specification and yields a realisation $F(M)$ of the overall system (see e.g. [5]). Hence F plays the role of a user of M and therefore, for proving the correctness of F, it should be enough if F relies on the black-box views of the correct subsystem realisations M. We show that this is indeed the case if F is stable (in the sense of Schoett [22]) which means that F preserves the behavioural equivalence of models.

The paper is organised as follows: We start, in Sect. 2, summarising the necessary assumptions and considering the basic notion of state over a given attribute signature. In Sect. 3, we develop the crucial concepts of our abstract object-oriented framework. In particular, we introduce (domain-constrained) class signatures and their models and we define the satisfaction relation for models and sentences. In Sect. 4, object-oriented specifications and proof rules for their correct realisation are discussed. In Sect. 5, we focus on the black-box views on object-oriented specifications and we establish a satisfaction preserving relation between glass-box and black-box views. In Sect. 6 we consider hierarchical object-oriented specifications and we discuss how to make use of black-box views for proving the correctness of realisations of hierarchical specifications. Finally, some concluding remarks are given in Sect. 7.

We assume that the reader is familiar with the basic notions of algebraic specifications [25] and labelled transition systems [24]. In particular, we build on the notion of an *order-sorted signature* $\Sigma = (S, \leq, F)$ where S is a set of *sorts*, $\leq$ is a partial order on S, and F is a set of *function symbols* $f : s_1, \ldots, s_n \to s$. A signature $\Sigma = (S, \leq, F)$ is a *subsignature* of a signature $\Sigma' = (S', \leq', F')$, denoted by $\Sigma \subseteq \Sigma'$, if $S \subseteq S'$, $\leq \subseteq \leq'$, and $F \subseteq F'$. A *(total)* Σ*-algebra* $A = ((s^A)_{s\in S}, (f^A)_{f\in F})$ consists of a set s^A for each sort s with $s^A \subseteq t^A$ if $s \leq t$, and an interpretation function $f^A : s_1^A, \ldots, s_n^A \to s^A$ for each $f : s_1, \ldots, s_n \to s \in F$. For signatures $\Sigma = (S, \leq, F)$, $\Sigma' = (S', \leq', F')$ with $\Sigma \subseteq \Sigma'$, the Σ*-reduct* of a Σ'-algebra A is given by $A{\restriction}_\Sigma = ((s^A)_{s\in S}, (f^A)_{f\in F})$. If A is a Σ-algebra and $X = (x_i : s_i)_{i\in I}$ is an S-sorted set of variables, a *valuation* $\rho : X \to A$ assigns to each variable x of sort $s \in S$ a value in s^A.

2. Basic Framework for States

We fix a framework for describing states in object-oriented systems. A state is given by an algebra over a signature that captures the structural features of a class hierarchy. State formulae express properties in a single state or between two states.

An *attribute signature* Σ_A is a triple $(\langle T, C\rangle, \leq, \langle P, A\rangle)$ such that $(T \cup C, \leq, P \cup A)$ is an order-sorted signature. For the sorts, the set T contains (sorts for) predefined types (e.g. for booleans, integers, collections, etc.), the set C contains classes. For the function symbols, the set P contains the predefined operations (e.g. addition, multiplication, etc.), the set A contains the attributes of classes. Finally, the ordering $\leq$ represents the subtype relationship.

A Σ_A*-state* is a Σ_A-algebra σ with a fixed interpretation of the primitive types and the predefined operations. In particular, for each class type c the carrier set c^σ in a Σ_A-state σ represents the actually existing objects of c; we have $c^\sigma \subseteq d^\sigma$ if $c \leq d$ in Σ_A; and for each $o \in c^\sigma$ and an attribute $a : c \to t$ the actual attribute value of o is given by $a^\sigma(o)$. The class of Σ_A-states is denoted by $State_{\Sigma_\mathrm{A}}$.

Example (Counter). An attribute signature for a class Counter with an attribute val which determines the actual value of a Counter object is given by $\Sigma_\mathrm{A}^\mathrm{Cnt} = (\langle T, C\rangle, \leq, \langle F, A\rangle)$ with T at least containing the sort `Integer`, $C = \{\mathsf{Counter}\}$, $\leq$ the trivial partial ordering on $\{\mathtt{Integer}, \mathsf{Counter}\}$, F at least containing the usual operations on `Integer`, and $A = \{\mathsf{val} : \mathsf{Counter} \to \mathtt{Integer}\}$.

An actual state where, for instance, exactly one Counter object o exists with value 4 is given by the $\Sigma_\mathrm{A}^\mathrm{Cnt}$-algebra σ with carrier set $\mathsf{Counter}^\sigma = \{o\}$ and $\mathsf{val}^\sigma(o) = 4$.

We employ the "states-as-algebras" [11] approach which proves to be advantageous for building hierarchical object-oriented specifications, since it provides the notion of (state) reducts.

For an attribute signature Σ_A we assume given Σ_A-*sentences* which are either

1. *mono-state* Σ_A-sentences φ with associated set $\mathrm{var}(\varphi)$ of sorted variables; or
2. *bi-state* Σ_A-sentences π with associated sets $\mathrm{var}_\mathrm{in}(\pi)$ of sorted *input variables* and $\mathrm{var}_\mathrm{out}(\pi)$ of sorted *output variables*.

Σ_A-sentences are further assumed to be equipped with a *satisfaction relation* $\models_\mathrm{st}$ for states:

1. $\sigma, \rho \models_\mathrm{st} \varphi$ for a mono-state Σ_A-sentence φ, a Σ_A-state σ, and a valuation $\rho : X \to \sigma$ where $\mathrm{var}(\varphi) \subseteq X$;
2. $\sigma, \rho; \sigma', \rho' \models_\mathrm{st} \pi$ for a bi-state Σ_A-sentence π, Σ_A-states σ, σ', and valuations $\rho : X_\mathrm{in} \to \sigma$, $\rho' : X_\mathrm{out} \to \sigma'$ where $\mathrm{var}_\mathrm{in}(\pi) \subseteq X_\mathrm{in}$ and $\mathrm{var}_\mathrm{out}(\pi) \subseteq X_\mathrm{out}$.

For a mono-state sentence φ we write $\sigma \models_\mathrm{st} \varphi$ if $\sigma, \rho \models_\mathrm{st} \varphi$ for all valuations $\rho : X \to \sigma$ with $\mathrm{var}(\varphi) \subseteq X$.

Note that the satisfaction relation assumes valuations which are defined on supersets of the corresponding sets of the free (input or output) variables. This is needed to handle sentences which are bound to operations with input and output parameters in the next sections.

Example (OCL). Our basic framework for states can be instantiated, for instance, by the "Object Constraint Language" (OCL [23]).

Attribute signatures are directly derived from the OCL primitive and collection types and from UML class signatures (cf. [18, App. A] and [13]). UML class names are translated into sorts, attributes and role names at association ends are both translated into attributes of appropriate sorts. The OCL type hierarchy, which also reflects the inheritance relation between UML classes, is translated into the ordering relation of the attribute signature.

Mono-state sentences are simply the OCL expressions of type `Boolean` which do not contain the OCL constructs `@pre` and `oclIsNew()`. Bi-state sentences are arbitrary OCL expressions of type `Boolean`. The satisfaction relation for bi-state OCL sentences is defined by

$$\sigma, \rho; \sigma', \rho' \models_\mathrm{st} \pi \quad \text{iff} \quad [\![\pi]\!]_{\sigma,\rho,\sigma',\rho'} = \mathit{true}$$

where $[\![\_]\!]$ denotes the interpretation of OCL expressions defined in the mathematical semantics of OCL (see [18, App. A]). Similarly, the satisfaction relation for mono-state OCL sentences is defined by

$$\sigma, \rho \models_\mathrm{st} \varphi \quad \text{iff} \quad [\![\varphi]\!]_{\sigma,\rho,\sigma,\rho} = \mathit{true}\ .$$

Considering the example of the counter above, mono-state sentences are, for instance,

```
self.val >= 0 ,
Counter.allInstances()->isEmpty()
```

with a variable `self` of type Counter. A bi-state sentence is, e.g.,

```
self.val >= self.val@pre+1
```

where `self` is considered as an input variable.

In all subsequent examples sentences will be represented by OCL expressions.

3. Class Signatures and Glass-Box Models

Class signatures extend the basic framework for states by introducing operations. For each operation we assume given a so-called domain constraint which can be considered as an application requirement in the sense of Poetzsch-Heffter [19] or as a UML stereotyped ≪pre≫ constraint. A domain constraint imposes obligations on both the user and the implementor of an operation: The user is obliged to call the operation only in a state where the domain constraint of the operation is satisfied. The implementor guarantees that then the operation is enabled and leads to a valid successor state, i.e., the operation terminates and does not produce a runtime error.

An *operation* op has the form $name(X_{\mathrm{in}}, X_{\mathrm{out}})$ where X_{in} is a set of sorted input variables and X_{out} is a set of sorted output variables; the input variables X_{in} of op are denoted by $\mathrm{var}_{\mathrm{in}}(op)$, the output variables X_{out} of op by $\mathrm{var}_{\mathrm{out}}(op)$. A *method* for a class C is an operation whose input variables contain a special variable `self` : C of sort C. A *constructor* for a class C is an operation whose output variables contain a special variable `new` : C of sort C. A *domain constraint* dom_{op} of an operation is a mono-state Σ_{A}-sentence with $\mathrm{var}(dom_{op}) \subseteq \mathrm{var}_{\mathrm{in}}(op)$.

Definition. A *class signature* $\Sigma = (\Sigma_{\mathrm{A}}, Op, dom)$ consists of an attribute signature $\Sigma_{\mathrm{A}} = (S, \leq, A)$, a set Op of methods and constructors whose input and output variables have types in S, and a family $dom = (dom_{op})_{op \in Op}$ of domain constraints dom_{op} for each operation $op \in Op$.

Example (Counter). A class signature for the example Counter (see Sect. 2) is given by $\Sigma^{\mathrm{Cnt}} = (\Sigma_{\mathrm{A}}^{\mathrm{Cnt}}, Op, dom)$ where

$$\begin{gathered} Op = \{\mathsf{createCounter}(\mathrm{out}\ \texttt{new} : \mathsf{Counter}), \\ \mathsf{inc}(\mathrm{in}\ \texttt{self} : \mathsf{Counter}), \\ \mathsf{dec}(\mathrm{in}\ \texttt{self} : \mathsf{Counter})\} \\ dom_{\mathsf{createCounter}} = \\ \mathsf{Counter}\texttt{.allInstances()->isEmpty()} \\ dom_{\mathsf{inc}} = \texttt{true} \\ dom_{\mathsf{dec}} = \texttt{self.}\mathsf{val}\ \texttt{> 0} \end{gathered}$$

In particular, the domain constraint of createCounter realises the singleton pattern.

Let us now provide a model-theoretic interpretation of class signatures. In this section, we consider the implementor's point of view which we also call the *glass-box view*.

The crucial idea is that, given a class signature $\Sigma = (\Sigma_{\mathrm{A}}, Op, dom)$, a (glass-box) Σ-model provides an abstract representation of a non-deterministic program that conforms to Σ. For this purpose we use as an appropriate formalisation non-deterministic labelled transition systems with output where the state space is given by $Q \subseteq State_{\Sigma_{\mathrm{A}}}$, i.e., a set of Σ_{A}-algebras over the underlying attribute signature Σ_{A}. The labels, denoted by $Label_{\Sigma}$, express operation calls in a state $\sigma \in State_{\Sigma_{\mathrm{A}}}$, formally represented by pairs (op, ρ) consisting of the called operation op together with actual input parameters provided by a valuation $\rho : \mathrm{var}_{\mathrm{in}}(op) \to \sigma$. Similarly, the outputs, denoted by $Output_{\Sigma}$, represent the result of an operation call on an $op \in Op$ in a state $\sigma \in State_{\Sigma_{\mathrm{A}}}$ and are given by a valuation $\rho : \mathrm{var}_{\mathrm{out}}(op) \to \sigma$.

The essential part of a Σ-model is the transition relation

$$\Delta \subseteq Q \times Label_{\Sigma} \times (Q \times Output_{\Sigma})^{\perp}$$

where $(Q \times Output_{\Sigma})^{\perp}$ is defined by $(Q \times Output_{\Sigma}) \cup \{\perp\}$. A transition $(\sigma, (op, \rho), \sigma', \rho') \in \Delta$ with $\sigma' \in Q$ and $\rho \in Output_{\Sigma}$ models the fact that in state σ the operation call (op, ρ) is enabled and among the possible choices there is a valid successor state σ' with output ρ'. A transition $(\sigma, (op, \rho), \perp) \in \Delta$ models the fact that in state σ the operation call (op, ρ) is enabled but may diverge, i.e., may not terminate or lead to a runtime error. Generally, we assume that valuations ρ and ρ' occurring in a transition of Δ are well-defined w.r.t. pre- and post states:

(wdef1) For each $(\sigma, (op, \rho), \sigma', \rho') \in \Delta$ we have $\rho : \mathrm{var}_{\mathrm{in}}(op) \to \sigma$ and $\rho : \mathrm{var}_{\mathrm{out}}(op) \to \sigma'$.

(wdef2) For each $(\sigma, (op, \rho), \perp) \in \Delta$ we have $\rho : \mathrm{var}_{\mathrm{in}}(op) \to \sigma$.

Moreover, we assume that any (non-diverging) call of a constructor indeed creates a new object that did not exist in the previous state:

(cons) For each $(\sigma, (op, \rho), \sigma', \rho') \in \Delta$ with op a constructor for class c, we have $\rho'(\texttt{new} : c) \in c^{\sigma'} \setminus c^{\sigma}$.

Finally, we have to discuss the impact of the domain constraints $(dom_{op})_{op \in Op}$ on Σ-models. As pointed out above, the implementor must guarantee that whenever an operation call (op, ρ) occurs in a state σ with $\sigma, \rho \models_{\mathrm{st}} dom_{op}$ then the operation is enabled and does not diverge. It is important to note that it is enough if this property is satisfied for those states which are indeed reachable by arbitrary operation calls from an initial state which represents the start of any program execution. According to the contract principle it can be assumed that the user behaves correctly, i.e., in all operation calls leading to a reachable state the domain constraints of the corresponding operations have been satisfied. These states which in the following are considered as the relevant states are the *strongly reachable states* $\mathscr{R}^{dom}_{\sigma_0,\Delta}(Q)$ that are inductively defined as follows for a transition relation Δ over a class signature Σ as above and an initial state $\sigma_0 \in Q$:

1. $\sigma_0 \in \mathscr{R}^{dom}_{\sigma_0,\Delta}(Q)$.
2. If $\sigma \in \mathscr{R}^{dom}_{\sigma_0,\Delta}(Q)$ and $(\sigma, (op, \rho), \sigma', \rho') \in \Delta$ such that $\sigma, \rho \models_{\mathrm{st}} dom_{op}$, then $\sigma' \in \mathscr{R}^{dom}_{\sigma_0,\Delta}(Q)$.

The subset $\mathscr{R}^{dom}_{\sigma_0,\Delta}(\Delta) \subseteq \Delta$ of *domain-respecting transitions* on strongly reachable states is given by

$$\mathscr{R}^{dom}_{\sigma_0,\Delta}(\Delta) = \{(\sigma, (op, \rho), \sigma', \rho') \in \delta \mid \\ \sigma \in \mathscr{R}^{dom}_{\sigma_0,\Delta}(Q),\ \sigma, \rho \models_{\mathrm{st}} dom_{op}\} \ .$$

We can now formalise the additional property induced by the domain constraints which we require for Σ-models:

(dom1) For all $\sigma \in \mathscr{R}^{dom}_{\sigma_0,\Delta}(Q)$, $op \in Op$, and $\rho : \mathrm{var}_{\mathrm{in}}(op) \to \sigma$ with $\sigma, \rho \models_{\mathrm{st}} dom_{op}$, there are $\sigma' \in Q$ and $\rho' : \mathrm{var}_{\mathrm{out}}(op) \to \sigma'$ such that $(\sigma, (op, \rho), \sigma', \rho') \in \Delta$.

(dom2) If $(\sigma, (op, \rho), x) \in \Delta$ for $\sigma \in \mathscr{R}^{dom}_{\sigma_0,\Delta}(Q)$, $op \in Op$, and $\rho : \mathrm{var}_{\mathrm{in}}(op) \to \sigma$ with $\sigma, \rho \models_{\mathrm{st}} dom_{op}$ then $x \neq \bot$.

Definition. Let $\Sigma = (\Sigma_{\mathrm{A}}, Op, dom)$ be a class signature. A *glass-box* Σ*-model* is a triple (Q, σ_0, Δ) with $Q \subseteq State_{\Sigma_{\mathrm{A}}}$ such that $\sigma_0 \in Q$ and $\Delta \subseteq Q \times Label_{\Sigma} \times (Q \times Output_{\Sigma})^{\perp}$ which satisfy the conditions (wdef1), (wdef2), (cons), (dom1), and (dom2).

The class of glass-box Σ-models is denoted by $\mathrm{Mod}_{\mathrm{gb}}(\Sigma)$.

Example (Counter). The labelled transition system M^{Cnt} in Fig. 1 is a glass-box Σ^{Cnt}-model.

The strongly reachable states of M^{Cnt} are the initial state $\sigma^{(\emptyset)}$, states $\sigma^{(0)}$, $\sigma^{(1)}$, and all other states $\sigma^{(k)}$ with o`.val >= 0`. According to the conditions (dom1) and (dom2), the constructor call to createCounter must be enabled and terminate successfully in state $\sigma^{(\emptyset)}$, the method call o.inc must be enabled and terminate successfully in all states with o`.val >= 0` and the method call o.dec must be enabled and terminate successfully in all states with o`.val > 0`. Note, however, that conditions (dom1) and (dom2) say nothing about operation calls which occur in states where the domain constraint of the operation is not satisfied. In these situations the operation may also be enabled, like o.dec in $\sigma^{(0)}$ and o.inc in $\sigma^{(-1)}$, but then may lead to a state which is not strongly reachable like $\sigma^{(-1)}$, but only reachable.

Given a class signature $\Sigma = (\Sigma_{\mathrm{A}}, Op, dom)$, the set of Σ*-sentences* is defined as follows:

1. Each mono-state Σ_{A}-sentence φ is a (mono-state) Σ-sentence.
2. For each $op \in Op$ and for each bi-state Σ_{A}-sentence π with $\mathrm{var}_{\mathrm{in}}(\pi) \subseteq \mathrm{var}_{\mathrm{in}}(op)$ and $\mathrm{var}_{\mathrm{out}}(\pi) \subseteq \mathrm{var}_{\mathrm{out}}(op)$, $op : \pi$ is a (bi-state) Σ-sentence (bound by op).

The satisfaction relation $\models_{\mathrm{gb}}$ for (mono-state) Σ-sentences φ and (bi-state) Σ-sentences $op : \pi$ over a glass-box Σ-model $M = (Q, \sigma_0, \Delta)$ is given by

1. $M \models_{\mathrm{gb}} \varphi$ if for all $\sigma \in \mathscr{R}^{dom}_{\sigma_0,\Delta}(Q)$ and $\rho : \mathrm{var}(\varphi) \to \sigma$, we have $\sigma, \rho \models_{\mathrm{st}} \varphi$.
2. $M \models_{\mathrm{gb}} op : \pi$ if for all $(\sigma, (op, \rho), \sigma', \rho') \in \mathscr{R}^{dom}_{\sigma_0,\Delta}(\Delta)$, we have $\sigma, \rho; \sigma', \rho' \models_{\mathrm{st}} \pi$.

Note that in (2) $\sigma, \rho; \sigma', \rho' \models_{\mathrm{st}} \pi$ is well-defined (cf. Sect. 2), since $\rho : \mathrm{var}_{\mathrm{in}}(op) \to \sigma$, $\rho' : \mathrm{var}_{\mathrm{out}}(op) \to \sigma'$ and, by assumption on the form of Σ-sentences, $\mathrm{var}_{\mathrm{in}}(\pi) \subseteq \mathrm{var}_{\mathrm{in}}(op)$ and $\mathrm{var}_{\mathrm{out}}(\pi) \subseteq \mathrm{var}_{\mathrm{out}}(op)$.

The mono-state sentences correspond to global invariants, the bi-state sentences to operation specifications in the sense of Z [9]. The satisfaction for invariants is relaxed to hold in strongly reachable states only. As has been illustrated by the example of the counter above, requiring invariants to hold in all reachable, but not necessarily strongly reachable states would rule out meaningful implementations from the contract point of view. For instance, $M^{\mathrm{Cnt}} \models_{\mathrm{gb}}$ `self.val >= 0` which would not be the case if all reachable states including $\sigma^{(-1)}$ would be considered.

There is a strong similarity of the satisfaction relation $\models_{\mathrm{gb}}$ for glass-box models and the "implies"-style interpretation of pre-/postconditions discussed in the introduction: $M \models_{\mathrm{gb}} op : \pi$ if, and only if for all strongly reachable states σ and for all $(\sigma, (op, \rho), \sigma', \rho') \in \Delta$, it holds that $\sigma, \rho \models_{\mathrm{st}} dom_{op}$ implies $\sigma, \rho; \sigma', \rho' \models_{\mathrm{st}} \pi$.

4. Object-Oriented Specifications

We now have at disposition all ingredients that are needed to defining syntax and semantics of object-oriented specifications.

Definition. An *object-oriented specification* $Sp = (\Sigma, Inv, Beh)$ over Σ consists of a class signature Σ, a set Inv of mono-state Σ-sentences, called *invariants*, and a set Beh of bi-state Σ-sentences, called *behaviour specifications*.

The *glass-box semantics* of Sp is given by

$$[\![Sp]\!]_{\mathrm{gb}} = \{M \in \mathrm{Mod}_{\mathrm{gb}}(\Sigma) \mid \\ M \models_{\mathrm{gb}} Inv \text{ and } M \models_{\mathrm{gb}} Beh\} \ .$$

A glass-box Σ-model M is called a *correct realisation* of an object-oriented specification Sp over Σ if $M \in [\![Sp]\!]_{\mathrm{gb}}$. A specification Sp is *consistent*, if it has at least one correct realisation, i.e. $[\![Sp]\!]_{\mathrm{gb}} \neq \emptyset$.

Note that the definition of consistency above and of subclassing by subsorting can be extended to express also behavioural subtyping in the sense that for all redefined operations the domains are (at most) weakened and the behavioural specifications are (at worst) strengthened for subclasses. Moreover, the satisfaction relation for invariants

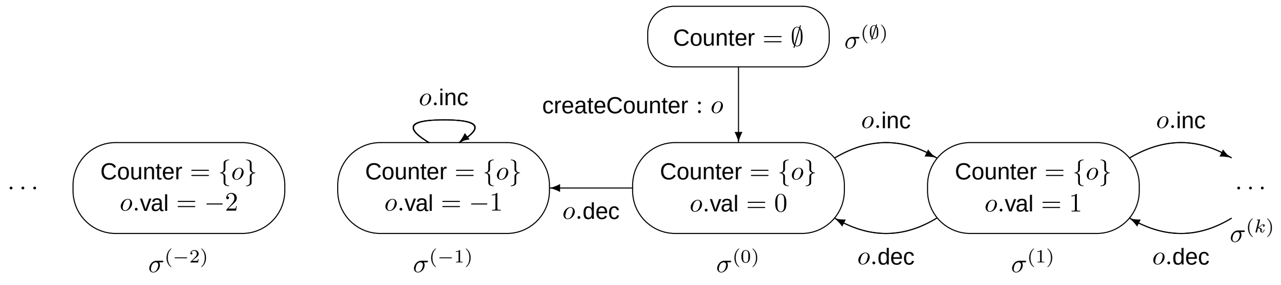

Figure 1: Glass-box Σ^{Cnt}-model M^{Cnt}

implies that invariants of superclasses are valid for all objects of subclasses.

Example (Counter). The following specification Sp^{Cnt} integrates the various pieces of the previous examples into a consistent specification of Counter objects and their associated operations:

```
Sp^Cnt = predefined elements +
  class Counter
    attributes
      val : Counter -> Integer

    constructors
      createCounter(out new : Counter)

    methods
      inc(in self : Counter)
      dec(in self : Counter)

    domains
      createCounter :
        Counter.allInstances()->isEmpty()
      inc : true
      dec : self.val > 0

    invariants
      self.val >= 0

    behaviours
      inc : self.val = self.val@pre+1
      dec : self.val = self.val@pre-1
```

Obviously, M^{Cnt} (cf. Sect. 3) is a correct realisation of Sp^{Cnt}. Note that for the consistency it is essential that the domain of the operation dec is properly constrained by `self.val > 0`. If, for instance, $dom_{\mathsf{dec}} =$ `true`, the specification would be inconsistent since in this case the behaviour specification of dec would be incompatible with the given invariant.

In order to prove that a glass-box Σ-model M is a correct realisation of an object-oriented specification $Sp = (\Sigma, Inv, Beh)$ it has to be shown that all invariants and all behaviour specifications of Sp are satisfied by M. For this purpose, the following two proof rules (inv) and (beh) for proving an invariant φ and a behaviour specification $op : \pi$, respectively, can be applied:

(inv) If $\sigma_0 \models_{\mathrm{st}} \varphi$, and
if ($\sigma, \rho \models_{\mathrm{st}} dom_{op}$ and $\sigma \models_{\mathrm{st}} \varphi$) implies $\sigma' \models_{\mathrm{st}} \varphi$
for all $op \in Op$, $(\sigma, (op, \rho), \sigma', \rho') \in \Delta$,
then $M \models_{\mathrm{gb}} \varphi$.

(beh) If $M \models_{\mathrm{gb}} \varphi$, and
if ($\sigma, \rho \models_{\mathrm{st}} dom_{op}$ and $\sigma \models_{\mathrm{st}} \varphi$) implies
$\sigma, \rho; \sigma', \rho' \models_{\mathrm{st}} \pi$
for all $(\sigma, (op, \rho), \sigma', \rho') \in \Delta$,
then $M \models_{\mathrm{gb}} op : \pi$.

Although the satisfaction of invariants and behaviour specifications has been defined independently of each other, it is methodologically desirable to intertwine proofs of invariants and behaviour specifications as done in rule (beh).

The rules (inv) and (beh) occur in several variations in the literature, see, for example, the invariant rule and the notion of class correctness by Meyer [15]. However, it is the first time, at least to our knowledge, that the soundness of these rules can be justified by an underlying satisfaction relation. Indeed, let us stress that both rules are only sound since our satisfaction relation has been relativised by taking into account only the strongly reachable states and the domain respecting transitions.

5. Black-Box Views on Object-Oriented Specifications

The notion of a glass-box Σ-model and the semantics of object-oriented specifications considered so far have reflected the implementor's point of view. In particular, a correct realisation may have an arbitrary behaviour if the domain constraints of the operations are not satisfied. However, according to the contract principle, we can assume that the user respects the given domain constraints of the operations. Then, the user can observe only strongly reachable states and domain respecting transitions. Thus, we can abstract from a given glass-box Σ-model $M = (Q, \sigma_0, \Delta)$ and obtain the so-called *black-box view* of M which is given by the transition system

$$\mathbf{Bb}_{\Sigma}(M) = (\mathscr{R}^{dom}_{\sigma_0,\Delta}(Q), \sigma_0, \mathscr{R}^{dom}_{\sigma_0,\Delta}(\Delta))\ .$$

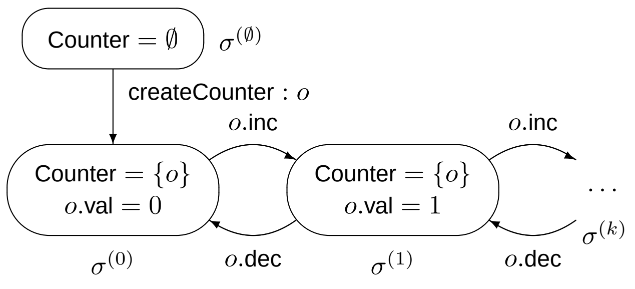

Figure 2: Black-box view of Σ^{Cnt}-model M^{Cnt}

Example (Counter). The black-box view of M^{Cnt} (cf. Sect. 3 and Fig. 1) yields the transition system $\mathbf{Bb}_{\Sigma^{\mathrm{Cnt}}}(M^{\mathrm{Cnt}})$ depicted in Fig. 2.

The black-box view of a glass-box Σ-model M has only (strongly) reachable states, all transitions respect the domain constraints and, moreover, an operation is always enabled if its domain constraint is satisfied. In particular, there is no diverging transition. These properties lead to the following definition of a black-box model.

Definition. Let $\Sigma = (\Sigma_{\mathrm{A}}, \mathit{Op}, \mathit{dom})$ be a class signature. A *black-box* Σ*-model* is a glass-box Σ model $M = (Q, \sigma_0, \Delta)$ such that

(bb1) Q is reachable w.r.t. σ_0 and Δ.

(bb2) For all $(\sigma, (\mathit{op}, \rho), \sigma', \rho') \in \Delta$ we have $\sigma, \rho \models_{\mathrm{st}} \mathit{dom}_{\mathit{op}}$.

(bb3) For all $\sigma \in Q$, $\mathit{op} \in \mathit{Op}$ and $\rho : \mathrm{var}_{\mathrm{in}}(\mathit{op}) \to \sigma$, with $\sigma, \rho \models_{\mathrm{st}} \mathit{dom}_{\mathit{op}}$, there exist $\sigma' \in Q$ and $\rho' : \mathrm{var}_{\mathrm{out}}(\mathit{op}) \to \sigma'$ such that $(\sigma, (\mathit{op}, \rho), \sigma', \rho') \in \Delta$.

The class of black-box Σ-models is denoted by $\mathrm{Mod}_{\mathrm{bb}}(\Sigma)$.

In fact, the black-box views of glass-box Σ-models are exactly the black-box Σ-models:

Proposition 1. *Let $\Sigma = (\Sigma_{\mathrm{A}}, \mathit{Op}, \mathit{dom})$ be a class signature. The definition of black-box views induces a surjective function $\mathbf{Bb}_\Sigma : \mathrm{Mod}_{\mathrm{gb}}(\Sigma) \to \mathrm{Mod}_{\mathrm{bb}}(\Sigma)$ which maps each $M \in \mathrm{Mod}_{\mathrm{gb}}(\Sigma)$ to its black-box view $\mathbf{Bb}_\Sigma(M)$.*

Proof. Let $M = (Q, \sigma_0, \Delta)$ be a glass-box Σ-model. We have to prove that $\mathbf{Bb}_\Sigma(M) = (\mathscr{R}^{\mathit{dom}}_{\sigma_0,\Delta}(Q), \sigma_0, \mathscr{R}^{\mathit{dom}}_{\sigma_0,\Delta}(\Delta))$ is a black-box Σ-model. Property (bb1) follows from the definition of $\mathscr{R}^{\mathit{dom}}_{\sigma_0,\Delta}(Q)$; property (bb2) follows from the definition of $\mathscr{R}^{\mathit{dom}}_{\sigma_0,\Delta}(\Delta)$; and property (bb3) follows from the definition of $\mathscr{R}^{\mathit{dom}}_{\sigma_0,\Delta}(Q)$ and (dom1).

For demonstrating that $\mathbf{Bb}_\Sigma : \mathrm{Mod}_{\mathrm{gb}}(\Sigma) \to \mathrm{Mod}_{\mathrm{bb}}(\Sigma)$ is surjective, let $B = (Q, \sigma_0, \Delta)$ be a black-box Σ-model. We show that $\mathbf{Bb}_\Sigma(B) = B$. Indeed, $Q = \mathscr{R}^{\mathit{dom}}_{\sigma_0,\Delta}(Q)$ by (bb1) and (bb2); and $\Delta = \mathscr{R}^{\mathit{dom}}_{\sigma_0,\Delta}(\Delta)$ by $Q = \mathscr{R}^{\mathit{dom}}_{\sigma_0,\Delta}(Q)$ and (bb2). □

Two glass-box Σ-models M and M' are called *behaviourally equivalent*, denoted by $M \equiv_{\mathrm{bb}} M'$ if $\mathbf{Bb}_\Sigma(M) = \mathbf{Bb}_\Sigma(M')$.

For black-box Σ-models $B = (Q, \sigma_0, \Delta)$ and (mono-state) Σ-sentences φ and (bi-state) Σ-sentences π we can define a simpler satisfaction relation $\models_{\mathrm{bb}}$ by

1. $M \models_{\mathrm{bb}} \varphi$ if for all $\sigma \in Q$ and $\rho : \mathrm{var}(\varphi) \to \sigma$, we have $\sigma, \rho \models_{\mathrm{st}} \varphi$.
2. $M \models_{\mathrm{bb}} \mathit{op} : \pi$ if for all $(\sigma, (\mathit{op}, \rho), \sigma', \rho') \in \Delta$, we have $\sigma, \rho; \sigma', \rho' \models_{\mathrm{st}} \pi$.

There is a strong similarity of the satisfaction relation $\models_{\mathrm{bb}}$ for black-box models and the "and"-style interpretation of pre-/postconditions discussed in the introduction: $M \models_{\mathrm{bb}} \mathit{op} : \pi$ if, and only if for all states σ and for all $(\sigma, (\mathit{op}, \rho), \sigma', \rho') \in \Delta$, it holds that $\sigma, \rho \models_{\mathrm{st}} \mathit{dom}_{\mathit{op}}$ and $\sigma, \rho; \sigma', \rho' \models_{\mathrm{st}} \pi$.

The black-box function $\mathbf{Bb}_\Sigma$ is compatible with the satisfaction relations for glass-box and black-box Σ-models.

Theorem 2. *Let $\Sigma = (\Sigma_{\mathrm{A}}, \mathit{Op}, \mathit{dom})$ be a class signature, $M \in \mathrm{Mod}_{\mathrm{gb}}(\Sigma)$ a glass-box Σ-model and ψ a Σ-sentence. Then $M \models_{\mathrm{gb}} \psi$ if, and only if $\mathbf{Bb}_\Sigma(M) \models_{\mathrm{bb}} \psi$.*

Proof. Let $M = (Q, \sigma_0, \Delta)$ be a glass-box Σ-model and let $B = (Q', \sigma_0, \Delta')$ be its black-box view. For mono-state Σ-sentences the claim follows from $Q' = \mathscr{R}^{\mathit{dom}}_{\sigma_0,\Delta}(Q)$ and the definition of the satisfaction relation $\models_{\mathrm{gb}}$ for mono-state sentences, for bi-state Σ-sentences the claim follows from $\Delta' = \mathscr{R}^{\mathit{dom}}_{\sigma_0,\Delta}(\Delta)$ and the definition of the satisfaction relation $\models_{\mathrm{gb}}$ for bi-state sentences. □

As a consequence of the theorem, we obtain that any behaviourally equivalent glass-box Σ-models $M \equiv_{\mathrm{bb}} M'$ satisfy the same Σ-sentences, i.e., for each Σ-sentence ψ, $M \models_{\mathrm{gb}} \psi$ if, and only if $M' \models_{\mathrm{gb}} \psi$.

Object-oriented specifications can be equipped with a black-box semantics which reflects the user's view on all correct realisations of a specification.

Definition. Let $\mathit{Sp} = (\Sigma, \mathit{Inv}, \mathit{Beh})$ be an object-oriented specification. The *black-box semantics* of Sp is given by

$$\llbracket \mathit{Sp} \rrbracket_{\mathrm{bb}} = \{M \in \mathrm{Mod}_{\mathrm{bb}}(\Sigma) \mid M \models_{\mathrm{bb}} \mathit{Inv} \text{ and } M \models_{\mathrm{bb}} \mathit{Beh}\} .$$

Obviously, the black-box semantics of an object-oriented specification Sp is just the image of the glass-box semantics of Sp (see Sect. 4) under the black-box function, i.e. $\llbracket \mathit{Sp} \rrbracket_{\mathrm{bb}} = \mathbf{Bb}_\Sigma(\llbracket \mathit{Sp} \rrbracket_{\mathrm{gb}})$.

6. Hierarchical Object-Oriented Specifications

When developing an object-oriented program we usually rely on several components which are plugged in a particular application. In this section, we extend our previous notions to take into account the hierarchical construction of object-oriented specifications and their correct realisations. Let us first summarise our basic requirements for modular system development which will later be reflected in the semantics of hierarchical specifications.

1. The various pieces of a hierarchical specification should be realisable independently of each other.
2. Any piece of software that is a correct realisation of a subsystem specification should be usable without modification for a correct realisation of the overall system specification.
3. Properties that are satisfied by a subsystem should be preserved if the subsystem is integrated into the overall system.

In order to express these requirements, we make use of the notion of subsignatures and reducts on order-sorted signatures and algebras (cf. Sect. 1): A class signature $\Sigma = (\Sigma_{\mathrm{A}}, \mathit{Op}, \mathit{dom})$ is a *subsignature* of a class signature $\hat{\Sigma} = (\hat{\Sigma_{\mathrm{A}}}, \hat{\mathit{Op}}, \hat{\mathit{dom}})$ if $\Sigma_{\mathrm{A}} \subseteq \hat{\Sigma_{\mathrm{A}}}$, $\mathit{Op} \subseteq \hat{\mathit{Op}}$, and $\mathit{dom}_{\mathit{op}} = \hat{\mathit{dom}}_{\mathit{op}}$ for all $\mathit{op} \in \mathit{Op}$. The Σ-*reduct* of a glass-box $\hat{\Sigma}$-model $\hat{M} = (\hat{Q}, \hat{\sigma_0}, \hat{\Delta})$ is given by the transition system $\hat{M}\restriction_{\Sigma} = (\hat{Q}\restriction_{\Sigma}, \hat{\sigma_0}\restriction_{\Sigma_{\mathrm{A}}}, \hat{\Delta}\restriction_{\Sigma})$ such that

$$\begin{aligned}
\hat{Q}\restriction_{\Sigma} &= \{\hat{\sigma}\restriction_{\Sigma_{\mathrm{A}}} \mid \hat{\sigma} \in \hat{Q}\} \\
\hat{\Delta}\restriction_{\Sigma} &= \{(\hat{\sigma}\restriction_{\Sigma_{\mathrm{A}}}, (\mathit{op}, \rho), \hat{\sigma}'\restriction_{\Sigma_{\mathrm{A}}}, \rho') \mid \\
&\qquad \mathit{op} \in \mathit{Op},\ (\hat{\sigma}, (\mathit{op}, \rho), \hat{\sigma}', \rho') \in \hat{\Delta}\} \cup \\
&\quad \{(\hat{\sigma}\restriction_{\Sigma_{\mathrm{A}}}, (\mathit{op}, \rho), \bot) \mid \\
&\qquad \mathit{op} \in \mathit{Op},\ (\hat{\sigma}, (\mathit{op}, \rho), \bot) \in \hat{\Delta}\}
\end{aligned}$$

where we use the same symbol ρ for the valuations $\rho : \mathrm{var}_{\mathrm{in}}(\mathit{op}) \to \hat{\sigma}$ and $\rho : \mathrm{var}_{\mathrm{in}}(\mathit{op}) \to \hat{\sigma}\restriction_{\Sigma_{\mathrm{A}}}$ and similarly for ρ'.

Definition. A *hierarchical object-oriented specification* $\mathit{Sp}_{\mathrm{H}} = (\mathit{Sp}, \hat{\mathit{Sp}})$ over $\Sigma \subseteq \hat{\Sigma}$ consists of a *subsystem* object-oriented specification $\mathit{Sp} = (\Sigma, \mathit{Inv}, \mathit{Beh})$ and a *body* object-oriented specification $\hat{\mathit{Sp}} = (\hat{\Sigma}, \hat{\mathit{Inv}}, \hat{\mathit{Beh}})$ such that Σ is a subsignature of $\hat{\Sigma}$.

Example (Counter). We extend the example Counter (see Sect. 3) by a user such that $\mathit{Sp}_{\mathrm{H}}^{\mathrm{Cnt}} = (\mathit{Sp}^{\mathrm{Cnt}}, \mathit{Sp}^{\mathrm{User}})$ forms a hierarchical object-oriented specification over $\Sigma^{\mathrm{Cnt}} \subseteq \Sigma^{\mathrm{User}}$:

$\mathit{Sp}^{\mathrm{User}} = \Sigma^{\mathrm{Cnt}} +$
```
class User
  attributes
    cnt : User -> Counter

  constructors
    createUser(in c : Counter, out new : User)

  domains
    createUser : c.val = 0

behaviours
  createUser : new.cnt = c
```

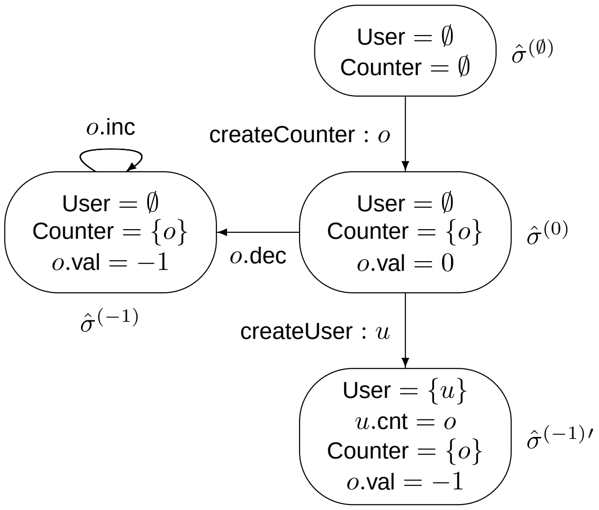

Figure 3: Extension of glass-box Σ^{Cnt}-model M^{Cnt} by a User

The semantics of a hierarchical object-oriented specification $\mathit{Sp}_{\mathrm{H}} = (\mathit{Sp}, \hat{\mathit{Sp}})$ over $\Sigma \subseteq \hat{\Sigma}$ should, as before, describe the class of all correct realisations of Sp_{H}. In accordance with the first requirement above, a correct realisation of Sp_{H} is taken to be a function F mapping any correct realisation M of Sp to a correct realisation of $\hat{\mathit{Sp}}$. Hence, once a correct realisation of Sp_{H} is provided, it will produce a glass-box $\hat{\Sigma}$-model with the desired properties independently of the particular choice of the correct realisation M of Sp. But note that this is not sufficient for satisfying our second requirement above, since it is not yet ensured that $F(M)$ preserves the model M. This means that indeed $F(M)$ should be an extension of M such that the reduct of $F(M)$ to the subsignature Σ yields again M, i.e. $F(M)\restriction_{\Sigma} = M$. Finally, the third requirement above means that for any Σ-sentence ψ we would like to have

(sat) If $M \models_{\mathrm{gb}} \psi$, then $F(M) \models_{\mathrm{gb}} \psi$.

However, the property (sat) is independent of the first two requirements, as illustrated by the following example:

Example. Let the realisation function for $\mathit{Sp}_{\mathrm{H}}^{\mathrm{Cnt}}$ be defined for the model M^{Cnt} of the counter example (see Sect. 3 and Fig. 1) such that $F(M^{\mathrm{Cnt}})$ contains the labelled transition system depicted in Fig. 3.

Obviously, $F(M^{\mathrm{Cnt}})\restriction_{\Sigma^{\mathrm{Cnt}}} = M^{\mathrm{Cnt}}$. But state $\hat{\sigma}^{(-1)\prime}$ does not satisfy the Counter invariant `self.val >= 0`. Since $\hat{\sigma}^{(-1)\prime}$ is a strongly reachable state of $F(M^{\mathrm{Cnt}})$, we have

$$M^{\mathrm{Cnt}} \models_{\mathrm{gb}} \texttt{self.val >= 0} \quad \text{but}$$
$$F(M^{\mathrm{Cnt}}) \not\models_{\mathrm{gb}} \texttt{self.val >= 0}$$

The reason for this undesired behaviour is that the construction of the Σ^{Cnt}-reduct of $F(M^{\mathrm{Cnt}})$ does not preserve strongly reachable states. In fact, $\hat{\sigma}^{(-1)\prime}\restriction_{\Sigma^{\mathrm{Cnt}}_{\mathrm{A}}} = \sigma^{(-1)}$ (cf. Fig. 1), but $\sigma^{(-1)}$ is not a strongly reachable state of M^{Cnt}.

Indeed, for ensuring (sat), it is obviously sufficient that a realisation F of a hierarchical object-oriented specification is constructed in such a way that reducts preserve strongly reachable states and also the domain respecting transitions on reachable states. Since the strongly reachable states and the domain respecting transitions are captured by the black-box views, this property is equivalent to the fact that reducts are compatible with black-box views.

In summary, the above considerations lead to the following definition of the semantics of hierarchical object-oriented specifications.

Definition. Let $Sp_{\mathrm{H}} = (Sp, \hat{Sp})$ be a hierarchical object-oriented specification over $\Sigma \subseteq \hat{\Sigma}$. The *semantics* of Sp_{H} is given by

$$\begin{aligned}\llbracket Sp_{\mathrm{H}} \rrbracket_{\mathrm{gb}} = \{F : \llbracket Sp \rrbracket_{\mathrm{gb}} \to \mathrm{Mod}_{\mathrm{gb}}(\hat{\Sigma}) \mid \forall M \in \llbracket Sp \rrbracket_{\mathrm{gb}} \,.\\ F(M) \in \llbracket \hat{Sp} \rrbracket_{\mathrm{gb}},\\ F(M)\restriction_{\Sigma} = M,\\ \mathbf{Bb}_{\hat{\Sigma}}(F(M))\restriction_{\Sigma} = \mathbf{Bb}_{\Sigma}(M)\} \,.\end{aligned}$$

A function $F : \llbracket Sp \rrbracket_{\mathrm{gb}} \to \mathrm{Mod}_{\mathrm{gb}}(\hat{\Sigma})$ is called a *correct realisation* of Sp_{H} if $F \in \llbracket Sp_{\mathrm{H}} \rrbracket_{\mathrm{gb}}$. In order to prove that an F is a correct realisation, one has to show that the three properties required for F in the definition of $\llbracket Sp_{\mathrm{H}} \rrbracket_{\mathrm{gb}}$ are satisfied. The last two properties can be easily ensured by good programming practice. In particular, the compatibility of reducts with black-box views is guaranteed if the program extension defined by F does not produce any state changes on objects of the subsystem M that could not be produced by domain respecting transitions of M itself. For instance, in the counter example above the call of createUser has changed the value of the counter object to -1 which would not be possible if only the Counter operations would have been called inside their domains. Such situations can be excluded if program extensions respect the following principles:

- Attributes defined in a subsystem are not accessed by the program extension (which is guaranteed anyway, if the attributes are private).
- Operation of a subsystem are only called when their domain constraints are satisfied (which is the obligation of the user anyway).

Hence, for proving that F is a correct realisation, it remains to check that $F(M)$ behaves well, i.e. $F(M) \in \llbracket \hat{Sp} \rrbracket_{\mathrm{gb}}$, whenever M behaves well, i.e. $M \in \llbracket Sp \rrbracket_{\mathrm{gb}}$. Because of the quantification over all possible M which satisfy Sp this is quite tedious to show. In this scenario, however, F can be considered as a user of M, who should not take care how the implementor actually did implement operations, if their domain constraints are not satisfied. Thus, for the user it should be enough to consider the black-box views of the realisations of Sp and check that for all $B \in \llbracket Sp \rrbracket_{\mathrm{bb}}$, $F(B) \in \llbracket \hat{Sp} \rrbracket_{\mathrm{gb}}$. If the subsystem specification Sp is sufficiently complete it may even be the case that there exists only one black-box model of Sp, but many glass-box models which all are behaviourally equivalent.

Indeed, this strategy works well if the function F enjoys the so-called stability property which was originally introduced by Schoett [22] and which we conjecture is satisfied by standard object-oriented programming languages like Java, if program extensions respect domain constraints of subsystem operations. In our context, a function $F : \llbracket Sp \rrbracket_{\mathrm{gb}} \to \mathrm{Mod}_{\mathrm{gb}}(\hat{Sp})$ is called *stable*, if for all $M, M' \in \llbracket Sp \rrbracket_{\mathrm{gb}}$ it holds that $M \equiv_{\mathrm{bb}} M'$ implies $F(M) \equiv_{\mathrm{bb}} F(M')$. This means that F preserves the behavioural equivalence of models. According to the definition of behavioural equivalence (see Sect. 5), it is obvious that stability can be equivalently expressed by the fact that for all $M \in \llbracket Sp \rrbracket_{\mathrm{gb}}$, $\mathbf{Bb}_{\hat{\Sigma}}(F(M)) = \mathbf{Bb}_{\hat{\Sigma}}(F(\mathbf{Bb}_{\Sigma}(M)))$. In summary, we obtain the following theorem:

Theorem 3. *Let $Sp_{\mathrm{H}} = (Sp, \hat{Sp})$ be a hierarchical object-oriented specification over $\Sigma \subseteq \hat{\Sigma}$ and let $F : \llbracket Sp \rrbracket_{\mathrm{gb}} \to \mathrm{Mod}_{\mathrm{gb}}(\hat{\Sigma})$ be stable such that for all $B \in \llbracket Sp \rrbracket_{\mathrm{bb}}$, $F(B) \in \llbracket \hat{Sp} \rrbracket_{\mathrm{gb}}$. Then for all $M \in \llbracket Sp \rrbracket_{\mathrm{gb}}$, $F(M) \in \llbracket \hat{Sp} \rrbracket_{\mathrm{gb}}$.*

Proof. Let $M \in \llbracket Sp \rrbracket_{\mathrm{gb}}$. Then, $\mathbf{Bb}_{\Sigma}(M) \in \llbracket Sp \rrbracket_{\mathrm{bb}}$ and thus, by assumption, $F(\mathbf{Bb}_{\Sigma}(M)) \in \llbracket \hat{Sp} \rrbracket_{\mathrm{gb}}$. Hence, $\mathbf{Bb}_{\hat{\Sigma}}(F(\mathbf{Bb}_{\Sigma}(M))) \in \llbracket \hat{Sp} \rrbracket_{\mathrm{bb}}$. Since F is stable, we have $\mathbf{Bb}_{\hat{\Sigma}}(F(\mathbf{Bb}_{\Sigma}(M))) = \mathbf{Bb}_{\hat{\Sigma}}(F(M)) \in \llbracket \hat{Sp} \rrbracket_{\mathrm{bb}}$. Thus $F(M) \in \llbracket \hat{Sp} \rrbracket_{\mathrm{gb}}$ by Theorem 2. □

7. Conclusions

We have proposed an abstract framework for object-oriented specifications which is based on model theory. Following the design by contract principle we have studied semantic notions for glass-box and black-box views on object-oriented specifications which express the implementor's and the user's point of view. Next steps are the investigation of refinement relations taking into account interfaces and their realisations by class specifications, the formalisation of object-oriented components and the development of proof calculi on top of our logical system.

References

[1] M. Abadi and K. R. M. Leino. A Logic of Object-Oriented Programs. In N. Dershowitz, editor, *Verification: Theory and Practice, Essays Dedicated to Zohar Manna on the Occasion of His 64th Birthday*, volume 2772 of *Lecture Notes in Computer Science*, pages 11–41. Springer, Berlin, 2003.

[2] E. Astesiano and G. Reggio. Towards a Well-Founded UML-Based Development Method. In *Proc. 1st Int. Conf. Software Engineering and Formal Methods (SEFM'03)*, pages 102–117. IEEE Computer Society, 2003.

[3] B. Beckert, U. Keller, and P. H. Schmitt. Translating the Object Constraint Language into First-order Predicate Logic. In S. Autexier and H. Mantel, editors, *Proc. Wsh. Verification (VERIFY'02)*, pages 113–123. Technical report, DIKU 2002/07, 2002.

[4] M. Bickford 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`.

[5] M. Bidoit and R. Hennicker. A General Framework for Modular Implementations of Modular System Specifications. In M.-C. Gaudel and J.-P. Jouannaud, editors, *Proc. Int. Conf. Theory and Practice of Software Development (TAPSOFT'93)*, volume 668 of *Lect. Notes Comp. Sci.*, pages 199–214. Springer, Berlin, 1993.

[6] M. Bidoit, R. Hennicker, F. Tort, and M. Wirsing. Correct Realizations of Interface Constraints with OCL. In R. France and B. Rumpe, editors, *Proc. 2nd Int. Conf. Unified Modeling Language (UML'99)*, volume 1723 of *Lect. Notes Comp. Sci.*, pages 399–415. Springer, Berlin, 1999.

[7] A. D. Brucker and B. Wolff. HOL-OCL: Experiences, Consequences and Design Choices. In J.-M. Jezequel, H. Hussmann, and S. Cook, editors, *Proc. 5th Int. Conf. Unified Modeling Language (UML'02)*, volume 2460 of *Lect. Notes Comp. Sci.*, pages 196–210. Springer, Berlin, 2002.

[8] F. S. de Boer and C. Pierik. Computer-Aided Specification and Verification of Annotated Object-Oriented Programs. In B. Jacobs and A. Rensink, editors, *Formal Methods for Open Object-Based Distributed Systems*, pages 163–177. Kluwer, Dordrecht, 2002.

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

[10] J. L. Fiadeiro and T. S. E. Maibaum. Temporal Theories as Modularisation Units for Concurrent System Specification. *Formal Aspects Comp.*, 4(3):239–272, 1992.

[11] H. Ganzinger. Programs as Transformations of Algebraic Theories (Extended Abstract). *Informatik Fachberichte*, 50:22–41, 1981.

[12] R. Hennicker, H. Hußmann, and M. Bidoit. On the Precise Meaning of OCL Constraints. In T. Clark and J. Warmer, editors, *Advances in Object Modelling with the OCL*, volume 2263 of *Lect. Notes Comp. Sci.*, pages 70–85. Springer, Berlin, 2002.

[13] R. Hennicker, A. Knapp, and H. Baumeister. Semantics of OCL Operation Specifications. In T. Baar, T. Clark, R. France, R. Hähnle, H. Hußmann, and P. H. Schmitt, editors, *Proc. Wsh. OCL 2.0 — Industry standard or scientific playground?*, San Francisco, 2003. 19 pages.

[14] G. T. Leavens, A. L. Baker, and C. Ruby. JML: A Notation for Detailed Design. In H. Kilov, B. Rumpe, and I. Simmonds, editors, *Behavioral Specifications of Businesses and Systems*, chapter 12, pages 175–188. Kluwer, Dordrecht, 1999.

[15] B. Meyer. *Object-Oriented Software Construction*. Prentice-Hall, Upper Saddle River, New Jersey, 2nd edition, 1997.

[16] M. Möller, E.-R. Olderog, H. Rasch, and H. Wehrheim. Linking CSP-OZ with UML and Java: A Case Study. In E. A. Boiten, J. Derrick, and G. Smith, editors, *Proc. 4th Int. Conf. Integrated Formal Methods (IFM'04)*, volume 2999 of *Lect. Notes Comp. Sci.*, pages 267–286. Springer, Berlin, 2004.

[17] P. Müller and A. Poetzsch-Heffter. Formal Specification Techniques for Object-Oriented Programs. In M. Jarke, K. Pasedach, and K. Pohl, editors, *Informatik 97: Informatik als Innovationsmotor*, Informatik Aktuell, pages 602–611. Springer, Berlin, 1997.

[18] 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`.

[19] A. Poetzsch-Heffter. *Specification and Verification of Object-Oriented Programs*. Habilitationsschrift, Technische Universität München, 1997.

[20] M. Richters 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.

[21] J. Rothe, H. Tews, and B. Jacobs. The Coalgebraic Class Specification Language CCSL. *J. Universal Computer Science*, 7(2):175–193, 2001.

[22] O. Schoett. *Data Abstraction and the Correctness of Modular Programming*. PhD thesis, University of Edinburgh, 1986.

[23] J. Warmer and A. Kleppe. *The Object Constraint Language*. Addison–Wesley, Reading, Mass., &c., 1999.

[24] G. Winskel and M. Nielsen. Models of Concurrency. In S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science. Vol. 4: Semantic Modelling*, pages 1–148. Oxford University Press, Oxford, 1995.

[25] M. Wirsing. Algebraic Specification. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Vol. B: Formal Models and Semantics*, pages 675–788. Elsevier, Amsterdam, 1990.