A Hybrid Dynamic Logic for Event/Data-Based Systems

Rolf Hennicker[1], Alexandre Madeira[2,3](✉), and Alexander Knapp[4]

[1] Ludwig-Maximilians-Universität München, Munich, Germany
hennicke@pst.ifi.lmu.de
[2] CIDMA, University of Aveiro, Aveiro, Portugal
madeira@ua.pt
[3] QuantaLab, University of Minho, Braga, Portugal
[4] Universität Augsburg, Augsburg, Germany
knapp@informatik.uni-augsburg.de

Abstract. We propose $\mathcal{E}^{\downarrow}$-logic as a formal foundation for the specification and development of event-based systems with local data states. The logic is intended to cover a broad range of abstraction levels from abstract requirements specifications up to constructive specifications. Our logic uses diamond and box modalities over structured actions adopted from dynamic logic. Atomic actions are pairs $e /\!\!/ \psi$ where e is an event and ψ a state transition predicate capturing the allowed reactions to the event. To write concrete specifications of recursive process structures we integrate (control) state variables and binders of hybrid logic. The semantic interpretation relies on event/data transition systems; specification refinement is defined by model class inclusion. For the presentation of constructive specifications we propose operational event/data specifications allowing for familiar, diagrammatic representations by state transition graphs. We show that $\mathcal{E}^{\downarrow}$-logic is powerful enough to characterise the semantics of an operational specification by a single $\mathcal{E}^{\downarrow}$-sentence. Thus the whole development process can rely on $\mathcal{E}^{\downarrow}$-logic and its semantics as a common basis. This includes also a variety of implementation constructors to support, among others, event refinement and parallel composition.

1 Introduction

Event-based systems are an important kind of software systems which are open to the environment to react to certain events. A crucial characteristics of such systems is that not any event can (or should) be expected at any time. Hence the control flow of the system is significant and should be modelled by appropriate means. On the other hand components administrate data which may change upon the occurrence of an event. Thus also the specification of admissible data changes caused by events plays a major role.

A. Madeira—Supported by ERDF through COMPETE 2020 and by National Funds through FCT with POCI-01-0145-FEDER-016692 and UID/MAT/04106/2019, in a contract foreseen in nos. 4–6 of art. 23 of the DL 57/2016, changed by DL 57/2017.

There is quite a lot of literature on modelling and specification of event-based systems. Many approaches, often underpinned by graphical notations, provide formalisms aiming at being constructive enough to suggest particular designs or implementations, like e.g., Event-B [1,7], symbolic transition systems [17], and UML behavioural and protocol state machines [12,16]. On the other hand, there are logical formalisms to express desired properties of event-based systems. Among them are temporal logics integrating state and event-based styles [4], and various kinds of modal logics involving data, like first-order dynamic logic [10] or the modal μ-calculus with data and time [9]. The gap between logics and constructive specification is usually filled by checking whether *the* model of a constructive specification satisfies certain logical formulae.

In this paper we are interested in investigating a logic which is capable to express properties of event/data-based systems on various abstraction levels in a common formalism. For this purpose we follow ideas of [15], but there data states, effects of events on them and constructive operational specifications (see below) were not considered. The advantage of an expressive logic is that we can split the transition from system requirements to system implementation into a series of gradual refinement steps which are more easy to understand, to verify, and to adjust when certain aspects of the system are to be changed or when a product line of similar products has to be developed.

To that end we propose $\mathcal{E}^{\downarrow}$-logic, a dynamic logic enriched with features of hybrid logic. The dynamic part uses diamond and box modalities over structured actions. Atomic actions are of the form $e/\!\!/\psi$ with e an event and ψ a state transition predicate specifying the admissible effects of e on the data. Using sequential composition, union, and iteration we obtain complex actions that, in connection with the modalities, can be used to specify required and forbidden behaviour. In particular, if E is a finite set of events, though data is infinite we are able to capture all reachable states of the system and to express safety and liveness properties. But $\mathcal{E}^{\downarrow}$-logic is also powerful enough to specify concrete, recursive process structures by integrating state variables and binders from hybrid logic [6] with the subtle difference that our state variables are used to denote control states only. We show that the dynamic part of the logic is bisimulation invariant while the hybrid part, due to the ability to bind names to states, is not.

An axiomatic specification $Sp = (\Sigma, Ax)$ in $\mathcal{E}^{\downarrow}$ is given by an event/data signature $\Sigma = (E, A)$, with a set E of events and a set A of attributes to model local data states, and a set of $\mathcal{E}^{\downarrow}$-sentences Ax, called axioms, expressing requirements. For the semantic interpretation we use event/data transition systems (edts). Their states are reachable configurations $\gamma = (c, \omega)$ where c is a control state, recording the current state of execution, and ω is a local data state, i.e., a valuation of the attributes. Transitions between configurations are labelled by events. The semantics of a specification Sp is "loose" in the sense that it consists of *all* edts satisfying the axioms of the specification. Such structures are called models of Sp. Loose semantics allows us to define a simple refinement notion: Sp_1 refines to Sp_2 if the model class of Sp_2 is included in the model class of Sp_1. We may also say that Sp_2 is an implementation of Sp_1.

Our refinement process starts typically with axiomatic specifications whose axioms involve only the dynamic part of the logic. Hybrid features will successively be added in refinements when specifying more concrete behaviours, like loops. Aiming at a concrete design, the use of an axiomatic specification style may, however, become cumbersome since we have to state explicitly also all negative cases, what the system should not do. For a convenient presentation of constructive specifications we propose operational event/data specifications, which are a kind of symbolic transition systems equipped again with a model class semantics in terms of edts. We will show that $\mathcal{E}^{\downarrow}$-logic, by use of the hybrid binder, is powerful enough to characterise the semantics of an operational specification. Therefore we have not really left $\mathcal{E}^{\downarrow}$-logic when refining axiomatic by operational specifications. Moreover, since several constructive notations in the literature, including (essential parts of) Event-B, symbolic transition systems, and UML protocol state machines, can be expressed as operational specifications, $\mathcal{E}^{\downarrow}$-logic provides a logical umbrella under which event/data-based systems can be developed.

In order to consider more complex refinements we take up an idea of Sannella and Tarlecki [18,19] who have proposed the notion of constructor implementation. This is a generic notion applicable to specification formalisms based on signatures and semantic structures for signatures. As both are available in the context of $\mathcal{E}^{\downarrow}$-logic, we complement our approach by introducing a couple of constructors, among them event refinement and parallel composition. For the latter we provide a useful refinement criterion relying on a relationship between syntactic and semantic parallel composition. The logic and the use of the implementation constructors will be illustrated by a running example.

Hereafter, in Sect. 2, we introduce syntax and semantics of $\mathcal{E}^{\downarrow}$-logic. In Sect. 3, we consider axiomatic as well as operational specifications and demonstrate the expressiveness of $\mathcal{E}^{\downarrow}$-logic. Refinement of both types of specifications using several implementation constructors is considered in Sect. 4. Section 5 provides some concluding remarks. Proofs of theorems and facts can be found in [11].

2 A Hybrid Dynamic Logic for Event/Data Systems

We propose the logic $\mathcal{E}^{\downarrow}$ to specify and reason about event/data-based systems. $\mathcal{E}^{\downarrow}$-logic is an extension of the hybrid dynamic logic considered in [15] by taking into account changing data. Therefore, we first summarise our underlying notions used for the treatment of data. We then introduce the syntax and semantics of $\mathcal{E}^{\downarrow}$ with its hybrid and dynamic logic features applied to events and data.

2.1 Data States

We assume given a universe $\mathcal{D}$ of *data values*. A *data signature* is given by a set A of *attributes*. An A-*data state* ω is a function $\omega : A \to \mathcal{D}$. We denote by $\Omega(A)$ the set of all A-data states. For any data signature A, we assume given a set $\Phi(A)$ of *state predicates* to be interpreted over single A-data states, and a set

$\Psi(A)$ of *transition predicates* to be interpreted over pairs of pre- and post-A-data states. The concrete syntax of state and transition predicates is of no particular importance for the following. For an attribute $a \in A$, a state predicate may be $a > 0$; and a transition predicate e.g. $a' = a + 1$, where a refers to the value of attribute a in the pre-data state and a' to its value in the post-data state. Still, both types of predicates are assumed to contain true and to be closed under negation (written $\neg$) and disjunction (written $\vee$); as usual, we will then also use false, $\wedge$, etc. Furthermore, we assume for each $A_0 \subseteq A$ a transition predicate $\mathrm{id}_{A_0} \in \Psi(A)$ expressing that the values of attributes in A_0 are the same in pre- and post-A-data states.

We write $\omega \models^{\mathcal{D}}_{A} \varphi$ if $\varphi \in \Phi(A)$ is satisfied in data state ω; and $(\omega_1, \omega_2) \models^{\mathcal{D}}_{A} \psi$ if $\psi \in \Psi(A)$ is satisfied in the pre-data state ω_1 and post-data state ω_2. In particular, $(\omega_1, \omega_2) \models^{\mathcal{D}}_{A} \mathrm{id}_{A_0}$ if, and only if, $\omega_1(a_0) = \omega_2(a_0)$ for all $a_0 \in A_0$.

2.2 $\mathcal{E}^{\downarrow}$-Logic

Definition 1. *An* event/data signature *(*ed signature*, for short) $\Sigma = (E, A)$ consists of a finite set of* events E *and a data signature A. We write $E(\Sigma)$ for E and $A(\Sigma)$ for A. We also write $\Omega(\Sigma)$ for $\Omega(A(\Sigma))$, $\Phi(\Sigma)$ for $\Phi(A(\Sigma))$, and $\Psi(\Sigma)$ for $\Psi(A(\Sigma))$. The class of ed signatures is denoted by $Sig^{\mathcal{E}^{\downarrow}}$.*

Any ed signature Σ determines a class of semantic structures, the *event/data transition systems* which are reachable transition systems with sets of initial states and events as labels on transitions. The states are pairs $\gamma = (c, \omega)$, called *configurations*, where c is a *control state* recording the current execution state and ω is an $A(\Sigma)$-data state; we write $c(\gamma)$ for c and $\omega(\gamma)$ for ω.

Definition 2. *A Σ-*event/data transition system *(Σ-*edts*, for short) $M = (\Gamma, R, \Gamma_0)$ over an ed signature Σ consists of a set of* configurations $\Gamma \subseteq C \times \Omega(\Sigma)$ *for a set of* control states C*; a family of* transition relations $R = (R_e \subseteq \Gamma \times \Gamma)_{e \in E(\Sigma)}$*; and a non-empty set of* initial configurations $\Gamma_0 \subseteq \{c_0\} \times \Omega_0$ *for an* initial control state $c_0 \in C$ *and a set of* initial data states $\Omega_0 \subseteq \Omega(\Sigma)$ *such that Γ is* reachable *via R, i.e., for all $\gamma \in \Gamma$ there are $\gamma_0 \in \Gamma_0$, $n \geq 0$, $e_1, \dots, e_n \in E(\Sigma)$, and $(\gamma_i, \gamma_{i+1}) \in R_{e_{i+1}}$ for all $0 \leq i < n$ with $\gamma_n = \gamma$. We write $\Gamma(M)$ for Γ, $C(M)$ for C, $R(M)$ for R, $c_0(M)$ for c_0, $\Omega_0(M)$ for Ω_0, and $\Gamma_0(M)$ for Γ_0. The class of Σ-edts is denoted by $Edts^{\mathcal{E}^{\downarrow}}(\Sigma)$.*

Atomic actions are given by expressions of the form $e/\!\!/\psi$ with e an event and ψ a state transition predicate. The intuition is that the occurrence of the event e causes a state transition in accordance with ψ, i.e., the pre- and post-data states satisfy ψ, and ψ specifies the possible effects of e. Following the ideas of dynamic logic we also use complex, structured actions formed over atomic actions by union, sequential composition and iteration. All kinds of actions over an ed signature Σ are called *Σ-event/data actions* (*Σ-ed actions*, for short). The set $\Lambda(\Sigma)$ of Σ-ed actions is defined by the grammar

$$\lambda ::= e/\!\!/\psi \mid \lambda_1 + \lambda_2 \mid \lambda_1 ; \lambda_2 \mid \lambda^*$$

where $e \in E(\Sigma)$ and $\psi \in \Psi(\Sigma)$. We use the following shorthand notations for actions: For a subset $F = \{e_1, \dots, e_k\} \subseteq E(\Sigma)$, we use the notation F to denote the complex action $e_1 /\!\!/ \mathrm{true} + \dots + e_k /\!\!/ \mathrm{true}$ and $-F$ to denote the action $E(\Sigma) \setminus F$. For the action $E(\Sigma)$ we will write $\boldsymbol{E}$. For $e \in E(\Sigma)$, we use the notation e to denote the action $e /\!\!/ \mathrm{true}$ and $-e$ to denote the action $\boldsymbol{E} \setminus \{e\}$. Hence, if $E(\Sigma) = \{e_1, \dots, e_n\}$ and $e_i \in E(\Sigma)$, the action $-e_i$ stands for $e_1 /\!\!/ \mathrm{true} + \dots + e_{i-1} /\!\!/ \mathrm{true} + e_{i+1} /\!\!/ \mathrm{true} + \dots + e_n /\!\!/ \mathrm{true}$.

The actions $\Lambda(\Sigma)$ are *interpreted* over a Σ-edts M as the family of relations $(R(M)_\lambda \subseteq \Gamma(M) \times \Gamma(M))_{\lambda \in \Lambda(\Sigma)}$ defined by

- $R(M)_{e /\!\!/ \psi} = \{(\gamma, \gamma') \in R(M)_e \mid (\omega(\gamma), \omega(\gamma')) \models^{\mathcal{D}}_{A(\Sigma)} \psi\}$,
- $R(M)_{\lambda_1 + \lambda_2} = R(M)_{\lambda_1} \cup R(M)_{\lambda_2}$, i.e., union of relations,
- $R(M)_{\lambda_1 ; \lambda_2} = R(M)_{\lambda_1} ; R(M)_{\lambda_2}$, i.e., sequential composition of relations,
- $R(M)_{\lambda^*} = (R(M)_\lambda)^*$, i.e., reflexive-transitive closure of relations.

To define the event/data formulae of $\mathcal{E}^\downarrow$ we assume given a countably infinite set X of control state variables which are used in formulae to denote the control part of a configuration. They can be bound by the binder operator $\downarrow x$ and accessed by the jump operator $@x$ of hybrid logic. The dynamic part of our logic is due to the modalities which can be formed over any ed action over a given ed signature. $\mathcal{E}^\downarrow$ thus retains from hybrid logic the use of binders, but omits free nominals. Thus sentences of the logic become restricted to express properties of configurations reachable from the initial ones.

Definition 3. *The set* $\mathrm{Frm}^{\mathcal{E}^\downarrow}(\Sigma)$ *of* Σ-ed formulae *over an ed signature* Σ *is given by*

$$\varrho ::= \varphi \mid x \mid \downarrow x \,.\, \varrho \mid @x \,.\, \varrho \mid \langle \lambda \rangle \varrho \mid \mathrm{true} \mid \neg \varrho \mid \varrho_1 \vee \varrho_2$$

where $\varphi \in \Phi(\Sigma)$*,* $x \in X$*, and* $\lambda \in \Lambda(\Sigma)$*. We write* $[\lambda]\varrho$ *for* $\neg\langle\lambda\rangle\neg\varrho$ *and we use the usual boolean connectives as well as the constant* false *to denote* $\neg$true.[1] *The set* $\mathrm{Sen}^{\mathcal{E}^\downarrow}(\Sigma)$ *of* Σ-ed sentences *consists of all* Σ*-ed formulae without free variables, where the free variables are defined as usual with* $\downarrow x$ *being the unique operator binding variables.*

Given an ed signature Σ and a Σ-edts M, the satisfaction of a Σ-ed formula ϱ is inductively defined w.r.t. valuations $v : X \to C(M)$, mapping variables to control states, and configurations $\gamma \in \Gamma(M)$:

- $M, v, \gamma \models^{\mathcal{E}^\downarrow}_\Sigma \varphi$ iff $\omega(\gamma) \models^{\mathcal{D}}_{A(\Sigma)} \varphi$;
- $M, v, \gamma \models^{\mathcal{E}^\downarrow}_\Sigma x$ iff $c(\gamma) = v(x)$;
- $M, v, \gamma \models^{\mathcal{E}^\downarrow}_\Sigma \downarrow x \,.\, \varrho$ iff $M, v\{x \mapsto c(\gamma)\}, \gamma \models^{\mathcal{E}^\downarrow}_\Sigma \varrho$;
- $M, v, \gamma \models^{\mathcal{E}^\downarrow}_\Sigma @x \,.\, \varrho$ iff $M, v, \gamma' \models^{\mathcal{E}^\downarrow}_\Sigma \varrho$ for all $\gamma' \in \Gamma(M)$ with $c(\gamma') = v(x)$;
- $M, v, \gamma \models^{\mathcal{E}^\downarrow}_\Sigma \langle\lambda\rangle\varrho$ iff $M, v, \gamma' \models^{\mathcal{E}^\downarrow}_\Sigma \varrho$ for some $\gamma' \in \Gamma(M)$ with $(\gamma, \gamma') \in R(M)_\lambda$;

[1] We use true and false for predicates and formulae; their meaning will always be clear from the context. For boolean values we will use instead the notations *tt* and *ff*.

- $M, v, \gamma \models^{\mathcal{E}^{\downarrow}}_{\Sigma}$ true always holds;
- $M, v, \gamma \models^{\mathcal{E}^{\downarrow}}_{\Sigma} \neg\varrho$ iff $M, v, \gamma \not\models^{\mathcal{E}^{\downarrow}}_{\Sigma} \varrho$;
- $M, v, \gamma \models^{\mathcal{E}^{\downarrow}}_{\Sigma} \varrho_1 \vee \varrho_2$ iff $M, v, \gamma \models^{\mathcal{E}^{\downarrow}}_{\Sigma} \varrho_1$ or $M, v, \gamma \models^{\mathcal{E}^{\downarrow}}_{\Sigma} \varrho_2$.

If ϱ is a sentence then the valuation is irrelevant. M *satisfies* a sentence $\varrho \in \mathrm{Sen}^{\mathcal{E}^{\downarrow}}(\Sigma)$, denoted by $M \models^{\mathcal{E}^{\downarrow}}_{\Sigma} \varrho$, if $M, \gamma_0 \models^{\mathcal{E}^{\downarrow}}_{\Sigma} \varrho$ for all $\gamma_0 \in \Gamma_0(M)$.

By borrowing the modalities from dynamic logic [9,10], $\mathcal{E}^{\downarrow}$ is able to express liveness and safety requirements as illustrated in our running ATM example below. There we use the fact that we can state properties over all reachable states by sentences of the form $[\boldsymbol{E}^*]\varphi$. In particular, deadlock-freedom can be expressed by $[\boldsymbol{E}^*]\langle\boldsymbol{E}\rangle$true. The logic $\mathcal{E}^{\downarrow}$, however, is also suited to directly express process structures and, thus, the implementation of abstract requirements. The binder operator is essential for this. For example, we can specify a process which switches a boolean value, denoted by the attribute val, from *tt* to *ff* and back by the following sentence:

$$\downarrow x_0 \,.\, \mathsf{val} = tt \wedge \langle \mathsf{switch} /\!\!/ \mathsf{val}' = f\!f \rangle \langle \mathsf{switch} /\!\!/ \mathsf{val}' = tt \rangle x_0 .$$

2.3 Bisimulation and Invariance

Bisimulation is a crucial notion in both behavioural systems specification and in modal logics. On the specification side, it provides a standard way to identify systems with the same behaviour by abstracting the internal specifics of the systems; this is also reflected at the logic side, where bisimulation frequently relates states that satisfy the same formulae. We explore some properties of $\mathcal{E}^{\downarrow}$ w.r.t. bisimilarity. Let us first introduce the notion of bisimilarity in the context of $\mathcal{E}^{\downarrow}$:

Definition 4. *Let M_1, M_2 be Σ-edts. A relation $B \subseteq \Gamma(M_1) \times \Gamma(M_2)$ is a* bisimulation relation *between M_1 and M_2 if for all $(\gamma_1, \gamma_2) \in B$ the following conditions hold:*

(atom) for all $\varphi \in \Phi(\Sigma)$, $\omega(\gamma_1) \models^{\mathcal{D}}_{A(\Sigma)} \varphi$ iff $\omega(\gamma_2) \models^{\mathcal{D}}_{A(\Sigma)} \varphi$;
(zig) for all $e /\!\!/ \psi \in \Lambda(\Sigma)$ and for all $\gamma_1' \in \Gamma(M_1)$ with $(\gamma_1, \gamma_1') \in R(M_1)_{e/\!\!/\psi}$, there is a $\gamma_2' \in \Gamma(M_2)$ such that $(\gamma_2, \gamma_2') \in R(M_2)_{e/\!\!/\psi}$ and $(\gamma_1', \gamma_2') \in B$;
(zag) for all $e /\!\!/ \psi \in \Lambda(\Sigma)$ and for all $\gamma_2' \in \Gamma(M_2)$ with $(\gamma_2, \gamma_2') \in R(M_2)_{e/\!\!/\psi}$, there is a $\gamma_1' \in \Gamma(M_1)$ such that $(\gamma_1, \gamma_1') \in R(M_1)_{e/\!\!/\psi}$ and $(\gamma_1', \gamma_2') \in B$.

M_1 and M_2 are bisimilar*, in symbols $M_1 \sim M_2$, if there exists a bisimulation relation $B \subseteq \Gamma(M_1) \times \Gamma(M_2)$ between M_1 and M_2 such that*

(init) for any $\gamma_1 \in \Gamma_0(M_1)$, there is a $\gamma_2 \in \Gamma_0(M_2)$ such that $(\gamma_1, \gamma_2) \in B$ and for any $\gamma_2 \in \Gamma_0(M_2)$, there is a $\gamma_1 \in \Gamma_0(M_1)$ such that $(\gamma_1, \gamma_2) \in B$.

Now we are able to establish a Hennessy-Milner like correspondence for a fragment of $\mathcal{E}^{\downarrow}$. Let us call *hybrid-free sentences of* $\mathcal{E}^{\downarrow}$ the formulae obtained by the grammar

$$\varrho ::= \varphi \mid \langle\lambda\rangle\varrho \mid \text{true} \mid \neg\varrho \mid \varrho_1 \vee \varrho_2 .$$

Theorem 1. *Let M_1, M_2 be bisimilar Σ-edts. Then $M_1 \models^{\mathcal{E}^\downarrow}_{\Sigma} \varrho$ iff $M_2 \models^{\mathcal{E}^\downarrow}_{\Sigma} \varrho$ for all hybrid-free sentences ϱ.*

The converse of Theorem 1 does not hold, in general, and the usual image-finiteness assumption has to be imposed: A Σ-edts M is *image-finite* if, for all $\gamma \in \Gamma(M)$ and all $e \in E(\Sigma)$, the set $\{\gamma' \mid (\gamma, \gamma') \in R(M)_e\}$ is finite. Then:

Theorem 2. *Let M_1, M_2 be image-finite Σ-edts and $\gamma_1 \in \Gamma(M_1)$, $\gamma_2 \in \Gamma(M_2)$ such that $M_1, \gamma_1 \models^{\mathcal{E}^\downarrow}_{\Sigma} \varrho$ iff $M_2, \gamma_2 \models^{\mathcal{E}^\downarrow}_{\Sigma} \varrho$ for all hybrid-free sentences ϱ. Then there exists a bisimulation B between M_1 and M_2 such that $(\gamma_1, \gamma_2) \in B$.*

3 Specifications of Event/Data Systems

3.1 Axiomatic Specifications

Sentences of $\mathcal{E}^\downarrow$-logic can be used to specify properties of event/data systems and thus to write system specifications in an axiomatic way.

Definition 5. *An* axiomatic ed specification *$Sp = (\Sigma(Sp), Ax(Sp))$ in $\mathcal{E}^\downarrow$ consists of an ed signature $\Sigma(Sp) \in Sig^{\mathcal{E}^\downarrow}$ and a set of* axioms *$Ax(Sp) \subseteq \mathrm{Sen}^{\mathcal{E}^\downarrow}(\Sigma(Sp))$.*

The semantics *of Sp is given by the pair $(\Sigma(Sp), \mathrm{Mod}(Sp))$ where $\mathrm{Mod}(Sp) = \{M \in Edts^{\mathcal{E}^\downarrow}(\Sigma(Sp)) \mid M \models^{\mathcal{E}^\downarrow}_{\Sigma(Sp)} Ax(Sp)\}$. The edts in $\mathrm{Mod}(Sp)$ are called* models *of Sp and $\mathrm{Mod}(Sp)$ is the* model class *of Sp.*

As a direct consequence of Theorem 1 we have:

Corollary 1. *The model class of an axiomatic ed specification exclusively expressed by hybrid-free sentences is closed under bisimulation.*

This result does not hold for sentences with hybrid features. For instance, consider the specification $Sp = \big((\{e\}, \{a\}), \{\downarrow x\,.\,\langle e /\!\!/ a' = a\rangle x\}\big)$: An edts with a single control state c_0 and a loop transition $R_e = \{(\gamma_0, \gamma_0)\}$ for $c(\gamma_0) = c_0$ is a model of Sp. However, this is obviously not the case for its bisimilar edts with two control states c_0 and c and the relation $R'_e = \{(\gamma_0, \gamma), (\gamma, \gamma_0)\}$ with $c(\gamma_0) = c_0$, $c(\gamma) = c$ and $\omega(\gamma_0) = \omega(\gamma)$.

Example 1. As a running example we consider an ATM. We start with an abstract specification Sp_0 of fundamental requirements for its interaction behaviour based on the set of events $E_0 = \{\mathsf{insertCard}, \mathsf{enterPIN}, \mathsf{ejectCard}, \mathsf{cancel}\}$[2] and on the singleton set of attributes $A_0 = \{\mathsf{chk}\}$ where chk is boolean valued and records the correctness of an entered PIN. Hence our first ed signature is $\Sigma_0 = (E_0, A_0)$ and $Sp_0 = (\Sigma_0, Ax_0)$ where Ax_0 requires the following properties expressed by corresponding axioms (0.1–0.3):

[2] For shortening the presentation we omit further events like withdrawing money, etc.

– "Whenever a card has been inserted, a correct PIN can eventually be entered and also the transaction can eventually be cancelled."

$$[\boldsymbol{E}^*; \mathsf{insertCard}](\langle \boldsymbol{E}^*; \mathsf{enterPIN} /\!\!/ \mathsf{chk}' = tt\rangle \mathrm{true} \wedge \langle \boldsymbol{E}^*; \mathsf{cancel}\rangle \mathrm{true}) \qquad (0.1)$$

– "Whenever either a correct PIN has been entered or the transaction has been cancelled, the card can eventually be ejected."

$$[\boldsymbol{E}^*; (\mathsf{enterPIN} /\!\!/ \mathsf{chk}' = tt) + \mathsf{cancel}]\langle \boldsymbol{E}^*; \mathsf{ejectCard}\rangle \mathrm{true} \qquad (0.2)$$

– "Whenever an incorrect PIN has been entered three times in a row, the current card is not returned." This means that the card is kept by the ATM which is not modelled by an extra event. It may, however, still be possible that another card is inserted afterwards. So an ejectCard can only be forbidden as long as no next card is inserted.

$$[\boldsymbol{E}^*; (\mathsf{enterPIN} /\!\!/ \mathsf{chk}' = f\!f)^3; (-\mathsf{insertCard})^*; \mathsf{ejectCard}]\mathrm{false} \qquad (0.3)$$

where λ^n abbreviates the n-fold sequential composition $\lambda; \ldots; \lambda$.

The semantics of an axiomatic ed specification is loose allowing usually for many different realisations. A refinement step is therefore understood as a restriction of the model class of an abstract specification. Following the terminology of Sannella and Tarlecki [18,19], we call a specification refining another one an *implementation*. Formally, a specification Sp' is a *simple implementation* of a specification Sp over the same signature, in symbols $Sp \rightsquigarrow Sp'$, whenever $\mathrm{Mod}(Sp) \supseteq \mathrm{Mod}(Sp')$. Transitivity of the inclusion relation ensures gradual step-by-step development by a series of refinements.

Example 2. We provide a refinement $Sp_0 \rightsquigarrow Sp_1$ where $Sp_1 = (\Sigma_0, Ax_1)$ has the same signature as Sp_0 and Ax_1 are the sentences (1.1–1.4) below; the last two use binders to specify a loop. As is easily seen, all models of Sp_1 must satisfy the axioms of Sp_0.

– "At the beginning a card can be inserted with the effect that chk is set to $f\!f$; nothing else is possible at the beginning."

$$\begin{aligned}&\langle \mathsf{insertCard} /\!\!/ \mathsf{chk}' = f\!f\rangle \mathrm{true} \wedge \\ &[\mathsf{insertCard} /\!\!/ \neg(\mathsf{chk}' = f\!f)]\mathrm{false} \wedge [-\mathsf{insertCard}]\mathrm{false}\end{aligned} \qquad (1.1)$$

– "Whenever a card has been inserted, a PIN can be entered (directly afterwards) and also the transaction can be cancelled; but nothing else."

$$\begin{aligned}[\boldsymbol{E}^*; \mathsf{insertCard}](&\langle \mathsf{enterPIN}\rangle \mathrm{true} \wedge \langle \mathsf{cancel}\rangle \mathrm{true} \wedge \\ &[-\{\mathsf{enterPIN}, \mathsf{cancel}\}]\mathrm{false})\end{aligned} \qquad (1.2)$$

- "Whenever either a correct PIN has been entered or the transaction has been cancelled, the card can eventually be ejected and the ATM starts from the beginning."

$$\downarrow x_0 \,.\, [\boldsymbol{E}^*; (\mathsf{enterPIN} /\!\!/ \mathsf{chk}' = tt) + \mathsf{cancel}]\langle \boldsymbol{E}^*; \mathsf{ejectCard}\rangle x_0 \qquad (1.3)$$

- "Whenever an incorrect PIN has been entered three times in a row the ATM starts from the beginning." Hence the current card is kept.

$$\downarrow x_0 \,.\, [\boldsymbol{E}^*; (\mathsf{enterPIN} /\!\!/ \mathsf{chk}' = f\!f)^3]x_0 \qquad (1.4)$$

3.2 Operational Specifications

Operational event/data specifications are introduced as a means to specify in a more constructive style the properties of event/data systems. They are not appropriate for writing abstract requirements for which axiomatic specifications are recommended. Though $\mathcal{E}^{\downarrow}$-logic is able to specify concrete models, as discussed in Sect. 2, the use of operational specifications allows a graphic representation close to familiar formalisms in the literature, like UML protocol state machines, cf. [12,16]. As will be shown in Sect. 3.3, finite operational specifications can be characterised by a sentence in $\mathcal{E}^{\downarrow}$-logic. Therefore, $\mathcal{E}^{\downarrow}$-logic is still the common basis of our development approach. Transitions in an operational specification are tuples $(c, \varphi, e, \psi, c')$ with c a source control state, φ a precondition, e an event, ψ a state transition predicate specifying the possible effects of the event e, and c' a target control state. In the semantic models an event must be enabled whenever the respective source data state satisfies the precondition. Thus isolating preconditions has a semantic consequence that is not expressible by transition predicates only. The effect of the event must respect ψ; no other transitions are allowed.

Definition 6. *An* operational ed specification $O = (\Sigma, C, T, (c_0, \varphi_0))$ *is given by an ed signature* Σ*, a set of* control states C*, a* transition relation specification $T \subseteq C \times \Phi(\Sigma) \times E(\Sigma) \times \Psi(\Sigma) \times C$*, an* initial control state $c_0 \in C$*, and an* initial state predicate $\varphi_0 \in \Phi(\Sigma)$*, such that* C *is* syntactically reachable*, i.e., for every* $c \in C \setminus \{c_0\}$ *there are* $(c_0, \varphi_1, e_1, \psi_1, c_1), \ldots, (c_{n-1}, \varphi_n, e_n, \psi_n, c_n) \in T$ *with* $n > 0$ *such that* $c_n = c$*. We write* $\Sigma(O)$ *for* Σ*, etc.*

A Σ*-edts* M *is a* model *of* O *if* $C(M) = C$ *up to a bijective renaming,* $c_0(M) = c_0$*,* $\Omega_0(M) \subseteq \{\omega \mid \omega \models^{\mathcal{D}}_{A(\Sigma)} \varphi_0\}$*, and if the following conditions hold:*

- *for all* $(c, \varphi, e, \psi, c') \in T$ *and* $\omega \in \Omega(A(\Sigma))$ *with* $\omega \models^{\mathcal{D}}_{A(\Sigma)} \varphi$*, there is a* $((c, \omega), (c', \omega')) \in R(M)_e$ *with* $(\omega, \omega') \models^{\mathcal{D}}_{A(\Sigma)} \psi$*;*

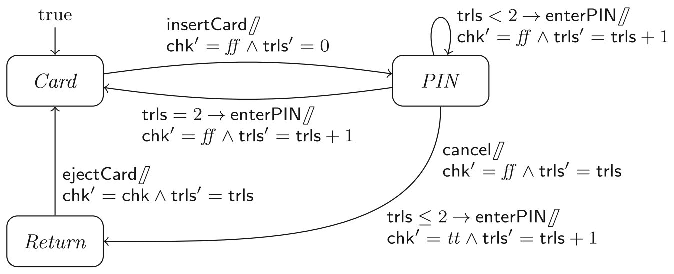


Fig. 1. Operational ed specification *ATM*

- *for all* $((c, \omega), (c', \omega')) \in R(M)_e$ *there is a* $(c, \varphi, e, \psi, c') \in T$ *with* $\omega \models^{\mathcal{D}}_{A(\Sigma)} \varphi$ *and* $(\omega, \omega') \models^{\mathcal{D}}_{A(\Sigma)} \psi$.

The class of all models of O *is denoted by* $\mathrm{Mod}(O)$. *The* semantics *of* O *is given by the pair* $(\Sigma(O), \mathrm{Mod}(O))$ *where* $\Sigma(O) = \Sigma$.

Example 3. We construct an operational ed specification, called *ATM*, for the ATM example. The signature of *ATM* extends the one of Sp_1 (and Sp_0) by an additional integer-valued attribute trls which counts the number of attempts to enter a correct PIN (with the same card). *ATM* is graphically presented in Fig. 1. The initial control state is *Card*, and the initial state predicate is true. Preconditions are written before the symbol $\longrightarrow$. If no precondition is explicitly indicated it is assumed to be true. Due to the extended signature, *ATM* is not a simple implementation of Sp_1, and we will only formally justify the implementation relationship in Example 5.

Operational specifications can be composed by a syntactic parallel composition operator which synchronises shared events. Two ed signatures Σ_1 and Σ_2 are *composable* if $A(\Sigma_1) \cap A(\Sigma_2) = \emptyset$. Their parallel composition is given by $\Sigma_1 \otimes \Sigma_2 = (E(\Sigma_1) \cup E(\Sigma_2), A(\Sigma_1) \cup A(\Sigma_2))$.

Definition 7. *Let* Σ_1 *and* Σ_2 *be composable ed signatures and let* O_1 *and* O_2 *be operational ed specifications with* $\Sigma(O_1) = \Sigma_1$ *and* $\Sigma(O_2) = \Sigma_2$. *The* parallel composition *of* O_1 *and* O_2 *is given by the operational ed specification* $O_1 \parallel O_2 = (\Sigma_1 \otimes \Sigma_2, C, T, (c_0, \varphi_0))$ *with* $c_0 = (c_0(O_1), c_0(O_2))$, $\varphi_0 = \varphi_0(O_1) \wedge \varphi_0(O_2)$, *and* C *and* T *are inductively defined by* $c_0 \in C$ *and*

- *for* $e_1 \in E(\Sigma_1) \setminus E(\Sigma_2)$, $c_1, c_1' \in C(O_1)$, *and* $c_2 \in C(O_2)$, *if* $(c_1, c_2) \in C$ *and* $(c_1, \varphi_1, e_1, \psi_1, c_1') \in T(O_1)$, *then* $(c_1', c_2) \in C$ *and* $((c_1, c_2), \varphi_1, e_1, \psi_1 \wedge \mathrm{id}_{A(\Sigma_2)}, (c_1', c_2)) \in T$;
- *for* $e_2 \in E(\Sigma_2) \setminus E(\Sigma_1)$, $c_2, c_2' \in C(O_2)$, *and* $c_1 \in C(O_1)$, *if* $(c_1, c_2) \in C$ *and* $(c_2, \varphi_2, e_2, \psi_2, c_2') \in T(O_2)$, *then* $(c_1, c_2') \in C$ *and* $((c_1, c_2), \varphi_2, e_2, \psi_2 \wedge \mathrm{id}_{A(\Sigma_1)}, (c_1, c_2')) \in T$;

– *for* $e \in E(\Sigma_1) \cap E(\Sigma_2)$, $c_1, c'_1 \in C(O_1)$, *and* $c_2, c'_2 \in C(O_2)$, *if* $(c_1, c_2) \in C$, $(c_1, \varphi_1, e, \psi_1, c'_1) \in T(O_1)$, *and* $(c_2, \varphi_2, e, \psi_2, c'_2) \in T(O_2)$, *then* $(c'_1, c'_2) \in C$ *and* $((c_1, c_2), \varphi_1 \wedge \varphi_2, e, \psi_1 \wedge \psi_2, (c'_1, c'_2)) \in T$.[3]

3.3 Expressiveness of $\mathcal{E}^{\downarrow}$-Logic

We show that the semantics of an operational ed specification O with finitely many control states can be characterised by a single $\mathcal{E}^{\downarrow}$-sentence ϱ_O, i.e., an edts M is a model of O iff $M \models^{\mathcal{E}^{\downarrow}}_{\Sigma(O)} \varrho_O$. Using Algorithm 1, such a characterising sentence is

$$\varrho_O = {\downarrow} c_0 \,.\, \varphi_0 \wedge \mathrm{sen}(c_0, \mathit{Im}_O(c_0), C(O), \{c_0\}) \ ,$$

where $c_0 = c_0(O)$ and $\varphi_0 = \varphi_0(O)$. Algorithm 1 closely follows the procedure in [15] for characterising a finite structure by a sentence of $\mathcal{D}^{\downarrow}$-logic. A call sen(c, I, V, B) performs a recursive breadth-first traversal through O starting from c, where I holds the unprocessed quadruples (φ, e, ψ, c') of transitions 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, provided its precondition holds, in the resulting formula, binding any newly reached state. Then it requires that no other transitions with source state c exist using calls to fin. Having visited all states in V, it finally requires all states in $C(O)$ to be pairwise different.

Algorithm 1. Constructing a sentence from an operational ed specification

```
Require: $O \equiv$ finite operational ed specification
         $\mathit{Im}_O(c) = \{(\varphi, e, \psi, c') \mid (c, \varphi, e, \psi, c') \in T(O)\}$ for $c \in C(O)$
         $\mathit{Im}_O(c, e) = \{(\varphi, \psi, c') \mid (c, \varphi, e, \psi, c') \in T(O)\}$ for $c \in C(O)$, $e \in E(\Sigma(O))$
function sen($c, I, V, B$)    ▷ $c$: state, $I$: image to visit, $V$: states to visit, $B$: bound states
    if $I \neq \emptyset$ then
        $(\varphi, e, \psi, c') \leftarrow$ choose $I$
        if $c' \in B$ then
            return $@c \,.\, \varphi \to \langle e /\!\!/ \psi \rangle (c' \wedge \mathrm{sen}(c, I \setminus \{(\varphi, e, \psi, c')\}, V, B))$
        else
            return $@c \,.\, \varphi \to \langle e /\!\!/ \psi \rangle ({\downarrow} c' \,.\, \mathrm{sen}(c, I \setminus \{(\varphi, e, \psi, c')\}, V, B \cup \{c'\}))$
    $V \leftarrow V \setminus \{c\}$
    if $V \neq \emptyset$ then
        $c' \leftarrow$ choose $B \cap V$
        return $\mathrm{fin}(c) \wedge \mathrm{sen}(c', \mathit{Im}_O(c'), V, B)$
    return $\mathrm{fin}(c) \wedge \bigwedge_{c_1 \in C(O), c_2 \in C(O) \setminus \{c_1\}} \neg @c_1 \,.\, c_2$
function fin($c$)
    return $@c \,.\, \bigwedge_{e \in E(\Sigma(O))} \bigwedge_{P \subseteq \mathit{Im}_O(c,e)}$
                $[e /\!\!/ (\bigwedge_{(\varphi,\psi,c') \in P} (\varphi \wedge \psi)) \wedge$
                    $\neg(\bigvee_{(\varphi,\psi,c') \in \mathit{Im}_O(c,e) \setminus P} (\varphi \wedge \psi))](\bigvee_{(\varphi,\psi,c') \in P} c')$
```

[3] Note that joint moves with e cannot become inconsistent due to composability of ed signatures.

It is $\text{fin}(c)$ where this algorithm mainly deviates from [15]: To ensure that no other transitions from c exist than those specified in O, $\text{fin}(c)$ produces the requirement that at state c, for every event e and for every subset P of the transitions outgoing from c, whenever an e-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-transition must have one of the states as its target that are target states of P. The rather complicated formulation is due to possibly overlapping preconditions where for a single event e the preconditions of two different transitions may be satisfied simultaneously. For a state c, where all outgoing transitions for the same event have disjoint preconditions, the $\mathcal{E}^{\downarrow}$-formula returned by $\text{fin}(c)$ is equivalent to

$$@c\,.\ \bigwedge_{e\in E(\Sigma(O))} \bigwedge_{(\varphi,\psi,c')\in Im_O(c,e)} [e /\!\!/ \varphi \wedge \psi]c' \wedge \\ [e /\!\!/ \neg\big(\bigvee_{(\varphi,\psi,c')\in Im_O(c,e)} (\varphi \wedge \psi)\big)]\text{false}.$$

Example 4. We show the first few steps of representing the operational ed specification *ATM* of Fig. 1 as an $\mathcal{E}^{\downarrow}$-sentence ϱ_{ATM}. This top-level sentence is

$$\downarrow Card\,.\,\text{true} \wedge \text{sen}(Card, \{(\text{true}, \mathsf{insertCard}, \mathsf{chk}' = \mathit{ff} \wedge \mathsf{trls}' = 0, PIN)\}, \\ \{Card, PIN, Return\}, \{Card\}).$$

The first call of $\text{sen}(Card, \ldots)$ explores the single outgoing transition from *Card* to *PIN*, adds *PIN* to the bound states, and hence expands to

$$@Card\,.\,\text{true} \rightarrow \langle \mathsf{insertCard} /\!\!/ \mathsf{chk}' = \mathit{ff} \wedge \mathsf{trls}' = 0\rangle \downarrow PIN\,. \\ \text{sen}(Card, \emptyset, \{Card, PIN, Return\}, \{Card, PIN\}).$$

Now all outgoing transitions from *Card* have been explored and the next call of $\text{sen}(Card, \emptyset, \ldots)$ removes *Card* from the set of states to be visited, resulting in

$$\text{fin}(Card) \wedge \text{sen}(PIN, \{(\mathsf{trls} < 2, \mathsf{enterPIN}, \ldots), (\mathsf{trls} = 2, \mathsf{enterPIN}, \ldots), \\ (\mathsf{trls} \leq 2, \mathsf{enterPIN}, \ldots), (\text{true}, \mathsf{cancel}, \ldots)\}, \\ \{PIN, Return\}, \{Card, PIN\}).$$

As there is only a single outgoing transition from *Card*, the special case of disjoint preconditions applies for the finalisation call, and $\text{fin}(Card)$ results in

$$@Card\,.\,[\mathsf{insertCard} /\!\!/ \mathsf{chk}' = \mathit{ff} \wedge \mathsf{trls}' = 0]PIN \wedge \\ [\mathsf{insertCard} /\!\!/ \mathsf{chk}' = \mathit{tt} \vee \mathsf{trls}' \neq 0]\text{false} \wedge \\ [\mathsf{enterPIN} /\!\!/ \text{true}]\text{false} \wedge [\mathsf{cancel} /\!\!/ \text{true}]\text{false} \wedge [\mathsf{ejectCard} /\!\!/ \text{true}]\text{false}.$$

4 Constructor Implementations

The implementation notion defined in Sect. 3.1 is too simple for many practical applications. It requires the same signature for specification and implementation and does not support the process of constructing an implementation. Therefore, Sannella and Tarlecki [18,19] have proposed the notion of constructor implementation which is a generic notion applicable to specification formalisms which are based on signatures and semantic structures for signatures. We will reuse the ideas in the context of $\mathcal{E}^{\downarrow}$-logic.

The notion of *constructor* is the basis: for signatures $\Sigma_1, \dots, \Sigma_n, \Sigma \in \mathit{Sig}^{\mathcal{E}^{\downarrow}}$, a *constructor* κ from $(\Sigma_1, \dots, \Sigma_n)$ to Σ is a (total) function $\kappa : \mathit{Edts}^{\mathcal{E}^{\downarrow}}(\Sigma_1) \times \dots \times \mathit{Edts}^{\mathcal{E}^{\downarrow}}(\Sigma_n) \to \mathit{Edts}^{\mathcal{E}^{\downarrow}}(\Sigma)$. Given a constructor κ from $(\Sigma_1, \dots, \Sigma_n)$ to Σ and a set of constructors κ_i from $(\Sigma_i^1, \dots, \Sigma_i^{k_i})$ to Σ_i, $1 \le i \le n$, the constructor $(\kappa_1, \dots, \kappa_n); \kappa$ from $(\Sigma_1^1, \dots, \Sigma_1^{k_1}, \dots, \Sigma_n^1, \dots, \Sigma_n^{k_n})$ to Σ is obtained by the usual composition of functions. The following definitions apply to both axiomatic and operational ed specifications since the semantics of both is given in terms of ed signatures and model classes of edts. In particular, the implementation notion allows to implement axiomatic specifications by operational specifications.

Definition 8. *Given specifications $\mathit{Sp}, \mathit{Sp}_1, \dots, \mathit{Sp}_n$ and a constructor κ from $(\Sigma(\mathit{Sp}_1), \dots, \Sigma(\mathit{Sp}_n))$ to $\Sigma(\mathit{Sp})$, the tuple $\langle \mathit{Sp}_1, \dots, \mathit{Sp}_n \rangle$ is a* constructor implementation via *κ of Sp, in symbols $\mathit{Sp} \rightsquigarrow_\kappa \langle \mathit{Sp}_1, \dots, \mathit{Sp}_n \rangle$, if for all $M_i \in \mathrm{Mod}(\mathit{Sp}_i)$ we have $\kappa(M_1, \dots, M_n) \in \mathrm{Mod}(\mathit{Sp})$. The implementation involves a* decomposition *if $n > 1$.*

The notion of simple implementation in Sect. 3.1 is captured by choosing the identity. We now introduce a set of more advanced constructors in the context of ed signatures and edts. Let us first consider two central notions for constructors: signature morphisms and reducts. For data signatures A, A' a *data signature morphism* $\sigma : A \to A'$ is a function from A to A'. The *σ-reduct* of an A'-data state $\omega' : A' \to \mathcal{D}$ is given by the A-data state $\omega'|\sigma : A \to \mathcal{D}$ defined by $(\omega'|\sigma)(a) = \omega'(\sigma(a))$ for every $a \in A$. If $A \subseteq A'$, the injection of A into A' is a particular data signature morphism and we denote the reduct of an A'-data state ω' to A by $\omega' \upharpoonright A$. If $A = A_1 \cup A_2$ is the disjoint union of A_1 and A_2 and ω_i are A_i-data states for $i \in \{1, 2\}$ then $\omega_1 + \omega_2$ denotes the unique A-data state ω with $\omega \upharpoonright A_i = \omega_i$ for $i \in \{1, 2\}$. The σ-reduct $\gamma|\sigma$ of a configuration $\gamma = (c, \omega')$ is given by $(c, \omega'|\sigma)$, and is lifted to a set of configurations Γ' by $\Gamma'|\sigma = \{\gamma'|\sigma \mid \gamma' \in \Gamma'\}$.

Definition 9. *An* ed signature morphism *$\sigma = (\sigma_E, \sigma_A) : \Sigma \to \Sigma'$ is given by a function $\sigma_E : E(\Sigma) \to E(\Sigma')$ and a data signature morphism $\sigma_A : A(\Sigma) \to A(\Sigma')$. We abbreviate both σ_E and σ_A by σ.*

Definition 10. *Let $\sigma : \Sigma \to \Sigma'$ be an ed signature morphism and M' a Σ'-edts. The* σ-reduct *of M' is the Σ-edts $M'|\sigma = (\Gamma, R, \Gamma_0)$ such that $\Gamma_0 = \Gamma_0(M')|\sigma$, and Γ and $R = (R_e)_{e \in E(\Sigma)}$ are inductively defined by $\Gamma_0 \subseteq \Gamma$ and for all $e \in E(\Sigma)$, $\gamma', \gamma'' \in \Gamma(M')$: if $\gamma'|\sigma \in \Gamma$ and $(\gamma', \gamma'') \in R(M')_{\sigma(e)}$, then $\gamma''|\sigma \in \Gamma$ and $(\gamma'|\sigma, \gamma''|\sigma) \in R_e$.*

Definition 11. *Let $\sigma : \Sigma \to \Sigma'$ be an ed signature morphism. The* reduct constructor *κ_σ from Σ' to Σ maps any $M' \in \mathit{Edts}^{\mathcal{E}^{\downarrow}}(\Sigma')$ to its reduct $\kappa_\sigma(M') = M'|\sigma$. Whenever σ_A and σ_E are bijective functions, κ_σ is a* relabelling constructor*. If σ_E and σ_A are injective, κ_σ is a* restriction constructor.

Example 5. The operational specification ATM is a constructor implementation of Sp_1 via the restriction constructor κ_ι determined by the inclusion signature morphism $\iota : \Sigma(\mathit{Sp}_1) \to \Sigma(\mathit{ATM})$, i.e., $\mathit{Sp}_1 \rightsquigarrow_{\kappa_\iota} \mathit{ATM}$.

A further refinement technique for reactive systems (see, e.g., [8]), is the implementation of simple events by complex events, like their sequential composition. To formalise this as a constructor we use *composite events* $\Theta(E)$ over a given set of events E, given by the grammar $\theta ::= e \mid \theta + \theta \mid \theta ; \theta \mid \theta^*$ with $e \in E$. They are *interpreted* over an (E, A)-edts M by $R(M)_{\theta_1+\theta_2} = R(M)_{\theta_1} \cup R(M)_{\theta_2}$, $R(M)_{\theta_1;\theta_2} = R(M)_{\theta_1} ; R(M)_{\theta_2}$, and $R(M)_{\theta^*} = (R(M)_\theta)^*$. Then we can introduce the intended constructor by means of reducts over signature morphisms mapping atomic to composite events:

Definition 12. *Let Σ, Σ' be ed signatures, D' a finite subset of $\Theta(E(\Sigma'))$, $\Delta' = (D', A(\Sigma'))$, and $\alpha : \Sigma \to \Delta'$ an ed signature morphism. The* event refinement *constructor κ_α from Δ' to Σ maps any $M' \in \mathit{Edts}^{\mathcal{E}^\downarrow}(\Delta')$ to its reduct $M'|\alpha \in \mathit{Edts}^{\mathcal{E}^\downarrow}(\Sigma)$.*

Finally, we consider a semantic, synchronous parallel composition constructor that allows for decomposition of implementations into components which synchronise on shared events. Given two composable signatures Σ_1 and Σ_2, the *parallel composition* $\gamma_1 \otimes \gamma_2$ of two configurations $\gamma_1 = (c_1, \omega_1)$, $\gamma_2 = (c_2, \omega_2)$ with $\omega_1 \in \Omega(A(\Sigma_1))$, $\omega_2 \in \Omega(A(\Sigma_2))$ is given by $((c_1, c_2), \omega_1 + \omega_2)$, and lifted to two sets of configurations Γ_1 and Γ_2 by $\Gamma_1 \otimes \Gamma_2 = \{\gamma_1 \otimes \gamma_2 \mid \gamma_1 \in \Gamma_1,\ \gamma_2 \in \Gamma_2\}$.

Definition 13. *Let Σ_1, Σ_2 be composable ed signatures. The* parallel composition constructor *$\kappa_\otimes$ from (Σ_1, Σ_2) to $\Sigma_1 \otimes \Sigma_2$ maps any $M_1 \in \mathit{Edts}^{\mathcal{E}^\downarrow}(\Sigma_1)$, $M_2 \in \mathit{Edts}^{\mathcal{E}^\downarrow}(\Sigma_2)$ to $M_1 \otimes M_2 = (\Gamma, R, \Gamma_0) \in \mathit{Edts}^{\mathcal{E}^\downarrow}(\Sigma_1 \otimes \Sigma_2)$, where $\Gamma_0 = \Gamma_0(M_1) \otimes \Gamma_0(M_2)$, and Γ and $R = (R_e)_{E(\Sigma_1) \cup E(\Sigma_2)}$ are inductively defined by $\Gamma_0 \subseteq \Gamma$ and*

- *for all $e_1 \in E(\Sigma_1) \setminus E(\Sigma_2)$, $\gamma_1, \gamma_1' \in \Gamma(M_1)$, and $\gamma_2 \in \Gamma(M_2)$, if $\gamma_1 \otimes \gamma_2 \in \Gamma$ and $(\gamma_1, \gamma_1') \in R(M_1)_{e_1}$, then $\gamma_1' \otimes \gamma_2 \in \Gamma$ and $(\gamma_1 \otimes \gamma_2, \gamma_1' \otimes \gamma_2) \in R_{e_1}$;*
- *for all $e_2 \in E(\Sigma_2) \setminus E(\Sigma_1)$, $\gamma_2, \gamma_2' \in \Gamma(M_2)$, and $\gamma_1 \in \Gamma(M_1)$, if $\gamma_1 \otimes \gamma_2 \in \Gamma$ and $(\gamma_2, \gamma_2') \in R(M_2)_{e_2}$, then $\gamma_1 \otimes \gamma_2' \in \Gamma$ and $(\gamma_1 \otimes \gamma_2, \gamma_1 \otimes \gamma_2') \in R_{e_2}$;*
- *for all $e \in E(\Sigma_1) \cap E(\Sigma_2)$, $\gamma_1, \gamma_1' \in \Gamma(M_1)$, and $\gamma_2, \gamma_2' \in \Gamma(M_2)$, if $\gamma_1 \otimes \gamma_2 \in \Gamma$, $(\gamma_1, \gamma_1') \in R(M_1)_{e_1}$, and $(\gamma_2, \gamma_2') \in R(M_2)_{e_2}$, then $\gamma_1' \otimes \gamma_2' \in \Gamma$ and $(\gamma_1 \otimes \gamma_2, \gamma_1' \otimes \gamma_2') \in R_e$.*

An obvious question is how the semantic parallel composition constructor is related to the syntactic parallel composition of operational ed specifications.

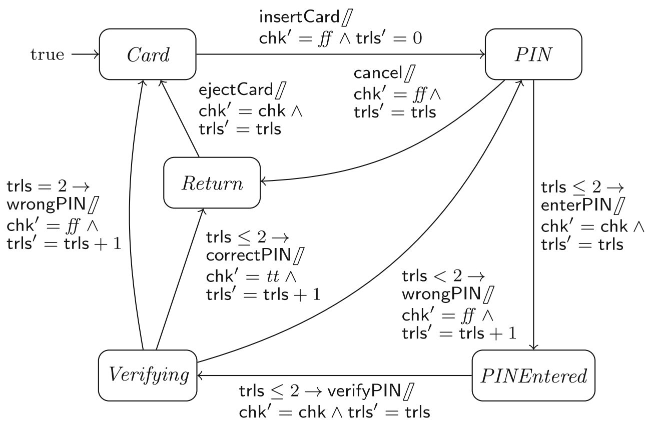
Proposition 1. *Let O_1, O_2 be operational ed specifications with composable signatures. Then* $\mathrm{Mod}(O_1) \otimes \mathrm{Mod}(O_2) \subseteq \mathrm{Mod}(O_1 \parallel O_2)$*, where* $\mathrm{Mod}(O_1) \otimes \mathrm{Mod}(O_2)$ *denotes* $\kappa_\otimes(\mathrm{Mod}(O_1), \mathrm{Mod}(O_2))$.

The converse $\mathrm{Mod}(O_1 \parallel O_2) \subseteq \mathrm{Mod}(O_1) \otimes \mathrm{Mod}(O_2)$ does not hold: Consider the ed signature $\Sigma = (E, A)$ with $E = \{e\}$, $A = \emptyset$, and the operational ed specifications $O_i = (\Sigma, C_i, T_i, (c_{i,0}, \varphi_{i,0}))$ for $i \in \{1, 2\}$ with $C_1 = \{c_{1,0}\}$, $T_1 = \{(c_{1,0}, \mathrm{true}, e, \mathrm{false}, c_{1,0})\}$, $\varphi_{1,0} = \mathrm{true}$; and $C_2 = \{c_{2,0}\}$, $T_2 = \emptyset$, $\varphi_{2,0} = \mathrm{true}$. Then $\mathrm{Mod}(O_1) = \emptyset$, but $\mathrm{Mod}(O_1 \parallel O_2) = \{M\}$ with M showing just the initial configuration.

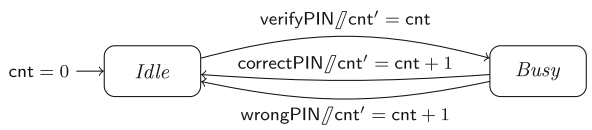
The next theorem shows the usefulness of the syntactic parallel composition operator for proving implementation correctness when a (semantic) parallel composition constructor is involved. The theorem is a direct consequence of Proposition 1 and Definition 8.

Theorem 3. *Let Sp be an (axiomatic or operational) ed specification, O_1, O_2 operational ed specifications with composable signatures, and κ an implementation constructor from $\Sigma(O_1) \otimes \Sigma(O_2)$ to $\Sigma(Sp)$: If $Sp \rightsquigarrow_{\kappa} O_1 \parallel O_2$, then $Sp \rightsquigarrow_{\kappa_\otimes ; \kappa} \langle O_1, O_2 \rangle$.*

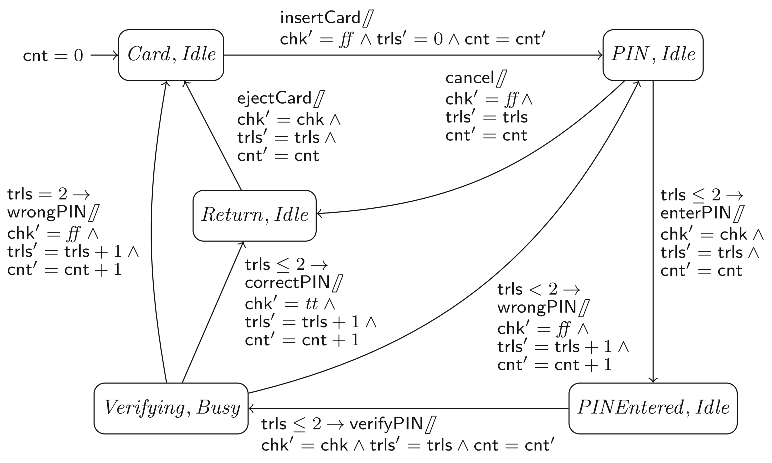
Example 6. We finish the refinement chain for the ATM specifications by applying a decomposition into two parallel components. The operational specification ATM of Example 3 (and Example 5) describes the interface behaviour of an ATM interacting with a user. For a concrete realisation, however, an ATM will also interact internally with other components, like, e.g., a clearing company which supports the ATM for verifying PINs. Our last refinement step hence realises the ATM specification by two parallel components, represented by the operational specification ATM' in Fig. 2a and the operational specification CC of a clearing company in Fig. 2b. Both communicate (via shared events) when an ATM sends a verification request, modelled by the event $\mathsf{verifyPIN}$, to the clearing company. The clearing company may answer with $\mathsf{correctPIN}$ or $\mathsf{wrongPIN}$ and then the ATM continues following its specification. For the implementation construction we use the parallel composition constructor $\kappa_\otimes$ from $(\Sigma(ATM'), \Sigma(CC))$ to $\Sigma(ATM') \otimes \Sigma(CC)$. The signature of CC consists of the events shown on the transitions in Fig. 2b. Moreover, there is one integer-valued attribute cnt counting the number of verification tasks performed. The signature of ATM' extends $\Sigma(ATM)$ by the events $\mathsf{verifyPIN}$, $\mathsf{correctPIN}$ and $\mathsf{wrongPIN}$. To fit the signature and the behaviour of the parallel composition of ATM' and CC to the specification ATM we must therefore compose $\kappa_\otimes$ with an event refinement constructor κ_α such that $\alpha(\mathsf{enterPIN}) = (\mathsf{enterPIN}; \mathsf{verifyPIN}; (\mathsf{correctPIN} + \mathsf{wrongPIN}))$; for the other events α is the identity and for the attributes the inclusion. The idea is therefore that the refinement looks like $ATM \rightsquigarrow_{\kappa_\otimes ; \kappa_\alpha} \langle ATM', CC \rangle$. To prove this refinement relation we rely on the syntactic parallel composition $ATM' \parallel CC$ shown in Fig. 2c, and on Theorem 3. It is easy to see that $ATM \rightsquigarrow_{\kappa_\alpha} ATM' \parallel CC$. In fact, all transitions for event $\mathsf{enterPIN}$ in Fig. 1 are split into several transitions in Fig. 2c according to the event refinement defined by α. For instance, the loop transition from PIN to PIN with precondition $\mathsf{trls} < 2$ in Fig. 1 is split into



(a) Operational ed specification ATM'



(b) Operational specification CC of a clearing company



(c) Syntactic parallel composition $ATM' \parallel CC$

Fig. 2. Operational ed specifications ATM', CC and their parallel composition

the cycle from $(PIN, Idle)$ via $(PINEntered, Idle)$ and $(Verifying, Busy)$ back to $(PIN, Idle)$ in Fig. 2c. Thus, we have $ATM \rightsquigarrow_{\kappa_\alpha} ATM' \parallel CC$ and can apply Theorem 3 such that we get $ATM \rightsquigarrow_{\kappa_\otimes;\,\kappa_\alpha} \langle ATM', CC\rangle$.

5 Conclusions

We have presented a novel logic, called $\mathcal{E}^{\downarrow}$-logic, for the rigorous formal development of event-based systems incorporating changing data states. To the best of our knowledge, no other logic supports the full development process for this kind of systems ranging from abstract requirements specifications, expressible by the dynamic logic features, to the concrete specification of implementations, expressible by the hybrid part of the logic.

The temporal logic of actions (TLA [13]) supports also stepwise refinement where state transition predicates are considered as actions. In contrast to TLA we model also the events which cause data state transitions. For writing concrete specifications we have proposed an operational specification format capturing (at least parts of) similar formalisms, like Event-B [1], symbolic transition systems [17], and UML protocol state machines [16]. A significant difference to Event-B machines is that we distinguish between control and data states, the former being encoded as data in Event-B. On the other hand, Event-B supports parameters of events which could be integrated in our logic as well. An institution-based semantics of Event-B has been proposed in [7] which coincides with our semantics of operational specifications for the special case of deterministic state transition predicates. Similarly, our semantics of operational specifications coincides with the unfolding of symbolic transition systems in [17] if we instantiate our generic data domain with algebraic specifications of data types (and consider again only deterministic state transition predicates). The syntax of UML protocol state machines is about the same as the one of operational event/data specifications. As a consequence, all of the aforementioned concrete specification formalisms (and several others) would be appropriate candidates for integration into a development process based on $\mathcal{E}^{\downarrow}$-logic.

There remain several interesting tasks for future research. First, our logic is not yet equipped with a proof system for deriving consequences of specifications. This would also support the proof of refinement steps which is currently achieved by purely semantic reasoning. A proof system for $\mathcal{E}^{\downarrow}$-logic must cover dynamic and hybrid logic parts at the same time, like the proof system in [15], which, however, does not consider data states, and the recent calculus of [5], which extends differential dynamic logic but does not deal with events and reactions to events. Both proof systems could be appropriate candidates for incorporating the features of $\mathcal{E}^{\downarrow}$-logic. Another issue concerns the separation of events into input and output as in I/O-automata [14]. Then also communication compatibility (see [2] for interface automata without data and [3] for interface theories with data) would become relevant when applying a parallel composition constructor.

References

1. Abrial, J.R.: Modeling in Event-B: System and Software Engineering. Cambridge University Press, Cambridge (2013)
2. de Alfaro, L., Henzinger, T.A.: Interface automata. In: Tjoa, A.M., Gruhn, V. (eds.) Proceedings 8th European Software Engineering Conference & 9th ACM SIGSOFT International Symposium Foundations of Software Engineering, pp. 109–120. ACM (2001)
3. Bauer, S.S., Hennicker, R., Wirsing, M.: Interface theories for concurrency and data. Theoret. Comput. Sci. **412**(28), 3101–3121 (2011)
4. ter Beek, M.H., Fantechi, A., Gnesi, S., Mazzanti, F.: An action/state-based model-checking approach for the analysis of communication protocols for service-oriented applications. In: Leue, S., Merino, P. (eds.) FMICS 2007. LNCS, vol. 4916, pp. 133–148. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-79707-4_11
5. Bohrer, B., Platzer, A.: A hybrid, dynamic logic for hybrid-dynamic information flow. In: Dawar, A., Grädel, E. (eds.) Proceedings of 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, pp. 115–124. ACM (2018)
6. Braüner, T.: Hybrid Logic and its Proof-Theory. Applied Logic Series. Springer, Heidelberg (2010). https://doi.org/10.1007/978-94-007-0002-4
7. Farrell, M., Monahan, R., Power, J.F.: An institution for Event-B. In: James, P., Roggenbach, M. (eds.) WADT 2016. LNCS, vol. 10644, pp. 104–119. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-72044-9_8
8. Gorrieri, R., Rensink, A.: Action refinement. In: Bergstra, J.A., Ponse, A., Smolka, S.A. (eds.) Handbook of Process Algebra, pp. 1047–1147. Elsevier, Amsterdam (2000)
9. Groote, J.F., Mousavi, M.R.: Modeling and Analysis of Communicating Systems. MIT Press, Cambridge (2014)
10. Harel, D., Kozen, D., Tiuryn, J.: Dynamic Logic. MIT Press, Cambridge (2000)
11. Hennicker, R., Madeira, A., Knapp, A.: A hybrid dynamic logic for event/data-based systems (2019). https://arxiv.org/abs/1902.03074
12. Knapp, A., Mossakowski, T., Roggenbach, M., Glauer, M.: An institution for simple UML state machines. In: Egyed, A., Schaefer, I. (eds.) FASE 2015. LNCS, vol. 9033, pp. 3–18. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46675-9_1
13. Lamport, L.: Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers. Addison-Wesley, Boston (2003)
14. Lynch, N.A.: Input/output automata: basic, timed, hybrid, probabilistic, dynamic, ... In: Amadio, R.M., Lugiez, D. (eds.) CONCUR 2003. LNCS, vol. 2761, pp. 191–192. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45187-7_12
15. Madeira, A., Barbosa, L.S., Hennicker, R., Martins, M.A.: A logic for the stepwise development of reactive systems. Theoret. Comput. Sci. **744**, 78–96 (2018)
16. Object Management Group: Unified Modeling Language 2.5. Standard formal/2015-03-01, OMG (2015)
17. Poizat, P., Royer, J.C.: A formal architectural description language based on symbolic transition systems and modal logic. J. Univ. Comp. Sci. **12**(12), 1741–1782 (2006)

18. Sannella, D., Tarlecki, A.: Toward formal development of programs from algebraic specifications: implementations revisited. Acta Inf. **25**(3), 233–281 (1988)
19. Sannella, D., Tarlecki, A.: Foundations of Algebraic Specification and Formal Software Development. EATCS Monographs in Theoretical Computer Science. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-17336-3