Consistency of Heterogeneously Typed Behavioural Models: A Coalgebraic Approach

Harald König[1,2] and Uwe Wolter[3]

[1] University of Applied Sciences, FHDW, Hannover, Germany
harald.koenig@fhdw.de
[2] Høgskulen på Vestlandet, Bergen, Norway
[3] University of Bergen, Norway
uwe.wolter@uib.no

Abstract. Systematic and formally underpinned consistency checking of heterogeneously typed interdependent behavioural models requires a common metamodel, into which the involved models can be translated. And, if additional system properties are imposed on the behavioural models by modal logic formulae, the question arises, whether these formulae are faithfully translated, as well.
In this paper, we propose a formal methodology based on natural transformations between coalgebraic specifications, which enables state-space preserving translations into a category of homogeneously typed systems, and we determine mild assumptions for the transformations to guarantee preservation and reflection of truth of translated formulae.

Keywords: Heterogeneous Behavioural Models, Coalgebra, Reactive System, Modal Logic, Category Theory

1 Introduction and Motivation

In model-based software projects, heterogenous(ly structured) but interdependent behavioural models can occur. These models may, however, prescribe the same (or overlapping parts of the same) real-world behaviour: A class diagram may prescribe domain services, a BPMN [4] diagram may model a process, in which these services are invoked (in a certain sequence). Although the behaviour of these systems is based on states and state changes, the concrete stimuli and effects of state changes are different. Beside the above example, there are labelled transition systems with or without output (per state or per transition), deterministic or non-deterministic, possibly timed or probabilistic.

Model checking is an automated technique that, given a finite-state model of a system and a formal property, systematically checks whether this property holds for (a given state or computation path in) that model. Usually, the property is given in terms of a formula in modal logic. Automatic verification of this formula is carried out on a homogeneously structured transition structure,

[4] https://www.omg.org/spec/BPMN/2.0/PDF

which is derived from the behavioural structure of a system, usually a Kripke Frame. To consequently model check a complete ensemble of interacting, but *heterogeneously structured* artefacts, formulas of modal logic can only formally be imposed on them, if the different involved types of transition structures are translated into a homogeneous transition structure. We call a formula, which spreads over different systems an *inter-model constraint*. In the sequel, we will consider such an ensemble of heterogeneously typed transition structures. Formulas can be imposed on single *local* components or they are *global* inter-model constraints.

In this paper, we propose a formalism for the translation of components' shapes into a common transition structure, enabling uniform formal reasoning about their interaction. The translation will not alter the state space, but only the transition structure. When translating local components' behaviour into a common formalism, formulas imposed on the local component, e.g. liveness or termination properties, must also be translated. Moreover, these local constraints interact with the global inter-model constraints: They might contradict each other or the former is a logical consequnce of the latter, etc. Thus, the following *research questions* arise:

1. How can we formally define the translation into a common type of transition structure?
2. Can we expect preservation (and reflection) of formula validity during a translation? If not, which requirements must the translation fulfill for formula validity invariance?

We will use *coalgebras* for the all-embracing metamodel of reactive systems. We think that this is the consequent continuation of the theory of *institutions* [19], where algebraic specifications (understood as endofunctors $\mathcal{F}$), specification morphisms (natural transformations between functors), and logical formulas enable a comprehensive view on heterogeneous data structures. It is an old insight [18] that dualisation of the structure maps of algebras enables an elegant description of behavioural systems instead of data structures and that logical formulas in the algebraic settings are replaced by formulas of a modal logic. Finally, coalgebras enable a comprehensive view on heterogeneous behavioural structures.

In this paper, we present the following novelties:

– We show how to synchronize reactive systems of different behavioural specifications in general. For this, we use coalgebras and corresponding specification morphisms.
– We provide a criterion characterizing preservation and reflection of validity of modal logic formulas during translation along specification morphisms.

The investigated temporal operators are based on predicates imposed on state spaces, i.e. we emphasize more the branching-time perspective of CTL (Computation Tree Logic) than the linear-time perspective of LTL (Linear Temporal Logic). For a comparison of these two logics, see [1].

Section 2 presents a practical problem, which has already been elaborated in a similar form in [11] and which is picked up in the paper each time a theoretical result must be illustrated. Section 3 reports on the necessary background and Section 4 presents the main results (Proposition 1 and 2) and applies them to the running example. Sections 5 and 6 conclude the paper.

2 Running Example

The process of fixing bugs, which have been reported as tickets by users of a software application, may be captured in a BPMN diagram, which models the ticket handling workflow. Automatic activities in this workflow rely on the existence of services provided by a backend system, seeing Activity ”Analyse Ticket Database” in Fig. 1.

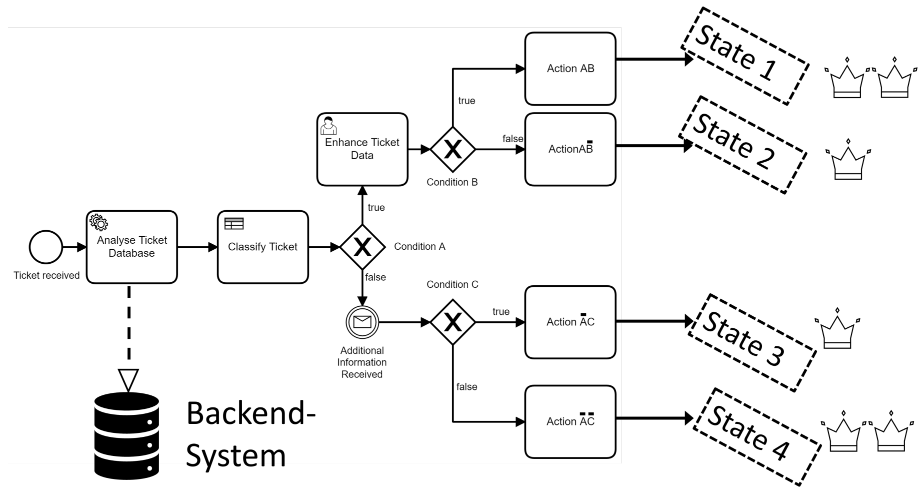

Fig. 1. BPMN Model with different decision points

Abstraction in BPMN Models: For the sake of simplicity we assume that each service task requests exactly one method of the backend system and that the output of the method call does not directly influence the process instance's state, e.g. the (business rule) task ”Classify Ticket” is always the successor of ”Analyse Ticket Database” independent of the output of the invoked method in the latter task.

The further activities in Fig. 1 are as follows:

– The automatic activity ”Analyse Ticket Database” invokes a method in the backend system, which exploits a knowledge database, in order to provide (semi-)automatically a solution for the current ticket. Depending on this data and the ticket classification, evaluation of *Condition A* decides, which of the following two branches is chosen.

- In the upper branch, a user activity enhances possibly missing information in the ticket. In the lower branch, more information is received by an intermediate catching event.
- Evaluation of *Condition B* and *Condition C* depend on this additional data and trigger one of the four activities $\{AB, A\overline{B}, \overline{A}C, \overline{A}\,\overline{C}\}$, which may or may not be automatic tasks and which may update the backend system with new information.

Consequently, the state of the data in the backend system influences state changes of the process instance and vice versa, thus:

The behaviours of the backend system and the workflow management system mutually depend on each other.

The software application may be run by customers of a software company, who sells this software, or it may as well be a ready-to-use solution built by a company on its own. In either case, service-level agreements (SLA) are offered in order to ensure the quality of bug fixing for the users. A typical agreement prescribes solutions to be provided as a software patch, which automatically updates the current software version and removes the reported bug (high-quality solution). In contrast, a low-quality solution is a temporary workaround being carried out by the user, for example, a change in the configuration of the application.

Often these promises are subject to certain preconditions: A software patch will be provided, only if the user's software runs in a certain mature version (and not in a recently offered ”Alpha”-version). This information may or may not be present prior to process start: If the information is not already stored in the backend system, it has to be retrieved by further inquiries, e.g. ”Enhance Ticket Data”, in Fig. 1.

Let us assume that a high-quality solution like the patch provisioning is present, if one of the actions AB or $\overline{A}\,\overline{C}$ has terminated, two crowns at state 1 and 4 in Fig. 1 (a single crown indicates low-quality solutions). It is then a goal for the software provider to guarantee his promise by using a formal model checking procedure. For this, a predicate V is defined which holds in a certain state, if the given software **V**ersion of the ticket reporter is some mature version. Furthermore, one can define a predicate H, which can be imposed on states of BPMN processes indicating the possibility to provide a **H**igh-Quality Solution. Since the process is not yet finished in states 1 to 4, predicate H could be invalidated due to other activities or events, hence we formulate the SLA by

”Inter-Model Constraint” φ: If the system is in a state with property V, then for each computation path, we eventually reach a state, from which henceforth property H holds.

Of course, this formula interacts with already given formulas on the local systems for example termination or liveness requirements of the BPMN-workflow. If ψ is such a formula, one has to ensure for instance, that φ and ψ are not contradictory. Moreover, if ψ is a logical consequence of φ, it is not required to be checked, but can be considered to be fulfilled, if φ holds.

The goal is now to use established model checkers to prove validity of the involved temporal formulas. In order to check validity of inter-model-constraints, one has to define a comprehensive state space of Backend System and BPMN process models especially taking into account their interactions (see above). We face two problems:

1. How can we formally define this comprehensive state space?
2. Shall validity of a local constraint ψ be checked on the local state space or on the comprehensive state space? Can we expect to obtain the same result?

A challenge is to cope with *private* and *common* features of these two transition structures: A feature is private, if it is known only to one of the structures and not known to the other, e.g. the backend system knows nothing about incoming events and roles (not shown in Fig. 1) of the BPMN process, the BPMN process does not consider outputs of the backend system's methods. An exception is the output of a condition's evaluation, which we can, however, interpret as an event, which lets an event-based gateway decide the alternative. Finally, common behaviour of both systems are states and method-call-triggered state changes (requests in the BPMN tasks and invocations of methods in the backend system).

In the next sections, we show how to translate different types of transition structures into an appropriate common transition structure. We illustrate our general approach by encoding (1) stateful backend systems and (2) BPMN process models as coalgebras.

3 Background

3.1 Notation

We use the following notations: $\mathcal{SET}$ is the category of sets and total mappings. For two sets X and Y we write Y^X for the set of all total maps from X to Y. Special sets are $1 = \{*\}$ (any singleton set) and $2 = \{\texttt{true}, \texttt{false}\}$. Instead of the set of all partial maps $A \overset{f}{\rightharpoonup} B$ between two sets A, B we consider the set of all total maps $A \xrightarrow{f} 1 + B$, i.e. $f(x)$ is undefined if and only $f(x) = *$. Both sets are isomorphic in $\mathcal{SET}$.

$$\begin{tikzcd} X \arrow[r, "f"] & Y \end{tikzcd}$$

$$\begin{tikzcd} \mathcal{F}X \arrow[r, "\mathcal{F}f"] \arrow[d, "\eta_X"] & \mathcal{F}Y \arrow[d, "\eta_Y"] \\ \mathcal{G}X \arrow[r, "\mathcal{G}f"'] & \mathcal{G}Y \end{tikzcd}$$

Fig. 2. Naturality Square

For functors between different categories we will use calligraphic letters like $\mathcal{F}$, $\mathcal{G}$, or $\mathcal{H}$. $\mathcal{ID}$ is the identity functor on $\mathcal{SET}$. Application of a functor $\mathcal{F} : \mathbb{C} \to \mathbb{D}$ will always be written without parentheses, e.g. $\mathcal{F}X$ (for objects) and $\mathcal{F}f$ (for morphisms). The power set functor is $\wp : \mathcal{SET} \to \mathcal{SET}$. $\wp_{fin} : \mathcal{SET} \to \mathcal{SET}$ will denote the functor assigning to a set the set of its *finite* subsets. For $X \xrightarrow{f} Y$ and a subset $A \subseteq X$, defined by a condition ϕ, we write

$$f(A) := \{f(x) \in Y \mid \phi(x)\} \text{ if } A = \{x \in X \mid \phi(x)\},$$

which is the application of $\wp f$ to A. To distinguish f from this counterpart, we often write $f(\_)$ for $\wp f$ and likewise for $\wp_{fin} f$.

Diagrams always depict commutative diagrams, e.g. the square in Fig. 2 is automatically assumed to be commutative, specifying the condition for a natural transformation $\eta : \mathcal{F} \Rightarrow \mathcal{G}$, i.e. a family $(\eta_X)_{X \in |\mathbb{C}|} : \mathcal{F}X \to \mathcal{G}X$ of $\mathbb{D}$ morphisms indexed by $\mathbb{C}$'s object collection (which we denote by $|\mathbb{C}|$) if $\mathcal{F}, \mathcal{G} : \mathbb{C} \to \mathbb{D}$.

3.2 Coalgebras

The investigation of *heterogenous*(ly typed) reactive systems requires a meta-model, which captures as many behavioural specifications as possible. A behavioural "specification" describes the way a system interacts with the environment. For deterministic labeled transition systems (DLTS) over an alphabet A, this specification is the set $(1+X)^A$, because for each system state a partial map assigns to an event $a \in A$ (from the environment) at most one follow-up state. In contrast to that, non-deterministic finitely-branching systems (NLTS) are based on an assignment $x \mapsto c(x) \in \wp_{fin}(A \times X)$ for all states x, i.e. $(a, y) \in c(x)$ means that in state x the event a may cause a state change from x to y. [5]

We obtain a common template for encoding different types of transition structures: They can formally be described by an assignment $\mathcal{F} : |\mathcal{SET}| \to |\mathcal{SET}|$, e.g. $X \mapsto (1+X)^A$ for DLTS or $X \mapsto \wp_{fin}(A \times X)$ for NLTS. Analogously, one can find similar assignments for all other types of transition structures. Moreover, in all cases, cf. [18], these assignments extend to functors $\mathcal{F} : \mathcal{SET} \to \mathcal{SET}$.

Example 1 (LTS) *The functor*

$$\mathcal{G} : \begin{cases} \qquad\qquad \mathcal{SET} \longrightarrow \mathcal{SET} \\ X \xrightarrow{f} Y \quad \mapsto \quad (1+X)^A \xrightarrow{(id_1 + f) \circ \_} (1+Y)^A \end{cases}$$

encodes DLTS and NLTS are specified by the functor

$$\wp_{fin}(A \times \_) : \begin{cases} \qquad\qquad \mathcal{SET} \longrightarrow \mathcal{SET} \\ X \xrightarrow{f} Y \quad \mapsto \quad \wp_{fin}(A \times X) \xrightarrow{(id_A \times f)(\_)} \wp_{fin}(A \times Y) \end{cases}$$

Example 2 (Modules of Object-Orientated Programs, [9,17]) *A package or a module of classes in an object-oriented environment with n visible methods $(m_j(x : I_j) : O_j)_{j=1..n}$ in its facade can be encoded as a coalgebra for the functor $\mathcal{F}_1$, which maps a set X to the Cartesian product of the family $((O_j \times X)^{I_j})_{j=1..n}$ of sets of maps*

$$\mathcal{F}_1 X = \prod_{j=1}^{n} (O_j \times X)^{I_j}.$$

[5] Other examples of reactive systems are finite or infinite streams, automata with output (i.e. UML state charts), activity diagrams or BPMN diagrams (with or without guard conditions) and probablistic or timed automata [18].

An $\mathcal{F}_1$-system $c : X \to \mathcal{F}_1 X$ represents a tuple $(m_1, ..., m_n)$ of maps, in which $m_j(x)$ is the application of method m_j for input $i \in I_j$. Depending on state x, it produces an output $o \in O_j$ and a new state $x' \in X$.

Generalizing Example 1 and 2, we define

Definition 1 ($\mathcal{F}$-Coalgebra). *Let $\mathcal{F} : \mathcal{SET} \to \mathcal{SET}$ be a functor. An $\mathcal{F}$-coalgebra (X, c) or $\mathcal{F}$-system is a map* $X \xrightarrow{c} \mathcal{F}X$.

In the context of coalgebras, $\mathcal{F}$ is called a (specification of a) transition structure *and c is the (transition)* structure map.

Furthermore, $\mathcal{F}$-coalgebras constitute themselves a category: A *homomorphism* between $\mathcal{F}$-coalgebras (X, c) and (Y, d) is a map $h : X \to Y$, for which

$$d \circ h = \mathcal{F}h \circ c.$$

Note that homomorphisms not only preserve, but also reflect the transition structure: The graph $\{(x, h(x)) \mid x \in X)\}$ of h yields a bisimilarity on $X \times Y$ [18].

It is then easy to see that we can give the following definition, see also [18].

Definition 2 (Category of $\mathcal{F}$-Coalgebras). *The category $\mathcal{F}$-Coalg has objects $\mathcal{F}$-coalgebras, see Def. 1, and morphisms the $\mathcal{F}$-coalgebra homomorphisms. Identities are identical maps id_X and composition is composition of set maps.*

The existence of initial objects in categories of algebras yields many important insights such as the principle of induction, initial semantics and term generation [22]. Dually, it is important that categories of $\mathcal{F}$-coalgebras possess a *final* object.[6] Corresponding resulting aspects are the principles of final semantics: The unique arrow from a coalgebra into the final object usually assigns to each state its behavioural semantics w.r.t. bisimilarity [10], for coalgebras with no proper quotient one obtains the principle of coinduction, which is a template for recursive implementations of algorithms e.g. on streams [18] etc.

Not every functor $\mathcal{F}$ yields reasonable coalgebras, especially for practical purposes in computer science, because there may not be a final object in $\mathcal{F}$-Coalg. A prominent example is the power set functor $X \mapsto \wp(X)$. Using Lambek's Theorem ("If $\mathcal{F}$-Coalg possesses a final object (Z, ζ), then ζ is a bijection.", see [10], Lemma 2.3.3.) and Cantor's diagonal argument ($X \ncong \wp(X)$ for all sets X), it is clear that there is no final object in $\wp$-Coalg. It is, however, possible to show that this deficit vanishes, if one restricts to the functor $\wp_{fin}(\_)$, see 3.1.

Because of these natural restrictions, we will consider the following restricted collection of $\mathcal{SET}$-endofunctors, whose respective category of coalgebras can be shown to possess a final object [18], and which are sufficient to deal with all important types of transition structures in computer science:

Definition 3 (Kripke Polynomial Functors - $\mathcal{KPF}$, [10]). *The collection $\mathcal{KPF}$ of $\mathcal{SET}$-endofunctors is defined inductively as follows:*

[6] An initial / a final object in a category $\mathbb{C}$ is an object $0/1$, for which there is exactly one morphism $0 \to X$ / exactly one morphism $X \to 1$ for all $X \in |\mathbb{C}|$.

(1) $\mathcal{ID}$ is in $\mathcal{KPF}$

(2) The functor $Const_A$ defined by $Const_A(X \xrightarrow{f} Y) = A \xrightarrow{id_A} A$ is in $\mathcal{KPF}$ for each set A.

(3) If $\mathcal{F}_1, \mathcal{F}_2 \in \mathcal{KPF}$, so is the functor $\mathcal{F}_1 \times \mathcal{F}_2$ defined by

$$(\mathcal{F}_1 \times \mathcal{F}_2)(X \xrightarrow{f} Y) = \mathcal{F}_1 X \times \mathcal{F}_2 X \xrightarrow{\mathcal{F}_1 f \times \mathcal{F}_2 f} \mathcal{F}_1 Y \times \mathcal{F}_2 Y \ .$$

(4) If I is an index set and in an I-indexed collection $(\mathcal{F}_i)_{i \in I}$, all $\mathcal{F}_i$ are in $\mathcal{KPF}$, then so is the functor $\coprod_{i \in I} \mathcal{F}_i$ defined by [7]

$$(\textstyle\coprod_{i \in I} \mathcal{F}_i)(X \xrightarrow{f} Y) = \textstyle\coprod_i \mathcal{F}_i X \xrightarrow{\coprod_i \mathcal{F}_i f} \textstyle\coprod_i \mathcal{F}_i Y \ .$$

(5) If A is a set and $\mathcal{F} \in \mathcal{KPF}$, then so is the functor $\mathcal{F}^A$ defined by

$$\mathcal{F}^A(X \xrightarrow{f} Y) = (\mathcal{F}X)^A \xrightarrow{\mathcal{F}f \circ \_} (\mathcal{F}Y)^A \ .$$

(6) If $\mathcal{F} \in \mathcal{KPF}$, then so is the functor $\wp_{fin}(\mathcal{F})$ defined by

$$(\wp_{fin}(\mathcal{F}))(X \xrightarrow{f} Y) = \wp_{fin}(\mathcal{F}X) \xrightarrow{(\mathcal{F}f)(\_)} \wp_{fin}(\mathcal{F}Y) \ .$$

It can be shown that automata with output (Moore and Mealy Machines) as well as automata with final states can be encoded using $\mathcal{KPF}$'s, the latter by $Const_2 \times \mathcal{F}$ for an arbitrary $\mathcal{F}$ in $\mathcal{KPF}$.

From now on, we always assume the involved functors to be contained in $\mathcal{KPF}$. Moreover, we use the following shorthand notation for an $\mathcal{F}$-coalgebra: For any $x \in X$ and $y \in c(x)$, we write $x \rightsquigarrow y$ to indicate the possibility for x to transition to y due to structure map c. If an alphabet A is involved, this can be extended to

$$x \overset{a}{\rightsquigarrow} y \ ,$$

for $a \in A$, for instance for $\mathcal{F} = \wp_{fin}(A \times \_)$ and an $\mathcal{F}$-coalgebra (X, c) with $(a, y) \in c(x)$.

3.3 Signature Morphisms

The example in Section 2 deals with two different behavioural systems: BPMN diagrams and class diagrams (for which behaviour is described by specifying, how a method application changes the object structure at runtime). In the literature the two different metamodels are also called *signatures*. In universal algebra, signature morphisms are the tool of choice to relate algebras of different signatures. It is an old observation, that (unsorted) signatures can also be encoded

[7] For a family $(A_i)_{i \in I}$ of sets the coproduct (sum) $\coprod_{i \in I} A_i$ denotes the disjoint union of all the sets A_i, $i \in I$.

with $\mathcal{SET}$-endofunctors $\mathcal{T}$, where algebras are maps $\alpha : \mathcal{T}X \to X$, and if categories of algebras are defined in such a way, then it is easy to see that signature morphisms can be encoded as natural transformations between the respective specifying functors [3]. Moreover, these transformations yield - by precomposition - a ”forgetful” functor in the opposite direction between the respective categories of algebras.

For coalgebras, we use the dual approach: Given a natural transformation $\eta : \mathcal{F} \Rightarrow \mathcal{G}$, an $\mathcal{F}$-system (X, c) can be translated by postcomposition

$$\begin{tikzcd} X \arrow[r, "c"] \arrow[rr, bend right, "\eta_X \circ c"'] & \mathcal{F}X \arrow[r, "\eta_X"] & \mathcal{G}X \end{tikzcd}$$

into a $\mathcal{G}$-system $(X, \eta_X \circ c)$. For $\mathcal{F}$-Coalg-homomorphism $h : (X, c) \to (Y, d)$, we obtain $\mathcal{G}h \circ (\eta_X \circ c) = (\eta_Y \circ d) \circ h$ by naturality, thus:

Lemma 1 (Co-Forgetful Functor) *Let $\eta : \mathcal{F} \Rightarrow \mathcal{G}$ be a natural transformation between two set-endofunctors. The assignment $c \mapsto \eta_X \circ c$ extends to a functor*

$$\mathcal{U}_\eta : \begin{cases} \qquad\qquad \mathcal{F}\text{-}Coalg \longrightarrow \mathcal{G}\text{-}Coalg \\ (X, c) \xrightarrow{h} (Y, d) \;\; \mapsto \;\; (X, \eta_X \circ c) \xrightarrow{\qquad h \qquad} (Y, \eta_Y \circ d) \end{cases}$$

Example 3 (Translation of Backend Behaviour to DLTS) *In our running example of Section 2, we can now translate transition structures of the backend system to DLTS. For this we use functor $\mathcal{F}_1$ from Example 2 to encode the available services $m_1, ..., m_n$ (methods) of the backend as an $\mathcal{F}_1$-system. Note that all methods decompose into two projections: $m_j = (m_{j,1}, m_{j,2}) : I_j \to O_j \times X$. Since we want the DLTS to be prepared for additional events, we choose for the functor $\mathcal{G}$ of Example 1*

$$A := I + \coprod_{j=1}^{n} I_j$$

as its input alphabet, where I is an arbitrary set of additional input stimuli. The translation is given by the following family of mappings indexed over $X \in |\mathcal{SET}|$:

$$\eta_X : \begin{cases} \prod_{j=1}^{n} (O_j \times X)^{I_j} \to (1 + X)^A \\ \quad (m_1, ..., m_n) \quad \mapsto \lambda i. \begin{cases} m_{j,2}(i) \;, & \text{if } i \in I_j \text{ for some } j \\ \quad * \quad , & \text{if } i \in I \end{cases} \end{cases}$$

where we denoted the result of η_X by a λ-expression. We omit the easy proof of naturality of $\eta : \mathcal{F}_1 \Rightarrow \mathcal{G}$ but emphasize that the translation along η forgets outputs and enables the behaviour embedding of the backend system in a transition structure with extended input options.

3.4 Predicate Lifting

The heart in the description of temporal operators in the next Section 3.5 is the transformation of truth of a property from one state to its sucessor state(s) by a structure map c in an $\mathcal{F}$-coalgebra (X, c). Truth, however, is based on predicates. If a predicate P like "Eventually a state is reached, which guarantees a high-quality solution." is true on a state x, we write $P(x)$, and we can equivalently describe P as a subset of the state set, namely those states where P holds. That is, the notations

$$P(x), x \in P \text{ and } \ x \models P$$

for $P \subseteq X$ (or equivalently $P \hookrightarrow X$) will synonymously be used. If x satisfies a predicate, we want to reason whether $c(x)$ satisfies this predicate, too. However, $c(x) \in \mathcal{F}X$ is not a single state, but - according to $\mathcal{F}$ - a more complex entity depending on the type of transition structure $\mathcal{F}$.

Hence it is necessary to transform (lift) predicates that are imposed on elements of a fixed set X to predicates on $\mathcal{F}X$. We recall inductively defined predicate lifting from Chapter 6 of [10] for $\mathcal{KPF}$'s: The operator

$$\mathrm{Pred}(\mathcal{F}) : \wp(X) \to \wp(\mathcal{F}X)$$

is defined on $\mathcal{KPF}$'s as follows: For $P \subseteq X$

(1) $\mathrm{Pred}(\mathcal{ID})(P) = P$
(2) $\mathrm{Pred}(Const_A)(P) = A$
(3) $\mathrm{Pred}(\mathcal{F}_1 \times \mathcal{F}_2)(P) = \mathrm{Pred}(\mathcal{F}_1)(P) \times \mathrm{Pred}(\mathcal{F}_2)(P)$
(4) $\mathrm{Pred}(\coprod_{i\in I} \mathcal{F}_i)(P) = \coprod_{i\in I} \mathrm{Pred}(\mathcal{F}_i)(P)$
(5) $\mathrm{Pred}(\mathcal{F}^A)(P) = \{f : A \to \mathcal{F}X \mid \forall a \in A : f(a) \in \mathrm{Pred}(\mathcal{F})(P)\}$
(6) $\mathrm{Pred}(\wp_{fin}(\mathcal{F}))(P) = \{U \subseteq \mathcal{F}X \mid U \subseteq \mathrm{Pred}(\mathcal{F})(P)\}$

We illustrate this definition for DLTS: Let $\mathcal{F}X = (1+X)^A$, then we calculate the lift of predicate $P \subseteq X$ along the syntactical structure of $\mathcal{F}$: $\mathrm{Pred}(1 + \_)(P) = \mathrm{Pred}(Const_1)(P) + \mathrm{Pred}(\mathcal{ID})(P) = 1 + P$ and with that

$$\mathrm{Pred}(\mathcal{F})(P) = \{f : A \to 1 + X \mid \forall a \in A : f(a) \in 1 + P\}$$

i.e. the lifted predicate is true for $f \in (1 + X)^A$, if *all* $f(a) \in P$ or $f(a) = *$, i.e. for each $a \in A$ all successor states, if any, must fulfill the predicate. This can also be expressed by saying that $\mathrm{Pred}(\mathcal{F})(P)$ contains exactly those f, for which $f(A) \subseteq 1 + P = \mathrm{Pred}(1 + \_)(P)$, thus, in this example, $\mathrm{Pred}(\mathcal{F})(P) = (1 + P)^A = \mathcal{F}P$. Indeed, this observation is always true, cf. [10], Lemma 6.1.6.:

Lemma 2 (Predicate Lifting) *Let $\mathcal{F}$ be a $\mathcal{KPF}$, $X \in |\mathcal{SET}|$, then for each predicate $P \overset{m}{\hookrightarrow} X$*

$$\mathrm{Pred}(\mathcal{F})(P) \overset{\mathcal{F}m}{\hookrightarrow} \mathcal{F}X$$

is the inclusion arrow of the lift, i.e., especially, $\mathrm{Pred}(\mathcal{F})(P) = \mathcal{F}P$. □

By structural induction along the definition of $\mathrm{Pred}(\mathcal{F})$ one can also prove:

Lemma 3 (Predicate Lifting is monotone) *In the above setting*

$$P_1 \subseteq P_2 \Rightarrow \mathrm{Pred}(\mathcal{F})(P_1) \subseteq \mathrm{Pred}(\mathcal{F})(P_2)$$

3.5 Temporal Operators

The basic temporal operator is the "next time"-Operator $\bigcirc$. All other operators can be derived from it (of course by using the basic logical operators $\neg$ (negation) and $\wedge$ (conjunction)). A temporal operator depends on a given $\mathcal{F}$-system (X, c) and is usually determined by an operation on subsets P (predicates) of X: If $P \subseteq X$, we denote with $\bigcirc P$ the subset of $\mathcal{F}X$, which contains those states, which reach states in P after a single application of structure map c.

Note that $\bigcirc$ usually depicts a path operator, i.e. $\bigcirc P$ holds for a computation path, if P holds on the second state. Our approach is more general in that it defines this operator for arbitrary transition structures. We will work with the following formal definition:

Definition 4 (Next Time Operator). *Let $\mathcal{F}$ be a $\mathcal{KPF}$ and (X, c) be an $\mathcal{F}$-system. We call*

$$\bigcirc_{\mathcal{F},c} : \begin{cases} \wp(X) \to \wp(X) \\ \quad P \mapsto c^{-1}(\text{Pred}(\mathcal{F})(P)) \end{cases}$$

the Next Time-Operator. *For $x \in X$ we write $x \models_{\mathcal{F},c} \bigcirc P$, if $x \in \bigcirc_{\mathcal{F},c} P$, equivalently, if $c(x) \in \text{Pred}(\mathcal{F})(P)$, or short*

$$x \models \bigcirc P$$

if $\mathcal{F}$ and c are clear from the context.

E.g., for DLTS: $x \models \bigcirc P \iff \forall a \in A, x' \in X : (x \overset{a}{\rightsquigarrow} x' \Rightarrow x' \models P)$.

In the sequel, fixed points of operators on power sets are important. Clearly, a fixed point of an operator $op : \wp(X) \to \wp(X)$ is a subset Q of X, for which $op(Q) = Q$. They are of major importance, if one considers monotone operators (i.e. $P \subseteq Q \Rightarrow op(P) \subseteq op(Q)$) on the boolean algebra $(\wp(X), \subseteq)$, because a consequence of the Theorem of Knaster and Tarski [20] yields

Lemma 4 (Fixed Points) *A monotone operator $op : \wp(X) \to \wp(X)$ possesses a least and a greatest fixed point written $\mu S.op(S)$, $\nu S.op(S)$, resp. Furthermore, if X is a finite set, there is $n_0 \in \mathbb{N}$, such that*

$$\mu S.op(S) = \bigcup_{k=0}^{n_0} op^k(\emptyset) \text{ and } \nu S.op(S) = \bigcap_{k=0}^{n_0} op^k(X).$$

Remark 1 (Finiteness) *We do not formulate the Knaster-Tarski Theorem in its full generality for arbitrary (infinite) sets, because we do not want to deal with the intricacies of approximants in the modal μ-calculus [4].*

$\bigcirc_{\mathcal{F},c}$ is a monotone operator by Lemma 3, and so is the operator $\neg \bigcirc_{\mathcal{F},c} \neg S$ where $\neg S$ denotes set complementation. Hence - from Lemma 4 - we can introduce the following existential path operators: For a fixed $\mathcal{F}$-system (X, c) and subsets P and Q of X

- $\exists\Box P := \nu S.(P \cap \neg \bigcirc \neg S)$ (henceforth)
- $\exists P\,\mathcal{U}\,Q := \mu S.(Q \cup (P \cap \neg \bigcirc \neg S))$ (until)

In words: $x \models \exists\Box P$, if there is a computation path starting at state x, on which P holds forever, $x \models \exists P\,\mathcal{U}\,Q$, if on a path from x, P holds for a while (maybe never) *until* Q holds once.

It is well-known [1] that all important temporal operators (e.g. of CTL) can be derived from these two operators, e.g. the constant `true` (intersection of an empty collection of sets) and further operators like $\exists\Diamond P = \exists\mathtt{true}\,\mathcal{U}\,P$ and with that

$$\forall\Box P := \neg\exists\Diamond\neg P$$

denoting that henceforth on all paths property P holds. Similarly

$$\forall\Diamond P := \neg\exists\Box\neg P$$

means, that for all computation paths, P holds eventually. We write $\Omega_{\mathcal{F},c}$ for any such temporal operator Ω, if the dependency from $\mathcal{F}$ and c is important.

The goal of the next section is to give an answer to both research questions on page 2: We define an appropriate common type of transition structure in order to formally define the *inter-model constraint*

$$\varphi := (\,V \Rightarrow \forall\Diamond(\forall\Box P)\,)$$

on page 4 in Section 2 and show that checking local formulas is independent of the underlying transition structure.

4 Formula Translation

4.1 Truth Preservation

In this section, we investigate how truth can be translated from $\mathcal{F}$-Coalg to $\mathcal{G}$-Coalg with the co-forgetful functor $\mathcal{U}_\eta : \mathcal{F}\text{-Coalg} \to \mathcal{G}\text{-Coalg}$ from Lemma 1 based on a natural transformation $\eta : \mathcal{F} \Rightarrow \mathcal{G}$ for two $\mathcal{KPF}$'s $\mathcal{F}$ and $\mathcal{G}$. The results are important fundaments for the question, how formulas of temporal logic in different reactive systems interact with each other in a heterogeneous modeling environment. In this context, the following definition is important:

$$\begin{tikzcd} P \arrow[r, hook, "m"] & X \end{tikzcd}$$

$$\begin{tikzcd} \mathrm{Pred}(\mathcal{F})(P) \arrow[r, hook, "\mathcal{F}m"] \arrow[d, "\eta_P"] & \mathcal{F}X \arrow[d, "\eta_X"] \\ \mathrm{Pred}(\mathcal{G})(P) \arrow[r, hook, "\mathcal{G}m"'] & \mathcal{G}X \end{tikzcd}$$

Fig. 3. Naturality Square for Predicate Inclusion

Definition 5 (Cartesian along Inclusions).
A natural transformation $\eta : \mathcal{F} \Rightarrow \mathcal{G}$ between functors $\mathcal{F}, \mathcal{G} : \mathcal{SET} \to \mathcal{SET}$ is said to be Cartesian along inclusions*, if we have in the naturality square in Fig. 3 that*

$$\eta_X^{-1}(\mathrm{Pred}(\mathcal{G})(P)) \subseteq \mathrm{Pred}(\mathcal{F})(P).[8]$$

[8] Equivalently: The square in Fig. 3 is a pullback square.

Our first result is

Proposition 1 (Compatibility of Next Time Operator). *Let $\eta : \mathcal{F} \Rightarrow \mathcal{G}$ for two $\mathcal{KPF}$'s $\mathcal{F}$ and $\mathcal{G}$ and $\mathcal{U}_\eta : \mathcal{F}$-Coalg $\to \mathcal{G}$-Coalg the emerging co-forgetful functor from Lemma 1. Let (X, c) be an $\mathcal{F}$-system and P a predicate, then the next time operator is compatible with transformations:*

$$\forall x \in X : x \models_{\mathcal{F},c} \bigcirc P \Rightarrow x \models_{\mathcal{G},\mathcal{U}_\eta c} \bigcirc P \tag{1}$$

If furthermore η is Cartesian along inclusions, implication (1) is an equivalence.

Proof. Note that (by the definition of Pred(_)) for any structure map d:

$$x \models_{\_,d} \bigcirc P \iff d(x) \in \mathrm{Pred}(\_)(P).$$

Fix a set X. The assumption of (1) is $c(x) \in \mathrm{Pred}(\mathcal{F})(P)$. From Lemma 2, we know that the square in Fig. 3 commutes. Thus by the Def. of $\mathcal{U}_\eta$

$$(\mathcal{U}_\eta c)(x) = \eta_X(c(x)) = \mathcal{G}m(\eta_P(x)) = \eta_P(x) \in \mathrm{Pred}(\mathcal{G})(P),$$

i.e. $x \models_{\mathcal{G},\mathcal{U}_\eta c} \bigcirc P$. Cartesian along inclusions yields $c(x) \in \mathrm{Pred}(\mathcal{F})(P)$, if $\eta_X(c(x)) = (\mathcal{U}_\eta c)(x) \in \mathrm{Pred}(\mathcal{G})(P)$. □

The following example shows that we cannot expect translation to always reflect truth, i.e. the precondition in the second part of Prop. 1 is necessary. For $\mathcal{F} = (\_)^A$ and $\mathcal{G} = 1 + \_$. We consider the natural transformation

$$\eta_X : \begin{cases} X^A \to 1 + X \\ \;f \;\mapsto \;\; * \end{cases}$$

which intuitively removes all transitions from an $\mathcal{F}$-system. Now consider the property $P = \texttt{false}$, i.e. $P = \emptyset$. Obviously for an $\mathcal{F}$-system (X, c), $x \models_{\mathcal{G},\mathcal{U}_\eta c} \bigcirc \emptyset$ for all $x \in X$, because for $d := \mathcal{U}_\eta c$ we obtain

$$\bigcirc_{\mathcal{G},d} P = d^{-1}(\mathrm{Pred}(\mathcal{G})(\emptyset)) = d^{-1}(1 + \emptyset) = d^{-1}(1) = X$$

since $d(x) = *$ for all $x \in X$. This is also intuitively clear, because a property holds in all successor states, if there are no such states. However, $x \not\models_{\mathcal{F},c} \bigcirc \emptyset$ for all $x \in X$, because all x possess transitions in an $\mathcal{F}$-system. And η is not Cartesian along inclusions: $\eta_X^{-1}(1 + P) = X^A \not\subseteq \emptyset = \emptyset^A$.

The following result delineates conditions for preservation and reflection of truth w.r.t. all temporal operators. We formulate it for finite state sets X, see Remark 1, which is sufficient for practical applications in software engineering.

Proposition 2 (Truth Preservation and Reflection). *Let $\eta : \mathcal{F} \Rightarrow \mathcal{G}$ for two $\mathcal{KPF}$'s $\mathcal{F}$ and $\mathcal{G}$, which is Cartesian along inclusions, and $\mathcal{U}_\eta : \mathcal{F}$-Coalg $\to \mathcal{G}$-Coalg the emerging co-forgetful functor from Lemma 1. Let (X, c) be an $\mathcal{F}$-system with finite state set X and Ω any of the above mentioned temporal operators, then:*

$$\forall x \in X : x \models_{\mathcal{F},c} \Omega\, P \iff x \models_{\mathcal{G},\mathcal{U}_\eta c} \Omega\, P \tag{2}$$

Proof. For the two elementary operators $\exists\Box$ and $\exists\_\mathcal{U}\_$, this follows from Lemma 4 and Prop. 1 by simple induction, because X is a finite set. The result then easily propagates to all derived operators, like $\forall\Box$ and $\forall\Diamond$, and furthermore to all nested formulas. $\Box$

This result causes truth preservation and reflection along specification morphisms $\eta : \mathcal{F} \Rightarrow \mathcal{G}$ of all temporal formulas, thus enabling validity checks being independent of whether they are carried out in the category of $\mathcal{F}$-systems or in the category of $\mathcal{G}$-systems, respectively.

4.2 The Case Study revisited

Examples 1, 2, and 3 showed how to encode the backend system and the common platform and provided the translation of the former to the latter. Our goal is now to provide a corresponding translation of a BPMN process model to the common platform. We assume the state space of a BPMN diagram to be a set of possible token distributions in the diagram, cf. [21], equivalently it can be seen as the set of enabled tasks and events, cf. [8]. Hence, we can encode BPMN models as coalgebras for the functor

$$\mathcal{F}_2 = (1 + \_)^n \times (1 + \_)^E \times ((1 + \_) \times R)^T.$$

Here E is the set of catch events in the process model, e.g. "Additional Information Received" in Fig. 1, T is the set of non-automatic tasks, e.g. "Enhance Ticket Data" or (the business rule task) "Classify Ticket" and R is set of roles in the process model assigned to user tasks, e.g. "IT-Staff" for the above user task. For the sake of simplicity, we assume that each automatic task calls exactly one method in the backend system.

Then the transition structure is given by assigning to a state $x \in X$ a triple $c(x) = (h_1, h_2, h_3)$ of maps. The function application $h_1(j)$ specifies, whether in state x an automatic task is ready to request method m_j with successor state x' (if $h_1(j) = x' \neq *$) or whether this is not the case ($h_1(j) = *$), i.e., the successor state is independent of input and output of the called method, cf. remark on abstraction in BPMN models in Sect. 2. Similarly, $h_2(e)$ specifies, whether in state x, the process is ready to receive event $e \in E$ or not, and $h_3(t) = (h_{3,1}(t), h_{3,2}(t))$ indicates whether a user task $t \in T$ with role assignment $h_{3,2}(t) \in R$ produces successor state $h_{3,1}(t)$ (or is disabled, if $h_{3,1}(t) = *$). The next goal is to find a natural transformation η' in

$$\begin{tikzcd} \mathcal{F}_1 \arrow[r, Rightarrow, "\eta"] & \mathcal{G} & \mathcal{F}_2 \arrow[l, Rightarrow, "\eta'"'] \end{tikzcd}$$

In order to translate input-independent part h_1 to LTS where the alphabet distinguishes between all elements $i \in I_j$, the map $h_1(j) : I_j \to X$ is a constant map. Furthermore, we have to bear in mind that process instances evolve due to events and user activities. Hence we allow these ingredients in the common platform and choose for the functor $\mathcal{G} : \mathcal{SET} \to \mathcal{SET}$ in Example 1 and 3:

$$A := I + \coprod_{j=1}^{n} I_j \quad \text{with} \quad I := E + T.$$

Then the transformation

$$\eta'_Y : \begin{cases} (1+Y)^n \times (1+Y)^E \times ((1+Y) \times R)^T \to (1+Y)^A \\ \qquad (h_1, h_2, h_3) \qquad \mapsto \lambda z. \begin{cases} h_1(j) & , \text{ if } z = i \in I_j \\ h_2(e) & , \text{ if } z = e \in E \\ h_{3,1}(t) & , \text{ if } z = t \in T \end{cases} \end{cases}$$

additionally forgets role assignments and thus faithfully translates the BPMN model into a DLTS.

4.3 Handshaking

Let (X, c) be a backend system ($\mathcal{F}_1$-system) and (Y, d) be a BPMN model ($\mathcal{F}_2$-system), then the two translated systems $\mathcal{U}_\eta c$ and $\mathcal{U}_{\eta'} d$ can be synchronized over a set of common communication channels, in our case the touch points are the methods $\{m_1, \ldots, m_n\}$ together with their inputs, i.e. it is the set

$$H := \coprod_{j=1}^{n} I_j.$$

The resulting system $s := \mathcal{U}_\eta c \,||_H\, \mathcal{U}_{\eta'} d$, is the parallel composition whose components communicate synchronously (handshake) over channels H and all actions outside H are independent and therefore can be executed autonomously in an interleaved manner [1].

Furthermore formula φ can now be formulated based on the specification $\mathcal{G}$ of the common transition structures. Model checking means then to ask whether in a certain state $(x, y) \in X \times Y$ of the composed $\mathcal{G}$-system formula φ holds:

$$(x, y) \overset{?}{\models}_{\mathcal{G},s} \varphi$$

Assume now, an additional formula ψ has been imposed on (X, c) or (Y, d), then the question is, how φ and ψ interact: Are they contradictory? Does one of them logically imply the other? Is a syntactical combination, e.g. $\psi \Rightarrow \varphi$, valid? etc ... Because φ only lives on our LTS-platform ($\mathcal{G}$-systems) and ψ lives either in $\mathcal{F}_1$- or in $\mathcal{F}_2$-systems, it is desirable, to know whether $x \models \psi$ is true independent of whether we check in $\mathcal{G}$ or in $\mathcal{F}_{1/2}$.

It follows now immediately from the definition of η (Example 1) and η' (see above in the present section), that both natural transformations are Cartesian along inclusion: Whenever $P \subseteq X$, then for any map m (or h) in the domain of η_X (or η'_X), such that its η_X-image maps into P, whenever the values shall be in X, we immediately see, that this is also the case for m, i.e. $\eta_X^{-1}((1+P)^A) \subseteq \mathcal{F}_{1/2}P$.

Hence Proposition 2 guarantees this independence and all model checks can be carried out on the common platform.

5 Related Work

Practical Approaches: The general idea of transforming different behavioural formalisms to a single semantic domain to reason about crosscutting concerns is nothing new [7]. [14] developed consistency checking for sequence diagrams and statecharts based on CSP, while Petri nets were used for the same scenario in [24]. Nevertheless, all approaches utilize fixed types of transition systems and no common framework, which can capture all possible types of transition structures. [6] tackles the problem of dealing with relationships between heterogeneous behavioural models. They coordinate the different models using a dedicated coordination language, which we formalize using morphisms between behavioural specifications.

Theoretical Approaches: Reasoning about heterogenously typed transition structures leads to the general theory of *(co-)institutions*, in which the same functor $\mathcal{U}_\eta$ as above is used to build the covariant model functor. In addition to our approach, a concrete (contravariant) functor for formula translation (of modal logics) is used: [5] defines (many-sorted) specification morphisms $\mathcal{F} \to \mathcal{G}$ as natural transformations from $\mathcal{G}$ to $\mathcal{F}$ and shows that formulae are preserved and reflected, if the negation operator is omitted (positive logic), see also [2]. [16] proves three different types of logics for coalgebras to be institutions. Another approach are parametrized endofunctors as comprehensive behavioural specifications, where the overall structure can be studied in terms of cofibrations [13]. [23] investigates co-institutions purely dual to classical institutions [19]. Finally, a good overview over the connection between coalgebars and modal logic is [12].

6 Future work

More general Translations: We plan to use more general natural transformations to relate specifications $\mathcal{F}$ and $\mathcal{G}$. An established possibility [18] is to investigate translation properties, when using

$$\eta : \mathcal{H}\mathcal{F} \Rightarrow \mathcal{G}\mathcal{H}$$

for some reasonably chosen functor $\mathcal{H} : \mathcal{SET} \to \mathcal{SET}$. This yields the translation of an $\mathcal{F}$-system (X, c) to the $\mathcal{G}$-system $(\mathcal{H}X, \eta_X \circ \mathcal{H}c)$ and thus enables transformation of the state space, too. The question is, whether we can expect similar results as above for this kind of translations.

Refined Formulas: Formulas of modal logic can be refined w.r.t. to input symbols, e.g. you want to express that property P holds after a transition with structure map c *only if there was a certain input/event*. A formalisation of this is described in [10], Chapter 6.5., which enables defining formulas as in Hennessy-Milner-Logic [15].

Evaluation and Implementation: Finally, we want to investigate, how application(framework)s for checking behavioural consistency in heterogeneous modeling scenarios can be implemented based on the insights of the present paper. It is a goal to formally underpin already existing work [11].

References

1. Baier, C., Katoen, J.P.: Principles of Model Checking. The MIT Press (2008)
2. Balan, A., Kurz, A., Velebil, J.: An institutional approach to positive coalgebraic logic. Journal of Logic and Computation **27**(6), 1799–1824 (2017)
3. Barr, M., Wells, C.: Category Theory for Computing Sciences. Prentice Hall (1990)
4. Bradfield, J.C., Stirling, C.: Modal Logics and mu-Calculi: An Introduction. In: Bergstra, J.A., Ponse, A., Smolka, S.A. (eds.) Handbook of Process Algebra, pp. 293–330. North-Holland / Elsevier (2001). https://doi.org/10.1016/b978-044482830-9/50022-9
5. Cırstea, C.: An institution of modal logics for coalgebras. The Journal of Logic and Algebraic Programming **67**(1-2), 87–113 (2006)
6. Deantoni, J.: Modeling the behavioral semantics of heterogeneous languages and their coordination. In: 2016 Architecture-Centric Virtual Integration (ACVI). pp. 12–18. IEEE (2016)
7. Engels, G., Küster, J.M., Heckel, R., Groenewegen, L.: A methodology for specifying and analyzing consistency of object-oriented behavioral models. In: Tjoa, A.M., Gruhn, V. (eds.) Proceedings of the 8th European Software Engineering Conference held jointly with 9th ACM SIGSOFT International Symposium on Foundations of Software Engineering 2001, Vienna, Austria, September 10-14, 2001. pp. 186–195. ACM (2001). https://doi.org/10.1145/503209.503235
8. Fiadeiro, J.L.: Categories for Software Engineering. Springer (2005)
9. Jacobs, B.: Objects and Classes, Co-Algebraically. In: Freitag, B., Jones, C.B., Lengauer, C., Schek, H. (eds.) Object Orientation with Parallelism and Persistence (the book grow out of a Dagstuhl Seminar in April 1995). pp. 83–103. Kluwer Academic Publishers (1995)
10. Jacobs, B.: Introduction to Coalgebra: Towards Mathematics of States and Observation, Cambridge Tracts in Theoretical Computer Science, vol. 59. Cambridge University Press (2016). https://doi.org/10.1017/CBO9781316823187
11. Kräuter, T.: Towards behavioral consistency in heterogeneous modeling scenarios. In: ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion, MODELS 2021 Companion, Fukuoka, Japan, October 10-15, 2021. pp. 666–671. IEEE (2021). https://doi.org/10.1109/MODELS-C53483.2021.00107
12. Kurz, A.: Coalgebras and modal logic. Course Notes for ESSLLI **2001** (2001)
13. Kurz, A., Pattinson, D.: Coalgebras and modal logic for parameterised endofunctors. Centrum voor Wiskunde en Informatica (2000)
14. Küster, J.M.: Towards Inconsistency Handling of Object-Oriented Behavioral Models. Electron. Notes Theor. Comput. Sci. **109**, 57–69 (2004). https://doi.org/10.1016/j.entcs.2004.02.056
15. Milner, R.: Communication and concurrency, vol. 84. Prentice hall Englewood Cliffs (1989)
16. Pattinson, D.: Translating logics for coalgebras. In: International Workshop on Algebraic Development Techniques. pp. 393–408. Springer (2002)
17. Reichel, H.: An Approach to Object Semantics based on Terminal Co-Algebras. Math. Struct. Comput. Sci. **5**(2), 129–152 (1995). https://doi.org/10.1017/S0960129500000694
18. Rutten, J.J.M.M.: Universal coalgebra: a theory of systems. Theor. Comput. Sci. **249**(1), 3–80 (2000). https://doi.org/10.1016/S0304-3975(00)00056-6

19. Sannella, D., Tarlecki, A.: Foundations of algebraic specification and formal software development. Springer Science & Business Media (2012)
20. Tarski, A.: A Lattice-Theoretical Fixpoint Theorem and its Applications. Pacific Journal of Mathematics **5**, 285–309 (1955)
21. Van Gorp, P., Dijkman, R.: A visual token-based formalization of BPMN 2.0 based on in-place transformations. Information and Software Technology **55**(2), 365–394 (2013). https://doi.org/https://doi.org/10.1016/j.infsof.2012.08.014, special Section: Component-Based Software Engineering (CBSE), 2011
22. Wechler, W.: Universal Algebra for Computer Scientists. Springer-Verlag Berlin, Heidelberg (1992)
23. Wolter, U.: (Co)Institutions for Coalgebras. Reports in Informatics 415 (2016)
24. Yao, S., Shatz, S.M.: Consistency checking of UML dynamic models based on Petri net techniques. In: 2006 15th International Conference on Computing. pp. 289–297. IEEE (2006)