Institution-based Encoding and Verification of Simple UML State Machines in CASL/SPASS

Tobias Rosenberger[1,2], Saddek Bensalem[2],
Alexander Knapp[3], and Markus Roggenbach[1]

[1] Swansea University, U.K.
{t.rosenberger.971978, m.roggenbach}@swansea.ac.uk
[2] Université Grenoble Alpes, France
Saddek.Bensalem@imag.fr
[3] Universität Augsburg, Germany
knapp@informatik.uni-augsburg.de

Abstract. We present a new approach on how to provide institution-based semantics for UML state machines. Rather than capturing UML state machines directly as an institution, we build up a new logical framework $\mathcal{M}^{\downarrow}_{\mathcal{D}}$ into which UML state machines can be embedded. A theoroidal comorphism maps $\mathcal{M}^{\downarrow}_{\mathcal{D}}$ into the CASL institution. This allows for symbolic reasoning on UML state machines. By utilising the heterogeneous toolset HETS that supports CASL, a broad range of verification tools, including the automatic theorem prover SPASS, can be combined in the analysis of a single state machine.

1 Introduction

As part of a longstanding line of research [9,10,19,8], we set out on a general programme to bring together multi-view system specification with UML diagrams and heterogeneous specification and verification based on institution theory, giving the different system views both a joint semantics and richer tool support.

Institutions, a formal notion of a logic, are a principled way of creating such joint semantics. They make moderate assumptions about the data constituting a logic, give uniform notions of well-behaved translations between logics and, given a graph of such translations, automatically give rise to a joint institution.

In this paper, we will focus on UML state machines, which are an object-based variant of Harel statecharts. Within the UML, state machines are a central means to specify system behaviour. Here, we capture simple UML state machines in what we claim to be a true semantical sense. Focus of this paper are state machines running in isolation — interacting state machines and with it the notion of the event pool are left to future work.

Compared to our previous attempts to institutionalise state machines [9,10,19,8], this paper takes a different approach. Rather than capturing UML state machines directly as an institution, we build up a new logical framework $\mathcal{M}^{\downarrow}_{\mathcal{D}}$ in which UML state machines can be embedded. Core of this framework is a new hybrid modal logic which allows us to logically encode the *presence* as well as the *absence* of transitions in the state machines. Data types, guards, and effects of events are specified in the algebraic specification language CASL. An algorithm translates UML state machines into $\mathcal{M}^{\downarrow}_{\mathcal{D}}$.

A theoroidal comorphism maps our logical framework $\mathcal{M}^{\downarrow}_{\mathcal{D}}$ into the CASL institution. This allows to us to utilise the heterogeneous toolset HETS [15] and its connected provers for analysing UML state machines. In this paper we demonstrate how to analyse a state machine with the automatic first-order prover SPASS [20], which is the default automated prover of HETS. Such symbolic reasoning can be of advantage as, in principle, it allows to verify properties of UML state machines with large or infinite state spaces. Such machines appear routinely in system modelling: though state machines usually have only finitely many control states, they have a large number of configurations, or even infinitely many, due to the data variables involved.

Compared to other symbolic approaches to directly encode UML state machines into a specific interactive theorem prover [11,5,1], our logical framework $\mathcal{M}^{\downarrow}_{\mathcal{D}}$ provides first an institutional semantics that is tool independent. Only in a second step, we translate $\mathcal{M}^{\downarrow}_{\mathcal{D}}$ into CASL. Via HETS, this opens access to a broad range of analysis tools, including SAT solvers, automatic first-order theorem provers, automated and interactive higher-order theorem provers, which all can be combined in the analysis of state machines.

This paper is organised as follows: First we provide some background on institutions, including the CASL institution in Sect. 2. Then we discuss simple UML state machines, how to capture their events, attributes, and transitions, and what their models are. In Sect. 4 we define a new hybrid, modal logic for specifying UML state machine transitions. Section 5 provides the translation into the CASL institution. In Sect. 6, we finally demonstrate the symbolic analysis of a simple UML state machine as enabled by the previous constructions. We conclude in Sect. 7 with an outlook to future work. A version of this paper including full proofs has been published in [**?**].

2 Background on Institutions

We briefly recall the basic definitions of institutions and theoroidal institution comorphisms as well as the algebraic specification language CASL. Subsequently we will develop an institutional frame for capturing simple UML state machines and present a theoroidal institution comorphism from this frame into CASL.

2.1 Institutions and Theoroidal Institution Comorphisms

Institutions are an abstract formalisation of the notion of logical systems combining signatures, structures, sentences, and satisfaction under the slogan "truth is invariant under change of notation". Institutions can be related in different ways by institution (forward) (co-)morphisms, where a so-called theoroidal institution comorphism covers a particular case of encoding a "poorer" logic into a "richer" one.

Formally [4], an institution $\mathcal{I} = (\mathbb{S}^{\mathcal{I}}, \mathit{Str}^{\mathcal{I}}, \mathrm{Sen}^{\mathcal{I}}, \models^{\mathcal{I}})$ consists of (i) a category of *signatures* $\mathbb{S}^{\mathcal{I}}$; (ii) a contravariant *structures functor* $\mathit{Str}^{\mathcal{I}} \colon (\mathbb{S}^{\mathcal{I}})^{\mathrm{op}} \to \mathrm{Cat}$, where Cat is the category of (small) categories; (iii) a *sentence functor* $\mathrm{Sen}^{\mathcal{I}} \colon \mathbb{S}^{\mathcal{I}} \to \mathrm{Set}$, where Set is the category of sets; and (iv) a family of *satisfaction relations* $\models^{\mathcal{I}}_{\Sigma} \subseteq |\mathit{Str}^{\mathcal{I}}(\Sigma)| \times \mathrm{Sen}^{\mathcal{I}}(\Sigma)$ indexed over $\Sigma \in |\mathbb{S}^{\mathcal{I}}|$, such that the following *satisfaction condition* holds for all $\sigma \colon \Sigma \to \Sigma'$ in $\mathbb{S}^{\mathcal{I}}$, $\varphi \in \mathrm{Sen}^{\mathcal{I}}(\Sigma)$, and $M' \in |\mathit{Str}^{\mathcal{I}}(\Sigma')|$:

$$\mathit{Str}^{\mathcal{I}}(\sigma)(M') \models^{\mathcal{I}}_{\Sigma} \varphi \iff M' \models^{\mathcal{I}}_{\Sigma'} \mathrm{Sen}^{\mathcal{I}}(\sigma)(\varphi)\,.$$

spec NAT =
free type *Nat* ::= 0 | *suc*(*Nat*)
ops __+__ : *Nat* × *Nat* → *Nat*
pred __<__ : *Nat* × *Nat*
∀ *n*, *m* : *Nat* · 0 + *n* = *n* · *suc*(*n*) + *m* = *suc*(*n* + *m*)
· ¬ *n* < 0 · 0 < *suc*(*n*) · *suc*(*m*) < *suc*(*n*) ⇔ *m* < *n*
end

Fig. 1. A CASL specification of the natural numbers

$Str^{\mathcal{I}}(\sigma)$ is called the *reduct* functor, $\mathrm{Sen}^{\mathcal{I}}(\sigma)$ the *translation* function.

A *theory presentation* $T = (\Sigma, \Phi)$ in the institution $\mathcal{I}$ consists of a signature $\Sigma \in |\mathbb{S}^{\mathcal{I}}|$, also denoted by $Sig(T)$, and a set of sentences $\Phi \subseteq \mathrm{Sen}^{\mathcal{I}}(\Sigma)$. Its *model class* $\mathrm{Mod}^{\mathcal{I}}(T)$ is the class $\{M \in Str^{\mathcal{I}}(\Sigma) \mid M \models^{\mathcal{I}}_{\Sigma} \varphi \text{ f. a. } \varphi \in \Phi\}$ of the Σ-structures satisfying the sentences in Φ. A *theory presentation morphism* $\sigma\colon (\Sigma, \Phi) \to (\Sigma', \Phi')$ is given by a signature morphism $\sigma\colon \Sigma \to \Sigma'$ such that $M' \models^{\mathcal{I}}_{\Sigma'} \mathrm{Sen}^{\mathcal{I}}(\sigma)(\varphi)$ for all $\varphi \in \Phi$ and $M' \in \mathrm{Mod}^{\mathcal{I}}(\Sigma', \Phi')$. Theory presentations in $\mathcal{I}$ and their morphisms form the category $\mathrm{Pres}^{\mathcal{I}}$.

A *theoroidal institution comorphism* $\nu = (\nu^{\mathbb{S}}, \mu^{\mathrm{Mod}}, \nu^{\mathrm{Sen}})\colon \mathcal{I} \to \mathcal{I}'$ consists of a functor $\nu^{\mathbb{S}}\colon \mathbb{S}^{\mathcal{I}} \to \mathrm{Pres}^{\mathcal{I}'}$ inducing the functor $\nu^{Sig} = \nu^{\mathbb{S}}; Sig\colon \mathbb{S}^{\mathcal{I}} \to \mathbb{S}^{\mathcal{I}'}$ on signatures, a natural transformation $\nu^{\mathrm{Mod}}\colon (\nu^{\mathbb{S}})^{\mathrm{op}}; \mathrm{Mod}^{\mathcal{I}'} \dot{\to} Str^{\mathcal{I}}$ on structures, and a natural transformation $\nu^{\mathrm{Sen}}\colon \mathrm{Sen}^{\mathcal{I}} \dot{\to} \nu^{Sig}; \mathrm{Sen}^{\mathcal{I}'}$ on sentences, such that for all $\Sigma \in |\mathbb{S}^{\mathcal{I}}|$, $M' \in |\mathrm{Mod}^{\mathcal{I}'}(\nu^{\mathbb{S}}(\Sigma))|$, and $\varphi \in \mathrm{Sen}^{\mathcal{I}}(\Sigma)$ the following *satisfaction condition* holds:

$$\nu^{\mathrm{Mod}}_{\Sigma}(M') \models^{\mathcal{I}}_{\Sigma} \varphi \iff M' \models^{\mathcal{I}'}_{\nu^{Sig}(\Sigma)} \nu^{\mathrm{Sen}}(\Sigma)(\varphi)\,.$$

2.2 CASL and the Institution CFOL$^{=}$

The algebraic specification language CASL [16] offers several specification levels: *Basic specifications* essentially list signature declarations and axioms, thus determining a category of first-order structures. *Structured specifications* serve to combine such basic specifications into larger specifications in a hierarchical and modular fashion. Of the many logics available in CASL, we will work with the institution CFOL$^{=}$, of which we briefly recall the main notions; a detailed account can be found e.g. in [14].

At the level of basic specifications, cf. Fig. 1, one can declare *sort*s, *op*erations, and *pred*icates with given argument and result sorts. Formally, this defines a *many-sorted signature* $\Sigma = (S, F, P)$ with a set S of sorts, a $S^* \times S$-sorted family $F = (F_{w,s})_{w\,s \in S^+}$ of *total function symbols*, and a S^*-sorted family $P = (P_w)_{w \in S^*}$ of *predicate symbols*. Using these symbols, one may then write axioms in first-order logic. Moreover, one can specify data *type*s, given in terms of alternatives consisting of data constructors and, optionally, selectors, which may be declared to be *generated* or *free*. Generatedness amounts to an implicit higher-order induction axiom and intuitively states that all elements of the data types are reachable by constructor terms ("no junk"); freeness additionally requires that all these constructor terms are distinct ("no confusion"). Basic CASL specifications denote the class of all algebras which fulfil the declared axioms, i.e., CASL has loose semantics. In structured CASL specifications, a *structured free* construct

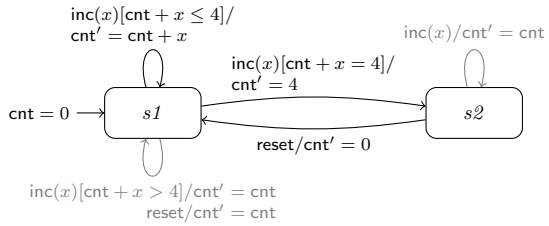


Fig. 2. Simple UML state machine $Counter$

can be used to ensure freeness (i.e., initial semantics) of a specification. For functions and predicates, the effect of the structured free construct corresponds to the effect of free types on sorts. A *many-sorted Σ-structure* M consists of a non-empty carrier set s^M for each $s \in S$, a total function $f^M : M_w \to M_s$ for each function symbol $f \in F_{w,s}$ and a predicate p^M for each predicate symbol $p \in P_w$. A *many-sorted Σ-sentence* is a closed many-sorted first-order formula over Σ or a sort generation constraint.

3 Simple UML State Machines

UML state machines [17] provide means to specify the reactive behaviour of objects or component instances. These entities hold an internal data state, typically given by a set of attributes or properties, and shall react to event occurrences by firing different transitions in different control states. Such transitions may have a guard depending on event arguments and the internal state and may change, as an effect, the internal control and data state of the entity as well as raise events on their own.

Figure 2 shows the example of a bounded, resettable counter working on an attribute cnt (assumed to take values in the natural numbers) that is initialised with 0. The counter can be reset to 0 or increased by a natural number x, subject to the current control state ($s1$ or $s2$) and the guards (shown in square brackets) and effects (after the slash) of the outgoing transitions. An effect describes how the data state before firing a transition (referred to by unprimed attribute names) relates to the data state after (primed names) in a single predicate; this generalises the more usual sequences of assignments such that $\mathsf{cnt}' = \mathsf{cnt} + x$ corresponds to $\mathsf{cnt} \leftarrow \mathsf{cnt} + x$ and $\mathsf{cnt}' = \mathsf{cnt}$ to a skip. The machine is specified non-deterministically: If event $\mathsf{inc}(x)$ occurs in state $s1$ such that the guard $\mathsf{cnt} + x = 4$ holds, the machine can either stay in $s1$ or it can proceed to $s2$. Seemingly, the machine does not react to reset in $s1$ and to inc in $s2$. However, UML state machines are meant to be input-enabled such that all event occurrences to which the machine shows no explicit reacting transition are silently discarded, as indicated by the "grey" transitions. Overall, the machine $Counter$ shall ensure that cnt never exceeds 4.

It is for such simple UML state machines as the counter in Fig. 2 that we want to provide proof support in SPASS via an institutional encoding in CASL. The sub-language covers the following fundamental state machine features: data, states, and (non-deterministic) guarded transitions for reacting to events. However, for the time being, we leave out not only all advanced modelling constructs, like hierarchical states or compound transitions, but also defer, most importantly, event-based communication

between state machines to future work. In the following we make first precise the syntax of the machines by means of event/data signatures, data states and transitions, guards and effects. Then we introduce semantic structures for the machines and define their model class. Syntax and semantics of simple UML state machines form the basis for their institutionalisation. We thus also introduce event/data signature morphisms and the corresponding formulæ translation and structure reducts in order to be able to change the interface of simple UML state machines.

3.1 Event/Data Signatures, Data States and Transitions

We capture the events for a machine in an *event signature* E that consists of a finite set of events $|E|$ and a map $\upsilon(E)$ assigning to each $e \in |E|$ a finite set of variables, where we write $e(X)$ for $e \in |E|$ and $\upsilon(E)(e) = X$, and also $e(X) \in E$ in this case. For the data state, we use a *data signature* A consisting of a finite set of attributes. An *event/data signature* Σ consists of an event signature $E(\Sigma)$ and a data signature $A(\Sigma)$.

Example 1. The event/data signature Σ of the simple UML state machine in Fig. 2 is given by the set of events $|E(\Sigma)| = \{\mathsf{inc}, \mathsf{reset}\}$ with argument variables $\upsilon(E(\Sigma))(\mathsf{inc}) = \{x\}$ and $\upsilon(E(\Sigma))(\mathsf{reset}) = \emptyset$ such that $\mathsf{inc}(x) \in E(\Sigma)$ and $\mathsf{reset} \in E(\Sigma)$; as well as the data signature $A(\Sigma) = \{\mathsf{cnt}\}$.

For specifying transition guards and effects, we exchange UML's notorious and intricate expression and action languages both syntactically and semantically by a straightforward CASL fragment rendering guards as data state predicates and effects as data transition predicates: We assume given a fixed universe $\mathcal{D}$ of *data values* and a CASL specification Dt with a dedicated sort dt in its signature $Sig(Dt)$ such that the universe dt^M of every model $M \in \mathrm{Mod}^{\text{CASL}}(Dt)$ is isomorphic to $\mathcal{D}$, i.e., there is a bijection $\iota_{M,dt} \colon dt^M \cong \mathcal{D}$. This puts at our disposal the open formulæ $\mathscr{F}^{\text{CASL}}_{Sig(Dt),X}$ over sorted variables $X = (X_s)_{s \in S}$ and their satisfaction relation $M, \beta \models^{\text{CASL}}_{Sig(Dt),X} \varphi$ for models $M \in \mathrm{Mod}^{\text{CASL}}(Dt)$, variable valuations $\beta \colon X \to M$, and formulæ $\varphi \in \mathscr{F}^{\text{CASL}}_{Sig(Dt),X}$.

Example 2. Consider the natural numbers $\mathbb{N}$ as data values $\mathcal{D}$. The CASL specification in Fig. 1 characterises $\mathbb{N}$ up to isomorphism as the carrier set of the dedicated sort $dt = \mathit{Nat}$. It specifies an abstract data type with sort *Nat*, operations $+, 0, \mathit{suc}$, and a predicate $<$.

The very simple choice of $\mathcal{D}$ capturing data with only a single sort can, in principal, be replaced by any institutional data modelling language that, for our purposes of a theoroidal institution comorphism (see Sect. 5), is faithfully representable in CASL; one such possibility are UML class diagrams, see [7].

Data states and guards. A *data state* ω for a data signature A is given by a function $\omega \colon A \to \mathcal{D}$; in particular, $\Omega(A) = \mathcal{D}^A$ is the set of A-data states. The guards of a machine are *state predicates* in $\mathscr{F}^{\mathcal{D}}_{A,X} = \mathscr{F}^{\text{CASL}}_{Sig(Dt),A \cup X}$, taking A as well as an additional set X as variables of sort dt. A state predicate $\phi \in \mathscr{F}^{\mathcal{D}}_{A,X}$ is to be interpreted over an A-data state ω and valuation $\beta \colon X \to \mathcal{D}$ and we define the *satisfaction relation* $\models^{\mathcal{D}}$ by

$$\omega, \beta \models^{\mathcal{D}}_{A,X} \phi \iff M, \iota^{-1}_{M,dt} \circ (\omega \cup \beta) \models^{\text{CASL}}_{Sig(Dt),A \cup X} \phi$$

where $M \in \mathrm{Mod}^{\text{CASL}}(Dt)$ and $\iota_{M,dt} \colon M(dt) \cong \mathcal{D}$. For a state predicate $\varphi \in \mathscr{F}^{\mathcal{D}}_{A,\emptyset}$ not involving any variables, we write $\omega \models^{\mathcal{D}}_{A} \varphi$ for $\omega \models^{\mathcal{D}}_{A,\emptyset} \varphi$.

Example 3. The guard $\mathsf{cnt} + x \leq 4$ of the machine in Fig. 2 features both the attribute cnt and the variable x. A data state fulfilling this state predicate for $x = 0$ is $\mathsf{cnt} \mapsto 3$.

Data transitions and effects. A *data transition* (ω, ω') for a data signature A is a pair of A-data states; in particular, $\Omega^2(A) = (\mathcal{D}^A)^2$ is the set of A-data transitions. It holds that $(\mathcal{D}^A)^2 \cong \mathcal{D}^{2A}$, where $2A = A \uplus A$ and we assume that no attribute in A ends in a prime $\prime$ and all attributes in the second summand are adorned with an additional prime. The effects of a machine are *transition predicates* in $\mathscr{F}^{2\mathcal{D}}_{A,X} = \mathscr{F}^{\mathcal{D}}_{2A,X}$. The satisfaction relation $\models^{2\mathcal{D}}$ for a transition predicate $\psi \in \mathscr{F}^{2\mathcal{D}}_{A,X}$, data transition $(\omega, \omega') \in \Omega^2(A)$, and valuation $\beta \colon X \to \mathcal{D}$ is defined as

$$(\omega, \omega'), \beta \models^{2\mathcal{D}}_{A,X} \psi \iff \omega + \omega', \beta \models^{\mathcal{D}}_{2A,X} \psi$$

where $\omega + \omega' \in \Omega(2A)$ with $(\omega + \omega')(a) = \omega(a)$ and $(\omega + \omega')(a') = \omega'(a)$.

Example 4. The effect $\mathsf{cnt}' = \mathsf{cnt} + x$ of the machine in Fig. 2 describes the increment of the value of attribute cnt by a variable amount x.

3.2 Syntax of Simple UML State Machines

A simple UML state machine U uses an event/data signature $\Sigma(U)$ for its events and attributes and consists of a finite set of *control states* $C(U)$, a finite set of *transition specifications* $T(U)$ of the form $(c, \phi, e(X), \psi, c')$ with $c, c' \in C(U)$, $e(X) \in E(\Sigma(U))$, a state predicate $\phi \in \mathscr{F}^{\mathcal{D}}_{A(\Sigma(U)),X}$, a transition predicate $\psi \in \mathscr{F}^{2\mathcal{D}}_{A(\Sigma(U)),X}$, an *initial control state* $c_0(U) \in C(U)$, and an *initial state predicate* $\varphi_0(U) \in \mathscr{F}^{\mathcal{D}}_{A(\Sigma(U)),\emptyset}$, such that $C(U)$ is *syntactically reachable*, i.e., for every $c \in C(U) \setminus \{c_0(U)\}$ there are $(c_0(U), \phi_1, e_1(X_1), \psi_1, c_1), \ldots, (c_{n-1}, \phi_n, e_n(X_n), \psi_n, c_n) \in T(U)$ with $n > 0$ such that $c_n = c$. Syntactic reachability guarantees initially connected state machine graphs. This simplifies graph-based algorithms (see Alg. 1).

Example 5. The machine in Fig. 2 has as its control states $\{s1, s2\}$, as its transition specifications $\{(s1, \mathsf{cnt} + x \leq 4, \mathsf{inc}(x), \mathsf{cnt}' = \mathsf{cnt} + x, s1), (s1, \mathsf{cnt} + x = 4, \mathsf{inc}(x), \mathsf{cnt}' = 4, s2), (s2, \mathrm{true}, \mathsf{reset}, \mathsf{cnt}' = 0, s1)\}$, as initial control state $s1$, and as initial state predicate $\mathsf{cnt} = 0$.

3.3 Event/Data Structures and Models of Simple UML State Machines

For capturing machines semantically, we use event/data structures that are given over an event/data signature Σ and consist of a transition system of configurations such that all configurations are reachable from its initial configurations. Herein, configurations show a control state, corresponding to machine states, and a data name from which a proper data state over $A(\Sigma)$ can be retrieved by a labelling function. Transitions connect configurations by events from $E(\Sigma)$ with their arguments instantiated by data from $\mathcal{D}$.

Formally, a Σ-*event/data structure* $M = (\Gamma, R, \Gamma_0, \omega)$ over an event/data signature Σ consists of a set of *configurations* $\Gamma \subseteq C \times D$ for some sets of *control states* C and *data names* D, a family of *transition relations* $R = (R_{e(\beta)} \subseteq \Gamma \times \Gamma)_{e(X) \in E(\Sigma), \beta\colon X \to \mathcal{D}}$, and a non-empty set of *initial configurations* $\Gamma_0 = \{c_0\} \times D_0 \subseteq \Gamma$ with a unique *initial control state* $c_0 \in C$ such that Γ is *reachable* via R, i.e., for all $\gamma \in \Gamma$ there are $\gamma_0 \in \Gamma_0$, $n \geq 0$, $e_1(X_1), \ldots, e_n(X_n) \in E(\Sigma)$, $\beta_1\colon X_1 \to \mathcal{D}, \ldots, \beta_n\colon X_n \to \mathcal{D}$, and $(\gamma_i, \gamma_{i+1}) \in R_{e_{i+1}(\beta_{i+1})}$ for all $0 \leq i < n$ with $\gamma_n = \gamma$; and a *data state labelling* $\omega\colon D \to \Omega(A(\Sigma))$. We write $c(M)(\gamma) = c$ and $\omega(M)(\gamma) = \omega(d)$ for $\gamma = (c, d) \in \Gamma$, $\Gamma(M)$ for Γ, $C(M)$ for $\{c(M)(\gamma) \mid \gamma \in \Gamma(M)\}$, $R(M)$ for R, $\Gamma_0(M)$ for Γ_0, $c_0(M)$ for c_0, and $\Omega_0(M)$ for $\{\omega(M)(\gamma_0) \mid \gamma_0 \in \Gamma_0\}$.

The restriction to reachable transition systems is not strictly necessary and could be replaced by constraining all statements on event/data structures to take into account only their reachable part (see, e.g., Lem. 1).

Example 6. For an event/data structure for the machine in Fig. 2 over its signature Σ in Ex. 1 we may choose the control states C as $\{s1, s2\}$, and the data names D as the set $\Omega(A(\Sigma)) = \mathcal{D}^{\{\mathsf{cnt}\}}$. In particular, the data state labelling ω is just the identity. The only initial configuration is $(s1, \{\mathsf{cnt} \mapsto 0\})$. A possible transition goes from configuration $(s1, \{\mathsf{cnt} \mapsto 2\})$ to configuration $(s2, \{\mathsf{cnt} \mapsto 4\})$ with the instantiated event $\mathsf{inc}(2)$.

A $\Sigma(U)$-event/data structure M is a *model* of a simple UML state machine U if $C(U) \subseteq C(M)$ up to a bijective renaming, $c_0(M) = c_0(U)$, $\Omega_0(M) \subseteq \{\omega \in |\Omega(A(\Sigma(U)))| \mid \omega \models^{\mathcal{D}}_{A(\Sigma(U))} \varphi_0(U)\}$, and if the following holds for all $(c, d) \in \Gamma(M)$:

- for all $(c, \phi, e(X), \psi, c') \in T(U)$ and $\beta\colon X \to \mathcal{D}$ with $\omega(M)(d), \beta \models^{\mathcal{D}}_{A(\Sigma(U)),X} \phi$, there is a $((c, d), (c', d')) \in R(M)_{e(\beta)}$ with $(\omega(M)(d), \omega(M)(d')), \beta \models^{2\mathcal{D}}_{A(\Sigma(U)),X} \psi$;
- for all $((c, d), (c', d')) \in R(M)_{e(\beta)}$ there is either some $(c, \phi, e(X), \psi, c') \in T(U)$ with $\omega(M)(d), \beta \models^{\mathcal{D}}_{A(\Sigma(U)),X} \phi$ and $(\omega(M)(d), \omega(M)(d')), \beta \models^{2\mathcal{D}}_{A(\Sigma(U)),X} \psi$, or $\omega(M)(d), \beta \not\models^{\mathcal{D}}_{A(\Sigma(U)),X} \bigvee_{(c,\phi,e(X),\psi,c') \in T(U)} \phi, c = c'$, and $\omega(M)(d) = \omega(M)(d')$.

A model of U thus on the one hand implements each transition prescribed by U, but on the other hand must not show transitions not covered by the specified transitions. Moreover, it is *input-enabled*, i.e., every event can be consumed in every control state: If no precondition of an explicitly specified transition is satisfied, there is a self-loop which leaves the data state untouched. In fact, input-enabledness, as required by the UML specification [17], can also be rendered as a syntactic transformation making a simple UML state machine U input-enabled by adding the following set of transition specifications for idling self-loops:

$$\{(c, \neg(\textstyle\bigvee_{(c,\phi,e(X),\psi,c') \in T(U)} \phi), e(X), 1_{A(\Sigma(U))}, c) \mid c \in C,\ e(X) \in E(\Sigma(U))\}\,.$$

Example 7. For the simple UML state machine in Fig. 2 the "grey" transitions correspond to an input-enabledness completion w.r.t. the "black" transitions.

The requirement of syntactic reachability for simple UML state machines is correlated with the requirement of (semantic) reachability of event/data structures, as a machine violating syntactic reachability cannot have a model. Equally, a machine with a non-satisfiable initial state predicate fails to have a model.

3.4 Event/Data Signature Morphisms, Reducts, and Translations

The external interface of a simple UML state machine is given by events, its internal interface by attributes. Both interfaces, represented as an event/data signature, are susceptible to change in the system development process which is captured by signature morphisms. Such changes have also to be reflected in the guards and effects, i.e., data state and transition predicates, by syntactical translations as well as in the interpretation domains by semantical reducts.

A *data signature morphism* from a data signature A to a data signature A' is a function $\alpha\colon A \to A'$. The α-*reduct* of an A'-data state $\omega'\colon A' \to \mathcal{D}$ along a data signature morphism $\alpha\colon A \to A'$ is given by the A-data state $\omega'|\alpha\colon A \to \mathcal{D}$ with $(\omega'|\alpha)(a) = \omega'(\alpha(a))$ for every $a \in A$; the α-*reduct* of an A'-data transition (ω', ω'') by the A-data transition $(\omega', \omega'')|\alpha = (\omega'|\alpha, \omega''|\alpha)$. The *state predicate translation* $\mathscr{F}^{\mathcal{D}}_{\alpha,X}\colon \mathscr{F}^{\mathcal{D}}_{A,X} \to \mathscr{F}^{\mathcal{D}}_{A',X}$ along a data signature morphism $\alpha\colon A \to A'$ is given by the CASL-formula translation $\mathscr{F}^{\text{CASL}}_{Sig(Dt),\alpha\cup 1_X}$ along the substitution $\alpha \cup 1_X$; the *transition predicate translation* $\mathscr{F}^{2\mathcal{D}}_{\alpha,X}$ by $\mathscr{F}^{\mathcal{D}}_{2\alpha,X}$ with $2\alpha\colon 2A \to 2A'$ defined by $2\alpha(a) = \alpha(a)$ and $2\alpha(a') = \alpha(a)'$. For each of these two reduct-translation-pairs the *satisfaction condition* holds due to the general substitution lemma for CASL:

$$\begin{aligned} \omega'|\alpha, \beta \models^{\mathcal{D}}_{A,X} \phi &\iff \omega', \beta \models^{\mathcal{D}}_{A',X} \mathscr{F}^{\mathcal{D}}_{\alpha,X}(\phi) \\ (\omega', \omega'')|\alpha, \beta \models^{2\mathcal{D}}_{A,X} \psi &\iff (\omega', \omega''), \beta \models^{2\mathcal{D}}_{A',X} \mathscr{F}^{2\mathcal{D}}_{\alpha,X}(\psi) \end{aligned}$$

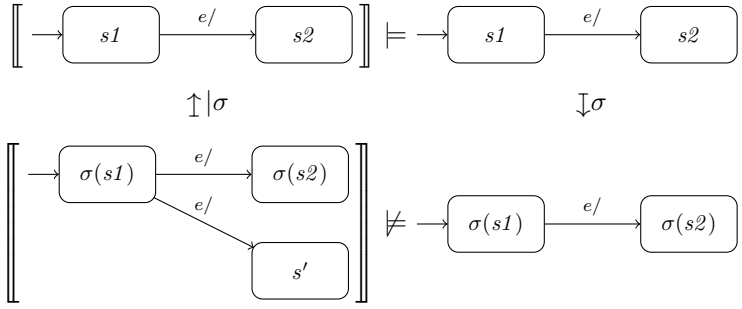
An *event signature morphism* $\eta\colon E \to E'$ is a function $\eta\colon |E| \to |E'|$ such that $\upsilon(E)(e) = \upsilon(E')(\eta(e))$ for all $e \in |E|$. An *event/data signature morphism* $\sigma\colon \Sigma \to \Sigma'$ consists of an event signature morphism $E(\sigma)\colon E(\Sigma) \to E(\Sigma')$ and a data signature morphism $A(\sigma)\colon A(\Sigma) \to A(\Sigma')$. The σ-*reduct* of a Σ'-event/data structure M' along σ is the Σ-event/data structure $M'|\sigma$ such that

- $\Gamma(M'|\sigma) \subseteq \Gamma(M')$ as well as $R(M'|\sigma) = (R(M'|\sigma)_{e(\beta)})_{e(X)\in E(\Sigma),\beta\colon X\to\mathcal{D}}$ are inductively defined by $\Gamma(M'|\sigma) \supseteq \Gamma_0(M')$ and, for all $\gamma', \gamma'' \in \Gamma(M')$, $e(X) \in E(\Sigma)$, and $\beta\colon X \to \mathcal{D}$, if $\gamma' \in \Gamma(M'|\sigma)$ and $(\gamma', \gamma'') \in R(M')_{E(\sigma)(e)(\beta)}$, then $\gamma'' \in \Gamma(M'|\sigma)$ and $(\gamma', \gamma'') \in R(M'|\sigma)_{e(\beta)}$;
- $\Gamma_0(M'|\sigma) = \Gamma_0(M')$; and
- $\omega(M'|\sigma)(\gamma') = (\omega(M')(\gamma'))|\sigma$ for all $\gamma' \in \Gamma(M'|\sigma)$.

Building a reduct of an event/data-structure does not affect the single configurations, but potentially reduces the set of configurations by restricting the available events, and the data state observable from the data name of a configuration. We denote by $\Gamma^F(M, \gamma)$ and $\Gamma^F(M)$, respectively, the set of configurations of a Σ-event/data structure M that are F-reachable from a configuration $\gamma \in \Gamma(M)$ and from an initial configuration $\gamma_0 \in \Gamma_0(M)$, respectively, with a set of events $F \subseteq E(\Sigma)$ where a $\gamma_n \in \Gamma(M)$ is F-*reachable in* M *from* a $\gamma_1 \in \Gamma(M)$ if there are $n \geq 1$, $e_2(X_2), \ldots, e_n(X_n) \in F$, $\beta_2\colon X_2 \to \mathcal{D}, \ldots, \beta_n\colon X_n \to \mathcal{D}$, and $(\gamma_i, \gamma_{i+1}) \in R(M)_{e_{i+1}(\beta_{i+1})}$ for all $1 \leq i < n$.

Although it is straightforward to define a translation of simple UML state machines along an event/data signature morphism, the rather restrictive notion of their models prevents the satisfaction condition to hold. In fact, this is already true for our previous endeavours to institutionalise UML state machines [10,8]. There machines themselves

were taken to be sentences over signatures comprising both events and states, and the satisfaction relation also required that a model shows exactly the transitions of such a machine sentence. For signature morphisms σ that are not surjective on states, building the reduct could result in less states and transitions, which leads to the following counterexample to the satisfaction condition [19]:



We therefore propose to make a detour through a more general hybrid modal logic. This logic is directly based on event/data structures and thus close to the domain of state machines. For forming an institution, its hybrid features allow to avoid control states as part of the signature and its event-based modalities allow to specify both mandatory and forbidden behaviour in a more fine-grained manner. Still, the logic is expressive enough to characterise the model class of a simple UML state machine syntactically.

4 A Hybrid Modal Logic for Event/Data Systems

The logic $\mathcal{M}_{\mathcal{D}}^{\downarrow}$ is a hybrid modal logic for specifying event/data-based reactive systems and reasoning about them. The $\mathcal{M}_{\mathcal{D}}^{\downarrow}$-signatures are the event/data signatures, the $\mathcal{M}_{\mathcal{D}}^{\downarrow}$-structures the event/data structures. The modal part of the logic allows to handle transitions between configurations where the modalities describe moves between configurations that adhere to a pre-condition or guard as a state predicate for an event with arguments and a transition predicate for the data change corresponding to effects. The hybrid part of the logic allows to bind control states of system configurations and to jump to configurations with such control states explicitly, but leaves out nominals as interfacing names as well as the possibility to quantify over control states. The logic builds on the hybrid dynamic logic $\mathcal{D}^{\downarrow}$ for specifying reactive systems without data [13] and its extension $\mathcal{E}^{\downarrow}$ to handle also data [6]. We restrict ourselves to modal operators consisting only of single instead of compound actions as done in dynamic logic. However, we still retain a box modality for accessing all configurations that are reachable from a given configuration. Moreover, we extend $\mathcal{E}^{\downarrow}$ by adding parameters to events.

The category of $\mathcal{M}_{\mathcal{D}}^{\downarrow}$*-signatures* $\mathbb{S}^{\mathcal{M}_{\mathcal{D}}^{\downarrow}}$ consists of the event/data signatures and signature morphisms. The Σ-event/data structures form the discrete category $Str^{\mathcal{M}_{\mathcal{D}}^{\downarrow}}(\Sigma)$ of $\mathcal{M}_{\mathcal{D}}^{\downarrow}$*-structures* over Σ. For each signature morphism $\sigma\colon \Sigma \to \Sigma'$ in $\mathbb{S}^{\mathcal{M}_{\mathcal{D}}^{\downarrow}}$ the σ*-reduct functor* $Str^{\mathcal{M}_{\mathcal{D}}^{\downarrow}}(\sigma)\colon Str^{\mathcal{M}_{\mathcal{D}}^{\downarrow}}(\Sigma') \to Str^{\mathcal{M}_{\mathcal{D}}^{\downarrow}}(\Sigma)$ is given by $Str^{\mathcal{M}_{\mathcal{D}}^{\downarrow}}(\sigma)(M') = M'|\sigma$. As the next step we introduce the formulæ and sentences of $\mathcal{M}_{\mathcal{D}}^{\downarrow}$ together with their

translation along $\mathbb{S}^{\mathcal{M}^{\downarrow}_{\mathcal{D}}}$-morphisms and their satisfaction over $Str^{\mathcal{M}^{\downarrow}_{\mathcal{D}}}$. We then show that for $\mathcal{M}^{\downarrow}_{\mathcal{D}}$ the satisfaction condition holds and thus obtain $\mathcal{M}^{\downarrow}_{\mathcal{D}}$ as an institution. Subsequently, we show that $\mathcal{M}^{\downarrow}_{\mathcal{D}}$ is simultaneously expressive enough to characterise the model class of simple UML state machines.

4.1 Formulæ and Sentences of $\mathcal{M}^{\downarrow}_{\mathcal{D}}$

$\mathcal{M}^{\downarrow}_{\mathcal{D}}$-formulæ aim at expressing control and data state properties of configurations as well as accessibility properties of configurations along transitions for particular events. The pure data state part is captured by data state sentences over $\mathcal{D}$. The control state part can be accessed and manipulated by hybrid operators for binding the control state in a state variable, $\downarrow s$; checking for a particular control state, s; and accessing all configurations with a particular control state, $@^F$, which, however, only pertains to reachable configurations relative to a set F of events. Transitions between configurations are covered by different modalities: a box modality for accessing all configurations that are reachable from a given configuration, $\Box^F$, again relative to a set F of events; a diamond modality for checking that an event with arguments is possible with a particular data state change, $\langle e(X) /\!\!/ \psi \rangle$; and a modality for checking the reaction to an event with arguments according to a pre-condition and a transition predicate, $\langle\!| e(X) : \phi /\!\!/ \psi |\!\rangle$.

Formally, the *Σ-event/data formulæ* $\mathscr{F}^{\mathcal{M}^{\downarrow}_{\mathcal{D}}}_{\Sigma,S}$ over an event/data signature Σ and a set of *state variables* S are inductively defined by

- φ — data state sentence $\varphi \in \mathscr{F}^{\mathcal{D}}_{A(\Sigma),\emptyset}$ holds in the current configuration;
- s — the control state of the current configuration is $s \in S$;
- $\downarrow s \,.\, \varrho$ — calling the current control state s, formula $\varrho \in \mathscr{F}^{\mathcal{M}^{\downarrow}_{\mathcal{D}}}_{\Sigma,S \uplus \{s\}}$ holds;
- $(@^F s)\varrho$ — in all configurations with control state $s \in S$ that are reachable with events from $F \subseteq E(\Sigma)$ formula $\varrho \in \mathscr{F}^{\mathcal{M}^{\downarrow}_{\mathcal{D}}}_{\Sigma,S}$ holds;
- $\Box^F \varrho$ — in all configurations that are reachable from the current configuration with events from $F \subseteq E(\Sigma)$ formula $\varrho \in \mathscr{F}^{\mathcal{M}^{\downarrow}_{\mathcal{D}}}_{\Sigma,S}$ holds;
- $\langle e(X) /\!\!/ \psi \rangle \varrho$ — in the current configuration there is a valuation of X and a transition for event $e(X) \in E(\Sigma)$ with these arguments that satisfies transition formula $\psi \in \mathscr{F}^{2\mathcal{D}}_{A(\Sigma),X}$ and makes $\varrho \in \mathscr{F}^{\mathcal{M}^{\downarrow}_{\mathcal{D}}}_{\Sigma,S}$ hold afterwards;
- $\langle\!| e(X) : \phi /\!\!/ \psi |\!\rangle \varrho$ — in the current configuration for all valuations of X satisfying state formula $\phi \in \mathscr{F}^{\mathcal{D}}_{A(\Sigma),X}$ there is a transition for event $e(X) \in E(\Sigma)$ with these arguments that satisfies transition formula $\psi \in \mathscr{F}^{2\mathcal{D}}_{A(\Sigma),X}$ and makes $\varrho \in \mathscr{F}^{\mathcal{M}^{\downarrow}_{\mathcal{D}}}_{\Sigma,S}$ hold afterwards;
- $\neg \varrho$ — in the current configuration $\varrho \in \mathscr{F}^{\mathcal{M}^{\downarrow}_{\mathcal{D}}}_{\Sigma,S}$ does not hold;
- $\varrho_1 \vee \varrho_2$ — in the current configuration $\varrho_1 \in \mathscr{F}^{\mathcal{M}^{\downarrow}_{\mathcal{D}}}_{\Sigma,S}$ or $\varrho_2 \in \mathscr{F}^{\mathcal{M}^{\downarrow}_{\mathcal{D}}}_{\Sigma,S}$ hold.

We write $(@s)\varrho$ for $(@^{E(\Sigma)}s)\varrho$, $\Box\varrho$ for $\Box^{E(\Sigma)}\varrho$, $\Diamond^F\varrho$ for $\neg\Box^F\neg\varrho$, $\Diamond\varrho$ for $\Diamond^{E(\Sigma)}\varrho$, $[e(X) /\!\!/ \psi]\varrho$ for $\neg\langle e(X) /\!\!/ \psi\rangle\neg\varrho$, and true for $\downarrow s \,.\, s$.

Example 8. An event/data formula can make two kinds of requirements on an event/data structure: On the one hand, it can require the presence of certain mandatory transitions, on the other hand it can require the absence of certain prohibited transitions. Considering the simple UML state machine in Fig. 2, the formula

$$(@s1)\langle\!|\mathsf{inc}(x) : \mathsf{cnt} + x = 4 /\!\!/ \mathsf{cnt}' = 4|\!\rangle s2$$

requires for each valuation of $\beta\colon \{x\} \to \mathbb{N}$ such that $\mathsf{cnt} + x = 4$ holds that there is a transition from control state $s1$ to control state $s2$ for the instantiated event $\mathsf{inc}(\beta)$ where cnt is changed to 4. On the other hand, the formula

$$(@s2)[\mathsf{reset} /\!\!/ \neg(\mathsf{cnt}' = 0)]\mathrm{false}$$

prohibits any transitions out of $s2$ that are labelled with the event reset but do not satisfy $\mathsf{cnt}' = 0$.

In the context of Fig. 2, these formulæ only have their explained intended meaning when $s1$ and $s2$ indeed refer to the eponymous states. However, $\mathcal{M}^{\downarrow}_{\mathcal{D}}$ does not show nominals for explicitly naming control states as part of the state machine's interface and the reference to specific states always has to build these states' context first using the modalities and the bind operator. On the other hand, as indicated in Sect. 3.4, the inclusion of nominals may interfere disadvantageously with the reduct formation.

Let $\sigma\colon \Sigma \to \Sigma'$ be an event/data signature morphism. The *event/data formulæ translation* $\mathscr{F}^{\mathcal{M}^{\downarrow}_{\mathcal{D}}}_{\sigma,S} : \mathscr{F}^{\mathcal{M}^{\downarrow}_{\mathcal{D}}}_{\Sigma,S} \to \mathscr{F}^{\mathcal{M}^{\downarrow}_{\mathcal{D}}}_{\Sigma',S}$ along σ is recursively given by

- $\mathscr{F}^{\mathcal{M}^{\downarrow}_{\mathcal{D}}}_{\sigma,S}(\varphi) = \mathscr{F}^{\mathcal{D}}_{A(\sigma),\emptyset}(\varphi)$;
- $\mathscr{F}^{\mathcal{M}^{\downarrow}_{\mathcal{D}}}_{\sigma,S}(s) = s$;
- $\mathscr{F}^{\mathcal{M}^{\downarrow}_{\mathcal{D}}}_{\sigma,S}(\downarrow s\,.\,\varrho) = \downarrow s\,.\,\mathscr{F}^{\mathcal{M}^{\downarrow}_{\mathcal{D}}}_{\sigma,S\uplus\{s\}}(\varrho)$;
- $\mathscr{F}^{\mathcal{M}^{\downarrow}_{\mathcal{D}}}_{\sigma,S}((@^F s)\varrho) = (@^{E(\sigma)(F)} s)\mathscr{F}^{\mathcal{M}^{\downarrow}_{\mathcal{D}}}_{\sigma,S}(\varrho)$;
- $\mathscr{F}^{\mathcal{M}^{\downarrow}_{\mathcal{D}}}_{\sigma,S}(\Box^F \varrho) = \Box^{E(\sigma)(F)}\mathscr{F}^{\mathcal{M}^{\downarrow}_{\mathcal{D}}}_{\sigma,S}(\varrho)$;
- $\mathscr{F}^{\mathcal{M}^{\downarrow}_{\mathcal{D}}}_{\sigma,S}(\langle e(X) /\!\!/ \psi\rangle\varrho) = \langle E(\sigma)(e)(X) /\!\!/ \mathscr{F}^{2\mathcal{D}}_{A(\sigma),X}(\psi)\rangle\mathscr{F}^{\mathcal{M}^{\downarrow}_{\mathcal{D}}}_{\sigma,S}(\varrho)$;
- $\mathscr{F}^{\mathcal{M}^{\downarrow}_{\mathcal{D}}}_{\sigma,S}(\langle\!|e(X) : \phi /\!\!/ \psi|\!\rangle\varrho) = \langle\!|E(\sigma)(e)(X) : \mathscr{F}^{\mathcal{D}}_{A(\sigma),X}(\phi) /\!\!/ \mathscr{F}^{2\mathcal{D}}_{A(\sigma),X}(\psi)|\!\rangle\mathscr{F}^{\mathcal{M}^{\downarrow}_{\mathcal{D}}}_{\sigma,S}(\varrho)$;
- $\mathscr{F}^{\mathcal{M}^{\downarrow}_{\mathcal{D}}}_{\sigma,S}(\neg\varrho) = \neg\mathscr{F}^{\mathcal{M}^{\downarrow}_{\mathcal{D}}}_{\sigma,S}(\varrho)$;
- $\mathscr{F}^{\mathcal{M}^{\downarrow}_{\mathcal{D}}}_{\sigma,S}(\varrho_1 \vee \varrho_2) = \mathscr{F}^{\mathcal{M}^{\downarrow}_{\mathcal{D}}}_{\sigma,S}(\varrho_1) \vee \mathscr{F}^{\mathcal{M}^{\downarrow}_{\mathcal{D}}}_{\sigma,S}(\varrho_2)$.

The set $\mathrm{Sen}^{\mathcal{M}^{\downarrow}_{\mathcal{D}}}(\Sigma)$ of *Σ-event/data sentences* is given by $\mathscr{F}^{\mathcal{M}^{\downarrow}_{\mathcal{D}}}_{\Sigma,\emptyset}$, the *event/data sentence translation* $\mathrm{Sen}^{\mathcal{M}^{\downarrow}_{\mathcal{D}}}(\sigma)\colon \mathrm{Sen}^{\mathcal{M}^{\downarrow}_{\mathcal{D}}}(\Sigma) \to \mathrm{Sen}^{\mathcal{M}^{\downarrow}_{\mathcal{D}}}(\Sigma')$ by $\mathscr{F}^{\mathcal{M}^{\downarrow}_{\mathcal{D}}}_{\sigma,\emptyset}$.

4.2 Satisfaction Relation for $\mathcal{M}^{\downarrow}_{\mathcal{D}}$

The $\mathcal{M}^{\downarrow}_{\mathcal{D}}$-satisfaction relation connects $\mathcal{M}^{\downarrow}_{\mathcal{D}}$-structures and $\mathcal{M}^{\downarrow}_{\mathcal{D}}$-formulæ, expressing whether in some configuration of the structure a particular formula holds with respect to

an assignment of control states to state variables. Let Σ be an event/data signature, M a Σ-event/data structure, S a set of state variables, $v\colon S \to C(M)$ a state variable assignment, and $\gamma \in \Gamma(M)$. The *satisfaction relation* for event/data formulæ is inductively given by

- $M, v, \gamma \models^{\mathcal{M}^{\downarrow}_{\mathcal{D}}}_{\Sigma,S} \varphi$ iff $\omega(M)(\gamma) \models^{\mathcal{D}}_{A(\Sigma)} \varphi$;
- $M, v, \gamma \models^{\mathcal{M}^{\downarrow}_{\mathcal{D}}}_{\Sigma,S} s$ iff $v(s) = c(M)(\gamma)$;
- $M, v, \gamma \models^{\mathcal{M}^{\downarrow}_{\mathcal{D}}}_{\Sigma,S} \downarrow s\,.\,\varrho$ iff $M, v\{s \mapsto c(M)(\gamma)\}, \gamma \models^{\mathcal{M}^{\downarrow}_{\mathcal{D}}}_{\Sigma,S\uplus\{s\}} \varrho$;
- $M, v, \gamma \models^{\mathcal{M}^{\downarrow}_{\mathcal{D}}}_{\Sigma,S} (@^F s)\varrho$ iff $M, v, \gamma' \models^{\mathcal{M}^{\downarrow}_{\mathcal{D}}}_{\Sigma,S} \varrho$ for all $\gamma' \in \Gamma^F(M)$ with $c(M)(\gamma') = v(s)$;
- $M, v, \gamma \models^{\mathcal{M}^{\downarrow}_{\mathcal{D}}}_{\Sigma,S} \Box^F \varrho$ iff $M, v, \gamma' \models^{\mathcal{M}^{\downarrow}_{\mathcal{D}}}_{\Sigma,S} \varrho$ for all $\gamma' \in \Gamma^F(M, \gamma)$;
- $M, v, \gamma \models^{\mathcal{M}^{\downarrow}_{\mathcal{D}}}_{\Sigma,S} \langle e(X) \mathbin{/\!\!/} \psi\rangle\varrho$ iff there is a $\beta\colon X \to \mathcal{D}$ and a $\gamma' \in \Gamma(M)$ such that $(\gamma, \gamma') \in R(M)_{e(\beta)}$, $(\omega(M)(\gamma), \omega(M)(\gamma')), \beta \models^{2\mathcal{D}}_{A(\Sigma),X} \psi$, and $M, v, \gamma' \models^{\mathcal{M}^{\downarrow}_{\mathcal{D}}}_{\Sigma,S} \varrho$;
- $M, v, \gamma \models^{\mathcal{M}^{\downarrow}_{\mathcal{D}}}_{\Sigma,S} \langle\!| e(X) : \phi \mathbin{/\!\!/} \psi |\!\rangle \varrho$ iff for all $\beta\colon X \to \mathcal{D}$ with $\omega(M)(\gamma), \beta \models^{\mathcal{D}}_{A(\Sigma),X} \phi$ there is some $\gamma' \in \Gamma(M)$ such that $(\gamma, \gamma') \in R(M)_{e(\beta)}$, $(\omega(M)(\gamma), \omega(M)(\gamma')), \beta \models^{2\mathcal{D}}_{A(\Sigma),X} \psi$, and $M, v, \gamma' \models^{\mathcal{M}^{\downarrow}_{\mathcal{D}}}_{\Sigma,S} \varrho$;
- $M, v, \gamma \models^{\mathcal{M}^{\downarrow}_{\mathcal{D}}}_{\Sigma,S} \neg\varrho$ iff $M, v, \gamma \not\models^{\mathcal{M}^{\downarrow}_{\mathcal{D}}}_{\Sigma,S} \varrho$;
- $M, v, \gamma \models^{\mathcal{M}^{\downarrow}_{\mathcal{D}}}_{\Sigma,S} \varrho_1 \vee \varrho_2$ iff $M, v, \gamma \models^{\mathcal{M}^{\downarrow}_{\mathcal{D}}}_{\Sigma,S} \varrho_1$ or $M, v, \gamma \models^{\mathcal{M}^{\downarrow}_{\mathcal{D}}}_{\Sigma,S} \varrho_2$.

This satisfaction relation is well-behaved with respect to reducts of $\mathcal{M}^{\downarrow}_{\mathcal{D}}$-structures. On the one hand, this is due to the use of abstract data names rather than data states in the structures, and on the other hand to the satisfaction condition of $\mathcal{D}$ and $2\mathcal{D}$.

Lemma 1. *Let $\sigma\colon \Sigma \to \Sigma'$ be a event/data signature morphism and M' a Σ'-event/data structure. For all $\varrho \in \mathscr{F}^{\mathcal{M}^{\downarrow}_{\mathcal{D}}}_{\Sigma,S}$, all $\gamma' \in \Gamma(M'|\sigma) \subseteq \Gamma(M')$, and all $v\colon S \to C(M'|\sigma) \subseteq C(M')$ it holds that*

$$M'|\sigma, v, \gamma' \models^{\mathcal{M}^{\downarrow}_{\mathcal{D}}}_{\Sigma,S} \varrho \iff M', v, \gamma' \models^{\mathcal{M}^{\downarrow}_{\mathcal{D}}}_{\Sigma',S} \mathscr{F}^{\mathcal{M}^{\downarrow}_{\mathcal{D}}}_{\sigma,S}(\varrho)\,.$$

For a $\Sigma \in |\mathbb{S}^{\mathcal{M}^{\downarrow}_{\mathcal{D}}}|$, an $M \in |Str^{\mathcal{M}^{\downarrow}_{\mathcal{D}}}(\Sigma)|$, and a $\rho \in \mathrm{Sen}^{\mathcal{M}^{\downarrow}_{\mathcal{D}}}(\Sigma)$ the *satisfaction relation* $M \models^{\mathcal{M}^{\downarrow}_{\mathcal{D}}}_{\Sigma} \rho$ holds if, and only if, $M, \emptyset, \gamma_0 \models^{\mathcal{M}^{\downarrow}_{\mathcal{D}}}_{\Sigma,\emptyset} \rho$ for all $\gamma_0 \in \Gamma_0(M)$.

Theorem 1. $(\mathbb{S}^{\mathcal{M}^{\downarrow}_{\mathcal{D}}}, Str^{\mathcal{M}^{\downarrow}_{\mathcal{D}}}, \mathrm{Sen}^{\mathcal{M}^{\downarrow}_{\mathcal{D}}}, \models^{\mathcal{M}^{\downarrow}_{\mathcal{D}}})$ *is an institution.*

4.3 Representing Simple UML State Machines in $\mathcal{M}^{\downarrow}_{\mathcal{D}}$

The hybrid modal logic $\mathcal{M}^{\downarrow}_{\mathcal{D}}$ is expressive enough to characterise the model class of a simple UML state machine U by a single sentence ϱ_U, i.e., an event/data structure M is a model of U if, and only if, $M \models^{\mathcal{M}^{\downarrow}_{\mathcal{D}}}_{\Sigma(U)} \varrho_U$. Such a characterisation is achieved by

Alg. 1 Constructing an $\mathcal{M}^{\downarrow}_{\mathcal{D}}$-sentence from a set of transition specifications

Require: $T \equiv$ a set of transition specifications
$Im_T(c) = \{(\phi, e(X), \psi, c') \mid (c, \phi, e(X), \psi, c') \in T\}$
$Im_T(c, e(X)) = \{(\phi, \psi, c') \mid (c, \varphi, e(X), \psi, c') \in T\}$

function $\mathrm{sen}(c, I, V, B)$ ▷ c: state, I: image to visit, V: states to visit, B: bound states
  if $I \neq \emptyset$ **then**
    $(\phi, e(X), \psi, c') \leftarrow$ **choose** I
    if $c' \in B$ **then**
      return $(@c)\langle\!|e(X) : \phi \mathbin{/\!\!/} \psi|\!\rangle(c' \wedge \mathrm{sen}(c, I \setminus \{(\phi, e(X), \psi, c')\}, V, B))$
    else
      return $(@c)\langle\!|e(X) : \phi \mathbin{/\!\!/} \psi|\!\rangle(\downarrow c' \,.\, \mathrm{sen}(c, I \setminus \{(\phi, e(X), \psi, c')\}, V, B \cup \{c'\}))$
  $V \leftarrow V \setminus \{c\}$
  if $V \neq \emptyset$ **then**
    $c' \leftarrow$ **choose** $B \cap V$
    return $\mathrm{sen}(c', Im_T(c'), V, B)$
  return $(\bigwedge_{c \in B} \mathrm{fin}(c)) \wedge \bigwedge_{c_1 \in B, c_2 \in B \setminus \{c_1\}} \neg(@c_1)c_2$

function $\mathrm{fin}(c)$
  return $(@c)\bigwedge_{e(X) \in E(\Sigma(U))} \bigwedge_{P \subseteq Im_T(c, e(X))} [e(X) \mathbin{/\!\!/} \big(\bigwedge_{(\phi,\psi,c') \in P}(\phi \wedge \psi)\big) \wedge \neg\big(\bigvee_{(\phi,\psi,c') \in Im_T(c,e(X)) \setminus P}(\phi \wedge \psi)\big)]\big(\bigvee_{(\phi,\psi,c') \in P} c'\big)$

means of Alg. 1 that is a slight variation of the characterisation algorithm for so-called operational specifications within $\mathcal{E}^{\downarrow}$ [6] by including also events with data arguments. The algorithm constructs a sentence expressing that semantic transitions according to explicit syntactic transition specifications are indeed possible and that no other semantic transitions not adhering to any of the syntactic transition specifications exist. For a set of transition specifications T, a call $\mathrm{sen}(c, I, V, B)$ performs a recursive breadth-first traversal starting from c, where I holds the unprocessed quadruples $(\phi, e(X), \psi, c')$ of transitions in T outgoing from c, V the remaining states to visit, and B the set of already bound states. The function first requires the existence of each outgoing transition of I in the resulting formula, binding any newly reached state. Having visited all states in V, it requires that no other transitions from the states in B exist using calls to fin, and adds the requirement that all states in B are pairwise different. Formula $\mathrm{fin}(c)$ expresses that at c, for all events $e(X)$ and for all subsets P of the transitions in T outgoing from c, whenever an $e(X)$-transition can be done with the combined effect of P but not adhering to any of the effects of the currently not selected transitions, the $e(X)$-transition must have one of the states as its target that are target states of P.

Example 9. Applying Alg. 1 to the set of explicitly mentioned, "black" transition specifications T of the simple UML state machine *Counter* in Fig. 2, i.e., calling $\mathrm{sen}(s1, Im_T(s1), \{s1, s2\}, \{s1\})$ yields $\varrho_{s1,s1}$ with

$$\varrho_{s1,s1} = (@s1)\langle\!|\mathsf{inc}(x) : \mathsf{cnt} + x \leq 4 \mathbin{/\!\!/} \mathsf{cnt}' = \mathsf{cnt} + x|\!\rangle(s1 \wedge \varrho_{s1,s2})$$
$$\varrho_{s1,s2} = (@s1)\langle\!|\mathsf{inc}(x) : \mathsf{cnt} + x = 4 \mathbin{/\!\!/} \mathsf{cnt}' = \mathsf{cnt} + x|\!\rangle\downarrow s2 \,.\, (\varrho_{s2,s1})$$
$$\varrho_{s2,s1} = (@s2)\langle\!|\mathsf{reset} : \mathrm{true} \mathbin{/\!\!/} \mathsf{cnt}' = 0|\!\rangle(s1 \wedge \varrho_{\mathrm{fin}})$$

$$
\begin{aligned}
\varrho_{\mathrm{fin}} &= \varrho_{\mathrm{fin}(s1)} \wedge \varrho_{\mathrm{fin}(s2)} \wedge \neg(@s1)s2 \\
\varrho_{\mathrm{fin}(s1)} &= (@s1)\big([\mathsf{inc}(x)/\!\!/\neg((\mathsf{cnt}+x \leq 4 \wedge \mathsf{cnt}' = \mathsf{cnt}+x) \vee \\
&\qquad (\mathsf{cnt}+x = 4 \wedge \mathsf{cnt}' = 4))]\mathrm{false} \wedge \\
&\quad [\mathsf{inc}(x)/\!\!/(\mathsf{cnt}+x \leq 4 \wedge \mathsf{cnt}' = \mathsf{cnt}+x) \wedge \\
&\qquad \neg(\mathsf{cnt}+x = 4 \wedge \mathsf{cnt}' = 4)]s1 \wedge \\
&\quad [\mathsf{inc}(x)/\!\!/(\mathsf{cnt}+x = 4 \wedge \mathsf{cnt}' = 4) \wedge \\
&\qquad \neg(\mathsf{cnt}+x \leq 4 \wedge \mathsf{cnt}' = \mathsf{cnt}+x)]s2 \wedge \\
&\quad [\mathsf{inc}(x)/\!\!/(\mathsf{cnt}+x \leq 4 \wedge \mathsf{cnt}' = \mathsf{cnt}+x) \wedge \\
&\qquad (\mathsf{cnt}+x = 4 \wedge \mathsf{cnt}' = 4)](s1 \vee s2) \wedge \\
&\quad [\mathsf{reset}/\!\!/\mathrm{true}]\mathrm{false}\big) \\
\varrho_{\mathrm{fin}(s2)} &= (@s2)\big([\mathsf{inc}(x)/\!\!/\mathrm{true}]\mathrm{false} \wedge \\
&\quad [\mathsf{reset}/\!\!/\neg(\mathsf{cnt}' = 0)]\mathrm{false} \wedge \\
&\quad [\mathsf{reset}/\!\!/\mathsf{cnt}' = 0]s1\big)
\end{aligned}
$$

In fact, there is no outgoing "black" transition for reset from $s1$, thus $P = \emptyset$ is the only choice for this event in $\mathrm{fin}(s1)$ and the clause $[\mathsf{reset}/\!\!/\mathrm{true}]\mathrm{false}$ is included. For $\mathsf{inc}(x)$ there are two outgoing transitions resulting four different clauses checking whether none, the one or the other, or both transitions are executable.

In order to apply the algorithm to simple UML state machines, the idling self-loops for achieving input-enabledness first have to be made explicit. For a syntactically input-enabled simple UML state machine U a characterising sentence then reads

$$\varrho_U = {\downarrow}c_0 \,.\, \varphi_0 \wedge \mathrm{sen}(c_0, \mathit{Im}_{T(U)}(c_0), C(U), \{c_0\}) \,,$$

where $c_0 = c_0(U)$ and $\varphi_0 = \varphi_0(U)$. Due to syntactic reachability, the bound states B of Alg. 1 become $C(U)$ when sen is called for $B = \{c_0(U)\}$ and V reaches $\emptyset$.

5 A Theoroidal Comorphism from $\mathcal{M}^{\downarrow}_{\mathcal{D}}$ to CASL

We define a theoroidal comorphism from $\mathcal{M}^{\downarrow}_{\mathcal{D}}$ to CASL. The construction mainly follows the standard translation of modal logics to first-order logic [2] which has been considered for hybrid logics also on an institutional level [12,3].

The basis is a representation of $\mathcal{M}^{\downarrow}_{\mathcal{D}}$-signatures and the frame given by $\mathcal{M}^{\downarrow}_{\mathcal{D}}$-structures as a CASL-specification as shown in Fig. 3. The signature translation

$$\nu^{\mathbb{S}} \colon \mathbb{S}^{\mathcal{M}^{\downarrow}_{\mathcal{D}}} \to \mathrm{Pres}^{\mathrm{CASL}}$$

maps a $\mathcal{M}^{\downarrow}_{\mathcal{D}}$-signature Σ to the CASL-theory presentation given by TRANS_{Σ} and a $\mathcal{M}^{\downarrow}_{\mathcal{D}}$-signature morphism to the corresponding theory presentation morphism. TRANS_{Σ} first of all covers the events and event names according to $E(\Sigma)$ (types Evt and EvtNm with several alternatives separated by "|") and the configurations (type Conf with a single constructor "conf") with their control states (sort Ctrl) and data states given by assignments to the attributes from $A(\Sigma)$ (separated by ";"). The remainder of TRANS_{Σ} sets the frame for describing reachable transition systems with a set of initial

from *Basic/StructuredDatatypes* **get** SET % import finite sets
spec $\text{TRANS}_\Sigma = Dt$
then free type $\mathrm{Evt} ::= \tau_\mathrm{e}(E(\Sigma))$
$\quad\quad$ % $\tau_\mathrm{e}(\{e(X)\}) = e(dt^{|X|})$, $\tau_\mathrm{e}(\{e(X)\} \cup E) = e(dt^{|X|}) \mid \tau_\mathrm{e}(E)$
$\quad$ **free type** $\mathrm{EvtNm} ::= \tau_\mathrm{n}(E(\Sigma))$ % $\tau_\mathrm{n}(\{e(X)\}) = e$, $\tau_\mathrm{n}(\{e(X)\} \cup E) = e \mid \tau_\mathrm{n}(E)$
$\quad$ **op** $\mathrm{nm} : \mathrm{Evt} \to \mathrm{EvtNm}$
$\quad$ **axiom** $\forall x_1, \dots, x_n : dt \cdot \mathrm{nm}(e(x_1, \dots, x_n)) = e$ % for each $e(x_1, \dots, x_n) \in E(\Sigma)$
then SET[sort EvtNm]
then sort Ctrl
$\quad$ **free type** $\mathrm{Conf} ::= \mathrm{conf}(\mathrm{c} : \mathrm{Ctrl}; \tau_\mathrm{a}(A(\Sigma)))$
$\quad\quad$ % $\tau_\mathrm{a}(\{a\}) = a : dt$, $\tau_\mathrm{a}(\{a\} \cup A) = a : dt; \tau_\mathrm{a}(A)$
$\quad$ **preds** $\mathrm{init} : \mathrm{Conf}$;
$\quad\quad$ $\mathrm{trans} : \mathrm{Conf} \times \mathrm{Evt} \times \mathrm{Conf}$
$\quad$ $\cdot\ \exists g : \mathrm{Conf} \cdot \mathrm{init}(g)$ % there is some initial configuration
$\quad$ $\cdot\ \forall g, g' : \mathrm{Conf} \cdot \mathrm{init}(g) \wedge \mathrm{init}(g') \Rightarrow \mathrm{c}(g) = \mathrm{c}(g')$ % single initial control state
$\quad$ **free** { **pred** $\mathrm{reachable} : Set[\mathrm{EvtNm}] \times \mathrm{Conf} \times \mathrm{Conf}$
$\quad\quad$ $\forall g, g', g'' : \mathrm{Conf}, E : Set[\mathrm{EvtNm}], e : \mathrm{Evt}$
$\quad\quad$ $\cdot\ \mathrm{reachable}(E, g, g)$
$\quad\quad$ $\cdot\ \mathrm{reachable}(E, g, g') \wedge \mathrm{nm}(e) \in E \wedge \mathrm{trans}(g', e, g'') \Rightarrow \mathrm{reachable}(E, g, g'')$ }
$\quad$ **then preds** $\mathrm{reachable}(E : Set[\mathrm{EvtName}], g : \mathrm{Conf}) \Leftrightarrow$
$\quad\quad\quad$ $\exists g_0 : \mathrm{Conf} \cdot \mathrm{init}(g_0) \wedge \mathrm{reachable}(E, g_0, g)$;
$\quad\quad$ $\mathrm{reachable}(g : \mathrm{Conf}) \Leftrightarrow \mathrm{reachable}(E(\Sigma), g)$
end

Fig. 3. Frame for translating $\mathcal{M}^{\downarrow}_{\mathcal{D}}$ into CASL

configurations (predicate init), a transition relation (predicate trans) and reachability predicates. The specification of the predicate reachable uses CASL's "structured free" construct to ensure reachability to be inductively defined. The model translation

$$\nu^{\mathrm{Mod}}_{\Sigma} \colon \mathrm{Mod}^{\mathrm{CASL}}(\nu^{\mathbb{S}}(\Sigma)) \to \mathit{Str}^{\mathcal{M}^{\downarrow}_{\mathcal{D}}}(\Sigma)$$

then can rely on this encoding. In particular, for a model $M' \in \mathrm{Mod}^{\mathrm{CASL}}(\nu^{\mathbb{S}}(\Sigma))$, there are, using the bijection $\iota_{M',dt} \colon dt^{M'} \cong \mathcal{D}$, an injective map $\iota_{M',\mathrm{Conf}} \colon \mathrm{Conf}^{M'} \rightarrowtail \mathrm{Ctrl}^{M'} \times \Omega(A(\Sigma))$ and a bijective map $\iota_{M',\mathrm{Evt}} \colon \mathrm{Evt}^{M'} \cong \{e(\beta) \mid e(X) \in E(\Sigma),\ \beta \colon X \to \mathcal{D}\}$. The $\mathcal{M}^{\downarrow}_{\mathcal{D}}$-structure resulting from a CASL-model of TRANS_Σ can thus be defined by

- $\Gamma(\nu^{\mathrm{Mod}}_{\Sigma}(M')) = \iota^{-1}_{M',\mathrm{Conf}}(\{g' \in M'_{\mathrm{Conf}} \mid \mathrm{reachable}^{M'}(g')\})$
- $R(\nu^{\mathrm{Mod}}_{\Sigma}(M'))_{e(\beta)} = \{(\gamma, \gamma') \in \Gamma(\nu^{\mathrm{Mod}}_{\Sigma}(M')) \times \Gamma(\nu^{\mathrm{Mod}}_{\Sigma}(M')) \mid \mathrm{trans}^{M'}(\iota_{M',\mathrm{Conf}}(\gamma), \iota^{-1}_{M',\mathrm{Evt}}(e(\beta)), \iota_{M',\mathrm{Conf}}(\gamma'))\}$)
- $\Gamma_0(\nu^{\mathrm{Mod}}_{\Sigma}(M')) = \{\gamma \in \Gamma(\nu^{\mathrm{Mod}}_{\Sigma}(M')) \mid \mathrm{init}^{M'}(\iota_{M',\mathrm{Conf}}(\gamma))\}$)
- $\omega(\nu^{\mathrm{Mod}}_{\Sigma}(M')) = \{(c, \omega) \in \Gamma(\nu^{\mathrm{Mod}}_{\Sigma}(M')) \mapsto \omega\}$

For $\mathcal{M}^{\downarrow}_{\mathcal{D}}$-sentences, we first define a formula translation

$$\nu^{\mathscr{F}}_{\Sigma,S,g} \colon \mathscr{F}^{\mathcal{M}^{\downarrow}_{\mathcal{D}}}_{\Sigma,S} \to \mathscr{F}^{\mathrm{CASL}}_{\nu^{Sig}(\Sigma),S\cup\{g\}}$$

which, mimicking the standard translation, takes a variable $g : \mathrm{Conf}$ as a parameter that records the "current configuration" and also uses a set S of state names for the control states. The translation embeds the data state and 2-data state formulæ using the substitution $A(\Sigma)(g) = \{a \mapsto a(g) \mid a \in A(\Sigma)\}$ for replacing the attributes $a \in A(\Sigma)$ by the accessors $a(g)$. The translation of $\mathcal{M}^{\downarrow}_{\mathcal{D}}$-formulæ then reads

- $\nu^{\mathscr{F}}_{\Sigma,S,g}(\varphi) = \mathscr{F}^{\text{CASL}}_{\nu^{Sig}(\Sigma),A(\Sigma)(g)}(\varphi)$
- $\nu^{\mathscr{F}}_{\Sigma,S,g}(s) = (s = \mathrm{c}(g))$
- $\nu^{\mathscr{F}}_{\Sigma,S,g}(\downarrow s \,.\, \varrho) = \exists s : \mathrm{Ctrl} \,.\, s = \mathrm{c}(g) \wedge \nu^{\mathscr{F}}_{\Sigma,S\uplus\{s\},g}(\varrho)$
- $\nu^{\mathscr{F}}_{\Sigma,S,g}((@^F s)\varrho) = \forall g' : \mathrm{Conf} \,.\, (\mathrm{c}(g') = s \wedge \mathrm{reachable}(F, g')) \Rightarrow \nu^{\mathscr{F}}_{\Sigma,S,g'}(\varrho)$
- $\nu^{\mathscr{F}}_{\Sigma,S,g}(\Box^F \varrho) = \forall g' : \mathrm{Conf} \,.\, \mathrm{reachable}(F, g, g') \Rightarrow \nu^{\mathscr{F}}_{\Sigma,S,g'}(\varrho)$
- $\nu^{\mathscr{F}}_{\Sigma,S,g}(\langle e(X) /\!\!/ \psi \rangle \varrho) = \exists X : dt \,.\, \exists g' : \mathrm{Conf} \,.\, \mathrm{trans}(g, e(X), g') \wedge \mathscr{F}^{\text{CASL}}_{\nu^{Sig}(\Sigma),A(\Sigma)(g)\cup A(\Sigma)(g')\cup 1_X}(\psi) \wedge \nu^{\mathscr{F}}_{\Sigma,S,g'}(\varrho)$
- $\nu^{\mathscr{F}}_{\Sigma,S,g}(\langle\!| e(X) : \phi /\!\!/ \psi |\!\rangle \varrho) = \forall X : dt \,.\, \mathscr{F}^{\text{CASL}}_{\nu^{Sig}(\Sigma),A(\Sigma)(g)\cup 1_X}(\phi) \Rightarrow \exists g' : \mathrm{Conf} \,.\, \mathrm{trans}(g, e(X), g') \wedge \mathscr{F}^{\text{CASL}}_{\nu^{Sig}(\Sigma),A(\Sigma)(g)\cup A(\Sigma)(g')\cup 1_X}(\psi) \wedge \nu^{\mathscr{F}}_{\Sigma,S,g'}(\varrho)$
- $\nu^{\mathscr{F}}_{\Sigma,S,g}(\neg\varrho) = \neg\nu^{\mathscr{F}}_{\Sigma,S,g}(\varrho)$
- $\nu^{\mathscr{F}}_{\Sigma,S,g}(\varrho_1 \vee \varrho_2) = \nu^{\mathscr{F}}_{\Sigma,S,g}(\varrho_1) \vee \nu^{\mathscr{F}}_{\Sigma,S,g}(\varrho_2)$

Example 10. The translation of $(@s1)\langle\!|\mathsf{inc}(x) : \mathsf{cnt} + x \leq 4 /\!\!/ \mathsf{cnt}' = \mathsf{cnt} + x|\!\rangle s1$ over the state set $\{s1\}$ and the configuration variable g is

$$
\begin{aligned}
& \nu^{\mathscr{F}}_{\Sigma,\{s1\},g}((@s1)\langle\!|\mathsf{inc}(x) : \mathsf{cnt} + x \leq 4 /\!\!/ \mathsf{cnt}' = \mathsf{cnt} + x|\!\rangle s1) \\
= \; & \forall g' : \mathrm{Conf} \,.\, (\mathrm{c}(g') = s1 \wedge \mathrm{reachable}(g')) \Rightarrow \\
& \quad \nu^{\mathscr{F}}_{\Sigma,\{s1\},g'}(\langle\!|\mathsf{inc}(x) : \mathsf{cnt} + x \leq 4 /\!\!/ \mathsf{cnt}' = \mathsf{cnt} + x|\!\rangle s1) \\
= \; & \forall g' : \mathrm{Conf} \,.\, (\mathrm{c}(g') = s1 \wedge \mathrm{reachable}(g')) \Rightarrow \\
& \quad \forall x : dt \,.\, \mathrm{cnt}(g') + x \leq 4 \Rightarrow \\
& \quad\quad \exists g'' : \mathrm{Conf} \,.\, \mathrm{trans}(g', \mathrm{inc}(x), g'') \wedge \\
& \quad\quad\quad \mathrm{cnt}(g'') = \mathrm{cnt}(g') + x \wedge \nu^{\mathscr{F}}_{\Sigma,\{s1\},g''}(s1) \\
= \; & \forall g' : \mathrm{Conf} \,.\, (\mathrm{c}(g') = s1 \wedge \mathrm{reachable}(g')) \Rightarrow \\
& \quad \forall x : dt \,.\, \mathrm{cnt}(g') + x \leq 4 \Rightarrow \\
& \quad\quad \exists g'' : \mathrm{Conf} \,.\, \mathrm{trans}(g', \mathrm{inc}(x), g'') \wedge \\
& \quad\quad\quad \mathrm{cnt}(g'') = \mathrm{cnt}(g') + x \wedge s1 = \mathrm{c}(g'')
\end{aligned}
$$

Building on the translation of formulæ, the sentence translation

$$\nu^{\mathrm{Sen}}_{\Sigma} : \mathrm{Sen}^{\mathcal{M}^{\downarrow}_{\mathcal{D}}}(\Sigma) \to \mathrm{Sen}^{\text{CASL}}(\nu^{Sig}(\Sigma))$$

only has to require additionally that evaluation starts in an initial state:

- $\nu^{\mathrm{Sen}}_{\Sigma}(\rho) = \forall g : \mathrm{Conf} \,.\, \mathrm{init}(g) \Rightarrow \nu^{\mathscr{F}}_{\Sigma,\emptyset,g}(\rho)$

The translation of CASL-models of TRANS_{Σ} into $\mathcal{M}^{\downarrow}_{\mathcal{D}}$-structures and the translation of $\mathcal{M}^{\downarrow}_{\mathcal{D}}$-formulæ into CASL-formulæ over TRANS_{Σ} fulfil the requirements of the "open" satisfaction condition of theoroidal comorphisms:

```
logic UMLState
spec Counter =
  var cnt;
  event inc(x);
  event reset;
  states s1, s2;
  init s1 : cnt = 0;
  trans s1 --> s1 : inc(x) [cnt + x < 4] / { cnt := cnt + x };
  trans s1 --> s2 : inc(x) [cnt + x = 4] / { cnt := cnt + x };
  trans s2 --> s1 : reset [cnt = 4] / { cnt := 0 };
end
```

Lst. 1. Representation of the simple UML state machine *Counter* in UMLState

Lemma 2. *For a* $\varrho \in \mathscr{F}^{\mathcal{M}^{\downarrow}_{\mathcal{D}}}_{\Sigma,S}$*, an* $M' \in \mathrm{Mod}^{\text{CASL}}(\nu^{\mathbb{S}}(\Sigma))$*, a* $v\colon S \to C(\nu^{\mathrm{Mod}}_{\Sigma}(M'))$*, and a* $\gamma \in \Gamma(\nu^{\mathrm{Mod}}_{\Sigma}(M'))$ *it holds with* $\beta'_{M',g}(v,\gamma) = \iota^{-1}_{M',\mathrm{Ctrl}} \circ v \cup \{g \mapsto \iota_{M',\mathrm{Conf}}(\gamma)\}$ *that*

$$\nu^{\mathrm{Mod}}_{\Sigma}(M'), v, \gamma \models^{\mathcal{M}^{\downarrow}_{\mathcal{D}}}_{\Sigma,S} \varrho \iff M', \beta'_{M',g}(v,\gamma) \models^{\text{CASL}}_{\nu^{Sig}(\Sigma),S\cup\{g\}} \nu^{\mathscr{F}}_{\Sigma,S,g}(\varrho)\,.$$

Theorem 2. $(\nu^{\mathbb{S}}, \nu^{\mathrm{Mod}}, \nu^{\mathrm{Sen}})$ *is a theoroidal comorphism from* $\mathcal{M}^{\downarrow}_{\mathcal{D}}$ *to* CASL.

6 Proving Properties of UML State Machines with HETS and SPASS

We implemented the translation of simple UML state machines into CASL specifications within the heterogeneous toolset HETS [15]. Based on this translation we explain how to prove properties symbolically in the automated theorem prover SPASS [20] for our running example of a counter.

6.1 Implementation in HETS

For a HETS chain from simple UML state machine to CASL and SPASS, we first defined the input language UMLState and extended HETS with a parser for this language. The syntax of UMLState closely follows the ideas of PlantUML [18], such that, in particular, its textual specifications can potentially be rendered graphically as UML state machines. Listing 1 gives a representation of our running example *Counter*, cf. Sect. 3, in UMLState. Note that UMLState uses more conventional UML syntax for effects on transitions, e.g., "`cnt := cnt` + x". Next, we extended HETS with a syntax representation of our logic $\mathcal{M}^{\downarrow}_{\mathcal{D}}$, cf. Sect. 4, and implemented Alg. 1 in HETS to automatically translate UMLState specifications into $\mathcal{M}^{\downarrow}_{\mathcal{D}}$ specifications, where we arrive at the institutional level. In this step, effects on transitions are turned into logical formulæ, like "$\mathsf{cnt}' = \mathsf{cnt} + x$". Finally, we extended HETS with an implementation of the comorphism from $\mathcal{M}^{\downarrow}_{\mathcal{D}}$ into CASL, cf. Sect. 5. The implementation has been bundled in a fork of HETS (`https://github.com/spechub/hets`) and provides a translation chain from UMLState via CASL to the input languages of various proof tools, such as the automated theorem prover SPASS.

spec COUNTER = TRANS
then pred $invar(g : Conf) \Leftrightarrow (c(g) = s1 \wedge cnt(g) \leq 4) \vee (c(g) = s2 \wedge cnt(g) \leq 4)$
%% induction scheme for "reachable" predicate, instantiated for "invar":
$\cdot ((\forall\ g : Conf \cdot init(g) \Rightarrow invar(g))$
$\wedge \forall\ g, g' : Conf; e : Evt$
$\cdot (reachable(g) \Rightarrow invar(g)) \wedge reachable(g) \wedge trans(g, e, g') \Rightarrow invar(g'))$
$\Rightarrow \forall\ g : Conf \cdot reachable(g) \Rightarrow invar(g)$
then ... *machine axioms* ...
then %implies
%% the safety assertion for our counter:
$\forall\ g : Conf \cdot reachable(g) \Rightarrow cnt(g) \leq 4$ %(Safe)%
%% steering SPASS with case distinction lemmas, could be generated algorithmically:
$\forall\ g, g' : Conf; e : Evt; k : Nat$
$\cdot\ init(g) \Rightarrow invar(g)$ %(InvarInit)%
$\cdot (reachable(g) \Rightarrow invar(g)) \wedge reachable(g) \wedge trans(g, e, g') \wedge e = reset$
$\Rightarrow invar(g')$ %(InvarReset)%
$\cdot (reachable(g) \Rightarrow invar(g)) \wedge reachable(g) \wedge trans(g, e, g') \wedge e = inc(k)$
$\Rightarrow invar(g')$ %(InvarInc)%
$\cdot (reachable(g) \Rightarrow invar(g)) \wedge reachable(g) \wedge trans(g, e, g')$
$\Rightarrow invar(g')$ %(InvarStep)%
$\cdot\ invar(g) \Rightarrow cnt(g) \leq 4$ %(InvarImpliesSafe)%

Fig. 4. CASL specification of our running example

6.2 Proving in SPASS

Figure 4 shows the CASL specification representing the state machine from Fig. 2, extended by a proof obligation %(Safe)% and proof infrastructure for it. We want to prove the safety property that `cnt` never exceeds 4 using the automated theorem prover SPASS.

The CASL specification COUNTER imports a specification TRANS which instantiates the generic frame translating $\mathcal{M}^{\downarrow}_{\mathcal{D}}$ into CASL, cf. Fig. 3. However, the first-order theorem prover SPASS does not support CASL's structured free that we use for expressing reachability. For invariance properties this deficiency can be circumvented by loosely specifying *reachable* (i.e., omitting the keyword `free`), introducing a predicate *invar*, and adding a first-order induction axiom. This means that we have to establish the safety property for a larger model class than we would have with freeness. When carrying out symbolic reasoning for invariant referring to a single configuration, the presented induction axiom suffices. Other properties would require more involved induction axioms, e.g., referring to several configurations.

Then the specification provides the machine axioms as stated (partially) in Ex. 9. The axioms following the *%implies* directive are treated as proof obligations. We first state the safety property that we wish to establish: in all reachable configurations, the counter value is less or equal 4 – %(Safe)%. The remainder steers the proving process in SPASS by providing suitable case distinctions. For invariants referring to a single configuration, these could also be generated automatically based on the transition structure of the state machine.

As proof of concept, we automatically verified this safety property in SPASS. In this experiment, we performed some optimising, semantics-preserving logical transformations on the result of applying the comorphism, to make the specification more digestible to the theorem prover. These transformations include the removal of double negations, splitting a conjunction into separate axioms, and turning existentially quantified control states into constants by Skolemisation.

7 Conclusions and Future Work

We have described a new, institution-based logical framework $\mathcal{M}^{\downarrow}_{\mathcal{D}}$ that captures simple UML state machines. This is in contrast to previous approaches that modelled UML parts directly as an institution and ran into difficulties in establishing the satisfaction condition [19]. By (1) defining an institution-based translation from $\mathcal{M}^{\downarrow}_{\mathcal{D}}$ into the CASL institution and (2) implementing and thus automatising our translation within HETS, we made it possible to analyse UML state machines with the broad range of provers accessible via HETS.

The resulting tool chain allows us to apply an automatic prover (as demonstrated here using the theorem prover SPASS), or several automatic provers, where they work and switch to interactive tools like Isabelle where necessary (not needed in the analysis of our example $Counter$). Not only does this switch require no manual reformulation into the interactive tool's input language, rather, it can be done even within one development: We could possibly show some lemmas via automatic first-order provers, some lemmas via domain-specific tools, then use those to prove a difficult lemma in an interactive prover, then apply all those lemmas to automatically prove the final theorem. HETS allows us to use the best language and the best tool for each job, and takes care of linking the results together under the hood.

It is future work to extend $\mathcal{M}^{\downarrow}_{\mathcal{D}}$ to cover more elements of UML state machines, such as hierarchical states and communication networks. The main challenge here will be to enrich $\mathcal{M}^{\downarrow}_{\mathcal{D}}$ in such a way that it offers suitable logical representations for the additional structural elements (hierarchical states or communication networks) rather than to flatten these: We anticipate symbolic reasoning on UML state machines to be "easier" if their structural elements are still "visible" in their CASL representations.

In the long term, we work towards heterogeneous verification of different UML diagrams. One possible setting would be to utilise interactions as a specification mechanism, where communicating state machines model implementations.

References

1. Balser, M., Bäumler, S., Knapp, A., Reif, W., Thums, A.: Interactive Verification of UML State Machines. In: Davies, J., Schulte, W., Barnett, M. (eds.) Proc. 6th Intl. Conf. Formal Engineering Methods (ICFEM 2004). Lect. Notes Comp. Sci., vol. 3308 (2004)
2. Blackburn, P., de Rijke, M., Venema, Y.: Modal Logic, Cambridge Tracts in Theoretical Computer Science, vol. 53. Cambridge University Press (2001)
3. Diaconescu, R., Madeira, A.: Encoding Hybridized Institutions into First-order Logic. Math. Struct. Comp. Sci. **26**(5), 745–788 (2016)

4. Goguen, J.A., Burstall, R.M.: Institutions: Abstract Model Theory for Specification and Programming. J. ACM **39**, 95–146 (1992)
5. Grönniger, H.: Systemmodell-basierte Definition objektbasierter Modellierungssprachen mit semantischen Variationspunkten. Ph.D. thesis, RWTH Aachen (2010)
6. Hennicker, R., Madeira, A., Knapp, A.: A Hybrid Dynamic Logic for Event/Data-Based Systems. In: Hähnle, R., van der Aalst, W.M.P. (eds.) Proc. 22nd Intl. Conf. Fundamental Approaches to Software Engineering. Lect. Notes Comp. Sci., vol. 11424, pp. 79–97. Springer (2019)
7. James, P., Knapp, A., Mossakowski, T., Roggenbach, M.: Designing Domain Specific Languages — A Craftsman's Approach for the Railway Domain Using Casl. In: Martí-Oliet, N., Palomino, M. (eds.) Rev. Sel. Papers 21st Intl. Ws. Recent Trends in Algebraic Development Techniques (WADT 2012). Lect. Notes Comp. Sci., vol. 7841, pp. 178–194. Springer (2012)
8. Knapp, A., Mossakowski, T.: UML Interactions Meet State Machines — An Institutional Approach. In: Bonchi, F., König, B. (eds.) Proc. 7th Intl. Conf. Algebra and Coalgebra in Computer Science (CALCO 2017). LIPIcs, vol. 72, pp. 15:1–15:15 (2017)
9. Knapp, A., Mossakowski, T., Roggenbach, M.: Towards an Institutional Framework for Heterogeneous Formal Development in UML — A Position Paper. In: De Nicola, R., Hennicker, R. (eds.) Software, Services, and Systems — Essays Dedicated to Martin Wirsing on the Occasion of His Retirement from the Chair of Programming and Software Engineering, Lect. Notes Comp. Sci., vol. 8950, pp. 215–230. Springer (2015)
10. Knapp, A., Mossakowski, T., Roggenbach, M., Glauer, M.: An Institution for Simple UML State Machines. In: Egyed, A., Schaefer, I. (eds.) Proc. 18th Intl. Conf. Fundamental Approaches to Software Engineering (FASE 2015). Lect. Notes Comp. Sci., vol. 9033, pp. 3–18. Springer (2015)
11. Kyas, M., Fecher, H., de Boer, F.S., Jacob, J., Hooman, J., van der Zwaag, M., Arons, T., Kugler, H.: Formalizing UML Models and OCL Constraints in PVS. In: Lüttgen, G., Mendler, M. (eds.) Proc. Ws. Semantic Foundations of Engineering Design Languages (SFEDL 2004). Electr. Notes Theo. Comp. Sci., vol. 115 (2005)
12. Madeira, A.: Foundations and Techniques for Software Reconfigurability. Ph.D. thesis, Universidade do Minho (2013)
13. Madeira, A., Barbosa, L.S., Hennicker, R., Martins, M.A.: Dynamic Logic with Binders and Its Application to the Development of Reactive Systems. In: Proc. 13th Intl. Coll. Theoretical Aspects of Computing. Lect. Notes Comp. Sci., vol. 9965, pp. 422–440. Springer (2016)
14. Mossakowski, T.: Relating CASL with Other Specification Languages: The Institution Level. Theo. Comp. Sci. **286**(2), 367–475 (2002)
15. Mossakowski, T., Maeder, C., Lüttich, K.: The Heterogeneous Tool Set. In: Grumberg, O., Huth, M. (eds.) Proc. 13th Intl. Conf. Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2007). Lect. Notes Comp. Sci., vol. 4424, pp. 519–522. Springer (2007)
16. Mosses, P.D.: CASL Reference Manual — The Complete Documentation of the Common Algebraic Specification Language, Lect. Notes Comp. Sci., vol. 2960. Springer (2004)
17. Object Management Group: Unified Modeling Language. Standard formal/17-12-05, OMG (2017), `http://www.omg.org/spec/UML/2.5.1`
18. Roques, A.: PlantUML. `https://plantuml.com/` (Accessed 2020-02-11)
19. Rosenberger, T.: Relating UML State Machines and Interactions in an Institutional Framework. Master's thesis, Elite Graduate Program Software Engineering (Universität Augsburg, Ludwig-Maximilians-Universität München, Technische Universität München) (2017)
20. Weidenbach, C., Dimova, D., Fietzke, A., Kumar, R., Suda, M., Wischnewski, P.: SPASS Version 3.5. In: Schmidt, R.A. (ed.) Proc. 22nd Intl. Conf. Automated Deduction. Lect. Notes Comp. Sci., vol. 5663, pp. 140–145. Springer (2009)