Testing Semantics: Connecting Processes and Process Logics

Dusko Pavlovic,[1,4] Michael Mislove[2,4] and James B. Worrell[3]

[1] Kestrel Institute, Palo Alto, CA
[2] Tulane University, New Orleans, LA
[3] Oxford University, Oxford, UK

Abstract. We propose a methodology based on testing as a framework to capture the interactions of a machine represented in a denotational model and the data it manipulates. Using a duality that models machines on the one hand, and the data they manipulate on the other, testing is used to capture the interactions of each with the objects on the other side: just as the data that are input into a machine can be viewed as tests that the machine can be subjected to, the machine can be viewed as a test that can be used to distinguish data. While this approach is based on duality theories that now are common in semantics, it accomplishes much more than simply moving from one side of the duality to the other; it faithfully represents the interactions that embody what is happening as the computation proceeds.
Our basic philosophy is that tests can be used as a basis for modeling interactions, as well as processes and the data on which they operate. In more abstract terms, tests can be viewed as formulas of process logics, and testing semantics connects processes and process logics, and assigns computational meanings to both.

1 Introduction: The problem of testing

Testing a family Ξ of systems by a family Θ of tests, or process logic formulas, is a map

$$\Xi \times \Theta \xrightarrow{\mathbb{T}} \Omega$$

where Ω is the type of observations, or truth values. The simplest case is $\Omega = \{0, 1\}$, where 1 represents "accept", or "succeed", or "truth", and 0 is "reject", or "fail", or "diverge", or "false". A richer semantics can be achieved if one replaces the truth values $\{0, 1\}$ by the interval $[0, 1]$, and interprets the result of a test as the probability a process passes it. But the problem with either approach is that once the test is performed, we have only the result. Making tests more dynamic requires taking a slightly different view.

The goal of testing is to find bugs, which distinguish an implemented, real system $R \in \Xi$ from an ideal reference system $S \in \Xi$, or to demonstrate that they

[4] The support of the NSF and the US Office of Naval Research is gratefully acknowledged

are indistinguishable. A bug can be construed as a test $b \in \varTheta$, which leads to an observation $R \models b$, different from the observation $S \models b$. On the other hand, if $(R \models t) = (S \models t)$ for all tests $t \in \varTheta$, then the systems are computationally indistinguishable, modulo testing equivalence

$$R \sim S \iff \forall t \in \varTheta.\ (R \models t) = (S \models t)$$

The basic methods of studying computation in terms of tests on automata go back to the 1950s and E.P. Moore's seminal paper [1]. Moore introduced distinguishing sequences of tests, as well as testing equivalence, and several other fundamental ideas, which later led to a broad range of methods of *conformance testing*, which is the discipline of proving that an implementation R conforms to a standard S. Other problems resolved through testing include determining the current or the final state of a given automaton, or characterizing an unknown automaton.[5] One of Moore's most interesting contributions was the method of extracting minimal automata, i.e. the canonical representatives of computational behaviors, from equivalence classes of states modulo testing equivalence.

The starting point of the present work is a small modification of Moore's idea: we represent equivalent states, which form a state of a minimal automaton, not as equivalence classes of states, but as the maps from tests to observations that they induce: two states are equivalent if and only if they induce the same map. Either way, the computational behaviors arise as the elements in the image L of the semantic map, in the form

$$\begin{tikzcd}
\varXi \arrow[rr, "\models"] \arrow[dr, two heads] & & \varOmega^{\varTheta} \\
& L \arrow[ur, hook] &
\end{tikzcd}$$

The choice of representatives, of course, does not matter for abstract theory, but it turns out to make a lot of difference when it comes to analyzing state-based systems which arise in the design of reactive and embedded systems, involving stochastic, continuous, temporal or hybrid dynamics. The study of labelled Markov processes [4] provides a striking example. On the other hand, a generic categorical framework where states are represented as truth assignments of logical formulas has been used in [5–7]. In this paper, we will confine our presentation to the possibilistic setting, leaving the probabilistic setting for further work. For this setting the categorical trace semantics of finite state automata [8] and context-free languages [9] are clear examples, and are close conceptual predecessors of testing semantics. What appears to be new is our ability to bring Turing machines into the same setting.

5 Excellent surveys of testing methodologies (albeit a bit outdated in applications) are [2, 3].

2 Logical connections

A *logical connection* is a contravariant adjunction $M^{op} \dashv P : \mathcal{S}^{op} \longrightarrow \mathcal{T}$ between a category of "spaces" and a category of "types" or "theories". In one direction, a space X is mapped to the type PX of "predicates" over it; in the other direction, a type A is mapped to the space MA of its models. Among the many dualities that are examples of logical connections, we mention just a few:

Self-duality of sets : $\wp^{op} \dashv \wp : \mathsf{Set}^{op} \longrightarrow \mathsf{Set}$, which can be viewed as duality of discrete spaces and complete atomic Boolean algebras (the category of which is equivalent to Set^{op}). In more detail, the functor associates to a set A the family $\wp(A)$ of all subsets of A, and to a mapping $f \colon A \longrightarrow B$ between sets, the mapping $f^{-1} \colon \wp(B) \longrightarrow \wp(A)$. The power set of a set is a complete, atomic Boolean algebra, and the mapping f^{-1} preserves all unions, intersections and complements. Thus our duality identifies each set with the complete atomic Boolean algebra it generates, and to each algebra, its set of atoms.

Here are some other notable connections, many of them dualities:

Stone duality: More generally, if we let $\mathcal{T}$ be Boolean algebras (viewed as propositional theories), then $\mathcal{S}$ becomes Stone spaces (whose points are the ultrafilters, i.e. models of Boolean propositional theories),

Topological spaces and complete Heyting algebras: Generalizing to intuitionistic logic, we can let $pt \dashv \mathcal{O} : \mathsf{Esp}^{op} \longrightarrow \mathsf{Frm}$ [10], At this level, we get a logical connection; to obtain a duality, a restriction to sober spaces and spatial frames, respectively, is needed, but that is not required for our results,

Various spectral correspondences: $C \dashv S : \mathsf{Esp}^{op} \longrightarrow \mathsf{Rng}$, connecting topological spaces and rings (and leading to significant extensions of the notion of a logical theory)

Denotational semantics: and connections of domains and spaces with program logics [11]

The Schizophrenic object The power set of a set can equally be represented as the family of functions from the set to the two-point set, $2 = \{0, 1\}$, where one identifies a subset with its characteristic function. Dually, 2 is a Boolean algebra, and the set of atoms of a Boolean algebra B is in one-to-one correspondence with the Boolean algebra maps from B to 2. Thus, 2 is a primary example of a *schizophrenic object*, one which lives in both categories and that gives rise to a duality using the morphisms of the category. In general, when $\mathcal{S}$ and $\mathcal{T}$ have enough limits and colimits, and in particular a final object 1, then a connection between them can be viewed as homming into a "schizophrenic object" Ω, that lives in both categories, as the type $P1$ and space $M1$. Indeed, it is easy to see that these two objects have the same underlying set $Obs = |P1| = |M1|$.[6] For

[6] We write $|C| = \mathcal{C}(1, C)$ for any object C of a category $\mathcal{C}$.

every space X we also have the canonical maps

$$\frac{\coprod_{|X|} 1 \longrightarrow X}{PX \longrightarrow P(\coprod_{|X|} 1) \xrightarrow{\sim} \prod_{|X|} P1}$$

where the isomorphism arises from the fact that $P : \mathcal{S}^{op} \longrightarrow \mathcal{T}$ is a right adjoint. Similarly, for every type A there is a canonical map $MA \longrightarrow \prod_{|A|} M1$. These maps are usually monic, which means that Ω is a cogenerator[7] both in $\mathcal{S}$ and in $\mathcal{T}$. Abusing notation, we define the functors $\Omega^X = \prod_{|X|} P1$ and $\Omega^A = \prod_{|A|} M1$, and arrive at monic natural transformations

$$PX \rightarrowtail \Omega^X \qquad \text{and} \qquad MA \rightarrowtail \Omega^A$$

3 Process logics as test algebras

Process logics are modal logics for describing the behavior of computational processes. Process formulas can be viewed as tests: a process *satisfies* a formula if and only if it passes the test that the formula represents.

The first and probably best known process logic is Hennessy-Milner logic [12], which will be presented in section 6.2. In fact, computational traces can be viewed as degenerate process formulas, with no logical operations, only modalities. On the other hand, dynamic logics can be viewed as a natural extension of process logics, where modalities are generated over arbitrary programs, and not just atomic actions.

In this work, process modalities are generated over a given alphabet Σ, representing atomic actions. Sometimes we distinguish the input alphabet Σ and the output alphabet Γ; or Σ represents the external actions (terminal symbols), and Γ the internal ones.

Besides modalities, process formulas are generated by various logical signatures, i.e. sets of logical connectors represented by the theory monad $T : \mathcal{T} \longrightarrow \mathcal{T}$. If a type $A \in \mathcal{T}$ is thought of as a set of propositional letters, then the type TA is the free propositional theory, containing all formulas generated by A in the given signature. E.g., if the only logical connector is conjunction, then TA is the free semilattice over A; but it has proven useful to also consider free commutative groups, rings, and even C^*-algebras of a certain type, as "logical" theories, generating tests for certain process behaviors. In all cases, the considered algebraic theories have a distinguished constant, denoting "truth", represented by a natural transformation $1 \xrightarrow{\top} T$.

Assumption: Ω is T-algebra. It is assumed that the schizophrenic object Ω comes equipped with a canonical algebraic structure $T\Omega \longrightarrow \Omega$, which lifts to all $TPX \longrightarrow PX$ along the inclusion $PX \rightarrowtail \Omega^X$.

[7] In fact, the duality of $\mathcal{S}$ and $\mathcal{T}$ is usually built by restricting them to the parts injectively cogenerated by the object Ω, embodying their connection.

3.1 Test theories

Test theories are obtained by extending T-algebras ("propositional theories") by the modal operators generated by Σ. For example, if T is the power set functor, then we generate the so-called modal Boolean algebras by lifting actions of Σ on a transition system to modal operators for the power set of its state space. In general, a test theory is a (weak) algebra for either of the functors

$$F_0X = TX + \Sigma \times X \qquad \text{or} \qquad F_1X = T(\Sigma \times X)$$

In both cases the universal test theory is obtained as the initial weak algebra

$$\Theta_i = \mu X.\ F_iX$$

Tests are thus generated by the grammars

$$t_0 ::= \top \mid f(t_0, \ldots, t_0) \mid a.t_0 \qquad \text{and} \qquad t_1 ::= \top \mid f(a.t_1, \ldots, a.t_1)$$

where f a logical connector from the signature of T. By pre-composing with the monad T, we see that the weak F_0-algebra Θ_0 is a weak algebra for the functor T, while Θ_1 is just the free T-algebra for the monad T generated by Σ. In fact, Θ_1 is the initial *action algebra:*

Definition 1. *An* action algebra *for a monad $T : \mathcal{T} \longrightarrow \mathcal{T}$ and alphabet Σ is an algebra $TA \xrightarrow{\alpha} A$ for the monad T, together with a map $\Sigma \times A \xrightarrow{\cdot} A$, called* prefixing. *An action algebra homomorphism is a T-algebra homomorphism which also preserves prefixing.*

Proposition 1. *The free action algebra for the monad T and the alphabet Σ generated by B is the initial weak algebra $\Theta_B = \mu X.\ T(B + \Sigma \times X)$.*

4 Automata and processes as coalgebras

Nondeterminism and more recently probabilistic choice are staples of computation. The constructors for choice operators are represented by a monad $S : \mathcal{S} \longrightarrow \mathcal{S}$.

Definition 2. *A (state)* machine *with inputs from Σ, outputs from Γ and final states predicated over Υ is represented by*

- *a coalgebra $X \longrightarrow GX$ where $GX = \Upsilon \times (S(\Gamma \times X))^{\Sigma}$*
- *an initial state $x \in X$.*

A process *is a machine where any state may be final, i.e. $\Upsilon = 1$. A process thus boils down to a coalgebra $\partial : X \longrightarrow (S(\Gamma \times X))^{\Sigma}$ and the initial state $x \in X$. A machine where $\Upsilon \neq 1$ is often called an* automaton. *When the coalgebra $X \longrightarrow GX$ is clear from the context, we speak of the automaton or process $x \in X$.*

A coalgebra structure of a machine consists of a pair $X \xrightarrow{\langle \Phi, \partial \rangle} \Upsilon \times (S(\Gamma \times X))^{\Sigma}$, where $\Phi : X \longrightarrow \Upsilon$ is the characteristic function of the final states, and $\partial : X \longrightarrow (S(\Gamma \times X))^{\Sigma}$ assigns to each state a choice of an output and a next state.[8] Final states are usually evaluated in the type of truth values $\Upsilon = L$. For the possibilistic automata, $\Upsilon = 2$, and $\Phi : X \longrightarrow 2$ is just the characteristic function of the set of final states. In general, Υ may be different from L, e.g. an arbitrary semiring [13].

The computational differences between *reactive* (or reading) machines, where $\Gamma = 1$, and *generating* (or writing) machines are discussed in [14]. Coalgebras $X \longrightarrow (S(\Gamma \times X))^{\Sigma}$ thus represent processes that both read and write, which is perhaps clearer in the transposed form $\Sigma \times X \longrightarrow S(\Gamma \times X)$.

Initially we focus on reactive processes, which are represented by the final weak coalgebra $\Xi = \nu X.\ (SX)^{\Sigma}$.

Assumption: Ω is S-algebra. It is assumed that Ω comes equipped with a canonical algebraic structure $S\Omega \longrightarrow \Omega$, which lifts to all $SMA \longrightarrow MA$ along the inclusion $MA \rightarrowtail \Omega^A$.

5 Testing semantics

The behaviors of processes from Ξ are captured by testing whether they satisfy formulas from Θ and observing the results in Ω via $\Xi \times \Theta \xrightarrow{\mathbb{T}} \Omega$. However, since Ξ and Θ generally live in the different universes $\mathcal{S}$ and $\mathcal{T}$, respectively, their interaction can only be observed using the connection between these universes, in one of the two forms:

$$\frac{\Xi \xrightarrow{\models} \Omega^{\Theta}}{\Theta \xrightarrow{=\!|} \Omega^{\Xi}}$$

In general, given a coalgebra $X \longrightarrow GX$, and an algebra $A \longleftarrow FA$, we define two semantic maps

$$\frac{X \xrightarrow{\models} MA}{A \xrightarrow{=\!|} PX}$$

connected by the adjunction. Each state $x \in X$ induces a map $x \models (-) : A \longrightarrow \Omega$ which maps each piece of data $a \in A$ to the observation $(x \models a) \in \Omega$ in which the computation of x on a will result. Dually, each piece of data $a \in A$ induces a map

$$a =\!| (-) \in PX \longrightarrow \Omega^X$$

[8] Anticipating semantics, we point out that the execution is always allowed to continue beyond a final state. This is in contrast with the *deadlock* states, which are represented by a choice functor G of the form $G = 1 + G'$. The deadlock states of a coalgebra $X \longrightarrow 1 + G'X$ are those that get mapped into 1.

which gives for each state $x \in X$ the observation $a \Dashv x$. Theorem 1 below describes how these various views of semantics transform the algebraic structure of tests and the coalgebraic structure of processes.

5.1 Connecting algebras and coalgebras: Representation theorem

Logical view. The logical operation of negation can be viewed as a very special case of a connection: if $\mathbb{A}$ is a pseudocomplemented lattice (Heyting algebra), then $\neg^{op} \vdash \neg : \mathbb{A}^{op} \longrightarrow \mathbb{A}$ is clearly a connection. Indeed, for every $\omega \in \mathbb{A}$, the operation $(-) \Rightarrow \omega : \mathbb{A}^{op} \longrightarrow \mathbb{A}$ is self adjoint. In posets and lattices, functors $F, G : \mathbb{A} \longrightarrow \mathbb{A}$ are monotone operators, algebras are super-fixpoints $a \geq Fa$, and colagebras are sub-fixpoints $a \leq Ga$; the initial algebra $\mu x.Fx$ is the least fixpoint, and the final coalgebra $\nu x.Gx$ is the greatest fixpoint.

For logical intuition, connections can be thought of as generalisations of negation. From that perspective, the following theorem can be viewed as a categorical elaboration of the fact that

$$\frac{Gx \leq \neg F \neg x}{\nu x.Gx \leq \nu x.\neg F \neg x \leq \neg \mu a.Fa}$$

What is the relevance of this fact? As explained in the introduction, the goal of this work is to explore the interplay of algebra and coalgebra in the theory of processes and in the practice of system specification. In practice, the behavior of a system is often specified as a quotient of a final coalgebra $\nu X.GX$ of processes using an initial algebra $\mu A.FA$ of tests. The connection $M^{op} \dashv P : \mathcal{S}^{op} \longrightarrow \mathcal{T}$ now allows deriving the semantics $\nu X.GX \xrightarrow{\models} M\mu X.FX$ if there is a distributive law $FP \longrightarrow PG$, i.e.

$$\frac{G \longrightarrow MFP}{\nu X.GX \longrightarrow \nu X.MFPX \longrightarrow M\mu A.FA}$$

The specified behavior is then the MFP-coalgebra L which is the image of $\nu X.GX$ in $\nu X.MFPX$. Furthermore, the carrier L can be conveniently represented as a subobject of $M\mu A.FA$. Informally, this is the content of the next theorem.

Relating a MFP-coalgebra and a M-image of a F-algebra requires a homomorphism which is consistent with the algebra and coalgebra structures both on the covariant and on the contravariant side of the correspondence (i.e., the "negation"). This is captured by the notion of *twisted* coalgebra homomorphisms, defined in the statement of the theorem.

Theorem 1. [9] *For a connection $M^{op} \dashv P : \mathcal{S}^{op} \longrightarrow \mathcal{T}$, endofunctors $G : \mathcal{S} \longrightarrow \mathcal{S}$ and $F : \mathcal{T} \longrightarrow \mathcal{T}$, and a distributive law $\lambda : FP \longrightarrow PG$ the following hold.*

[9] For simplicity and generality of the statement of the theorem, we avoid the finality and the initiality requirements, and spell out just the relations of F-algebras, and $G-$ and MFP-coalgebras.

(a) *The predicate functor* $P : \mathcal{S}^{op} \longrightarrow \mathcal{T}$ *lifts to* $\widehat{P} : (\mathcal{S}_G)^{op} \longrightarrow {}_F\mathcal{T}$, *mapping*

$$\frac{X \xrightarrow{\partial} GX}{\widehat{P}\partial : FPX \xrightarrow{\lambda} PGX \xrightarrow{P\partial} PX}$$

(b) $\widehat{P}$ *does not generally have an adjoint, but there is a correspondence of algebra homomorphisms and of* twisted *coalgebra homomorphisms*

$$\frac{\alpha \longrightarrow \widehat{P}\partial}{\Lambda\partial \longrightarrow M\alpha}$$

where $\Lambda : \mathcal{S}_G \longrightarrow \mathcal{S}_{MFP}$ *is the functor mapping the coalgebra* $X \xrightarrow{\partial} GX$ *to* $X \xrightarrow{\partial} GX \xrightarrow{\lambda'} MFPX$.

$$\begin{tikzcd}
FA \arrow[r, "Ff"] \arrow[dd, "\alpha"'] & FPX \arrow[d, "\lambda"] \\
& PGX \arrow[d, "P\partial"] \\
A \arrow[r, "f"'] & PX
\end{tikzcd}
\qquad
\begin{tikzcd}
MFPX \arrow[r, "MFf"] & MFA \\
GX \arrow[u, "\lambda'"] & \\
X \arrow[u, "\partial"] \arrow[r, "f'"'] & MA \arrow[uu, "M\alpha"']
\end{tikzcd}$$

(c) *If* $\mathcal{T}$ *is a regular category, and* $F : \mathcal{T} \longrightarrow \mathcal{T}$ *preserves reflective coequalizers, then* ${}_F\mathcal{T}$ *is a regular category. In particular, every* F*-algebra homomorphism* $\alpha \xrightarrow{f} \widehat{P}\partial$ *has a regular epi-mono factorisation.*

(d) *If* $\mathcal{S}$ *is a regular category, and* MFP *preserves weak pullbacks, then every twisted coalgebra homomorphism* $\Lambda\partial \xrightarrow{f'} M\alpha$ *has a regular epi-mono factorisation, which induces a coalgebra* $\ell : L \longrightarrow MFPL$ *as the image of* $\Lambda\partial$.

$$\begin{tikzcd}
MFPX \arrow[r, "MFPe"] & MFPL \arrow[r, "MFm'"] & MFA \\
GX \arrow[u, "\lambda'"] & & \\
X \arrow[u, "\partial"] \arrow[r, two heads, "e"'] & L \arrow[uu, dashed, "\ell"] \arrow[r, hook, "m"'] & MA \arrow[uu, "M\alpha"']
\end{tikzcd}$$

(e) *If the coalgebra* $X \longrightarrow GX$ *is final, then the coalgebra* $L \longrightarrow MFPL$ *is final if and only if the functor* $\Lambda : \mathcal{S}_G \longrightarrow \mathcal{S}_{MFP}$ *is essentially surjective.*

Comment. The correspondence $\mathcal{T}/P \cong \mathcal{S}/M^{op}$ thus lifts to ${}_F\mathcal{T}/\widehat{P} \cong \Lambda/M$, where the last denotes the comma construction for twisted homomorphisms. Abstractly, this does not seem like a very natural construction; the examples show that this is the ubiquitous framework where quotienting of the G-coalgebras ∂ induced by testing semantics takes place.

Definition 3. *A* duality *is a connection where the functors M and P are equivalences.*

Corollary 1. *Suppose that the connection $M^{op} \dashv P : \mathcal{S}^{op} \longrightarrow \mathcal{T}$ is a duality. Then the following are true.*

(f) *The algebra $\alpha : FA \longrightarrow A$ is initial if and only if the coalgebra $M(\alpha \circ \varepsilon) : MA \longrightarrow MFPMA$ is final. When that is the case, then the behavior $\ell : L \longrightarrow MFPL$ is a subcoalgebra of the final MFP-coalgebra.*

(g) *If $F \cong PGM$ (equivalently $G \cong MFP$), then the behavior $\ell : L \longrightarrow MFPL$, constructed in Theorem 1, is isomorphic to the coalgebra $\partial : X \longrightarrow GX$. If $\partial : X \longrightarrow GX$ is final and $\alpha : FA \longrightarrow A$ is initial, then $\partial \cong M\alpha$ and $\alpha = P\partial$.*

In many cases, the functor $F = PGM$ has a simpler representation than G, and the initial algebra $\Theta = \mu A.FA$ is easier to construct in $\mathcal{T}$ than the final coalgebra of $\Xi = \nu X.GX$ is in $\mathcal{S}$. In such cases, the isomorphism $\Xi = M\Theta$ offers significant technical advantages [4].

5.2 Specifying semantics

Given a coalgebra $X \xrightarrow{\partial} GX$ and the initial test algebra $F\Theta \xrightarrow{\varrho} \Theta$, we define a semantics $X \xrightarrow{\models} M\Theta$ by induction over Θ, using the fact that Ω is a T-algebra in $\mathcal{T}$ and an S-algebra in $\mathcal{S}$ — i.e. that each PX is a T-algebra in $\mathcal{T}$, whereas each $M\Theta$ is an S-algebra in $\mathcal{S}$. Given an initial state x of a machine X, we define a map $x \models (-) : \Theta \longrightarrow \Omega$.

Loose tests Since an element of $\Theta_0 = \mu X.\ TX + \Sigma \times X$ is in the form

$$t ::= \top \mid f(t_0 \dots t_n) \mid a.t$$

where $\top$ is the distinguished constant of the algebraic theory of the monad T, and f is an operation from that theory

$$\big(x \models \top\big) = \top \tag{1}$$

$$\big(x \models f(t_0 \dots t_n)\big) = f\big((x \models t_0) \dots (x \models t_n)\big) \tag{2}$$

$$\big(x \models a.t\big) = \big(\delta(x,a) \models t\big) \tag{3}$$

where $\delta : X \times \Sigma \longrightarrow SX$ is the transpose of $X \xrightarrow{\partial} GX = (SX)^{\Sigma}$, and $\models$ extends along $X \xrightarrow{\models} SX \longrightarrow M\Theta_0 \longrightarrow \Omega^{\Theta_0}$.

Remark. Clauses (1) and (2) say that $x \models (-) : A \longrightarrow \Omega$ is a T-algebra homomorphism. Clause (3) extends $x \models (-)$ beyond $T\Theta_0$ to $\Sigma \times \Theta_0$, using the fact that Ω (viz $M\Theta_0$) is an S-algebra, and extending $X \xrightarrow{\models} M\Theta_0$ to an S-algebra homomorphism $SX \xrightarrow{\models} M\Theta_0$.

Tight tests Since an element of $\Theta_1 = \mu X.\ T(\Sigma \times X)$ is in the form

$$t ::= \top \mid f(a_0.t_0 \ldots a_n.t_n)$$

the semantics retains clause 1, deletes clause 3, and replaces clause 2 with

$$\big(x \models f(a_0.t_0 \ldots a_n.t_n)\big) = f\big((\delta(x, a_0) \models t_0) \ldots (\delta(x, a_n) \models t_n)\big)$$

Note further that testing a coalgebra $X \xrightarrow{\langle \Phi, \partial \rangle} \Omega \times GX$, where $\Phi : X \longrightarrow \Omega$ denotes the final states, changes the base clause of semantics to $\big(x \models \top\big) = \Phi(x)$.

6 Possibilistic semantics

Possibilistic semantics is evaluated in $\Omega = \{0, 1\}$. In the simplest case, both state spaces and data types are modeled in the same universe $\mathcal{S} = \mathcal{T} = \mathsf{Set}$ of sets and functions. The contravariant powerset functor is self-adjoint $\wp^{op} \dashv \wp : \mathsf{Set}^{op} \longrightarrow \mathsf{Set}$, and maps a state to the type of predicates over it, and a type to the space of its models.[10]

Possibilistic systems Possibilistic nondeterminism means that there can be several possible transitions from a state $x \in X$, for a given action $a \in \Sigma$. The choice monad is thus based on the (covariant) finite powerset functor $S = \wp_f : \mathcal{S} \longrightarrow \mathcal{S}$. Simple processes are thus coalgebras in the form $X \longrightarrow (\wp_f X)^{\Sigma}$, or $X \longrightarrow \wp_f(\Sigma \times X)$.

6.1 Linear semantics: trace testing

A trace semantics describes computations over strings of symbols. The tests are thus pure modal formulas, with no logical operations except the constant $\top$. The logic monad is thus the smallest possible: $TA = \top$, for all $A \in \mathcal{T} = \mathsf{Set}$. The loose and the tight semantics for it coincide, and the test algebra Θ is initial for $FA = 1 + \Sigma \times A$, i.e. the free monoid Σ^*. Trace semantics have been investigated as an extension of coalgebraic methods in [8, 9]. We describe three examples.

[10] In the Hennessy-De Nicola [15] style testing semantics, tests are a special class of processes. In our testing framework, this means that tests and processes live in the same universe $\mathcal{S} = \mathcal{T}$, and moreover that the test algebra $F\Theta \longrightarrow \Theta$ is contained in (can be completed to) a choice coalgebra $\Xi \longrightarrow G\Xi$. Indeed, the trace algebra $\Theta = \Sigma^*$ is a coalgebra $\Sigma^* \longrightarrow \Sigma \times \Sigma^* \longrightarrow \wp_f(\Sigma \times \Sigma^*)$.

Finite state automata Possibilistic automata are coalgebras in the form $X \xrightarrow{\langle \Phi, \partial \rangle} 2 \times \wp_f(\Sigma \times X)$. The trace semantics of finite state automata is obtained by instantiating (1-3)

$$(x \models \top) = \Phi(x) \tag{4}$$

$$(x \models a.t) = \bigvee_{x \xrightarrow{a} y} (y \models t) \tag{5}$$

where $x \xrightarrow{a} y$ means that $y \in \delta(x, a)$. Note that (4) says that $x \models (-) : \Theta \longrightarrow \Omega$ only preserves $\top$ where Φ holds. The final states Φ are an explicit relativisation of the $\top$-preservation requirement; the semantics $x \models (-) : \Theta \longrightarrow \Omega$ is a T-algebra homomorphism up to Φ.

Let Aut denote bisimulation classes of finite-state automata, let $GX = 2 \times \wp_f(\Sigma \times X) \cong \wp_f(1 + \Sigma \times X)$, and let $\mathsf{Aut} \longrightarrow G(\mathsf{Aut})$ be final for all *finite* G-coalgebras. Then the trace semantics $\mathsf{Aut} \xrightarrow{\models} \wp\Sigma^*$ maps each automaton $x \in \mathsf{Aut}$ to the language $L_x = \{\sigma \in \Sigma^* \mid x \models \sigma\}$.

Pushdown automata While finite state automata behaviors were obtained by structuring the alphabet Σ, pushdown automata are obtained by structuring the state spaces X. Fix a set Γ, to be used as "non-terminal" symbols, and extend each state space X by the free monoid action to $X \times \Gamma^*$. A pushdown automaton is a coalgebra for the functor $G : \mathcal{S} \longrightarrow \mathcal{S}$, defined

$$GX = 2 \times \wp_f(X \times \Gamma^*)^{\Sigma+1}$$

where the "blank" symbol $\sqcup \in 1$ allows pure non-terminal rewrites. A start non-terminal symbol $Z_0 \in \Gamma$ is assumed to be distinguished, or freely added. The test algebra is still the same, $\Theta = \Sigma^*$.

Turing Machines Turing machines act on tapes. The obvious idea is to view the contents of a tape as a test. The problem is that the essential property of the tape is that it can be extended in both directions, so at the first sight, the Turing machine interaction does not seem not fit naturally into the inductive testing framework.

Another look at the acceptance condition for Turing machines offers a solution. A Turing machine X accepts a word $t \in \Sigma^*$ if and only if reaches a final state, in any configuration, after having started a computation with the head just to the left of the word t, presented on the tape. — So the accepted words initially extend to the right of the head. The left part of the tape is only used for intermediary computation.

A Turing machine can thus be modeled following the idea of a pushdown automaton: the tape to the left of the head can be viewed as a *stack*, and treated as a part of the state; the tape to the right of the head can be construed as another stack, containing the actual test. Unlike a pushdown automaton, a Turing machine allows words in the same alphabet in both stacks. A pushdown automaton had two disjoint alphabets, Γ and Σ for the left and the right stack, respectively.

Moreover, the right "stack" of a pushdown automaton is not a real stack, since it only allows popping.

A Turing machine can thus be viewed as a machine with two real stacks, representing the two parts of its tape, on the two sides of the head. Just like a pushdown automaton, besides the alphabet Σ, it may allow non-terminal symbols, at least $\sqcup$, used in computation, but not in the tested words.

A nondeterministic Turing machine is thus a coalgebra $X \xrightarrow{\langle\Phi,\partial\rangle} 2 \times \wp_f(X \times \Gamma \times \{\lhd, \rhd\})^{\Gamma}$, where $\Gamma \supseteq \Sigma + \{\sqcup\}$. As before, the component $X \xrightarrow{\Phi} 2$ marks the final states, whereas the transition function $X \times \Gamma \xrightarrow{\delta} \wp_f(X \times \Gamma \times \{\lhd, \rhd\})$ assigns to each state and each input the possible next states, outputs, and the direction for the move of the head. We represent the move of the head by popping a symbol from one stack and pushing it onto the other.

6.2 Branching semantics: set-tree testing

Here not only are the universes $\mathcal{S}$ and $\mathcal{T}$ identical, but we also take the logic monad T on to be the same as the choice monad S: they are both the finite powerset $\wp_f : \mathsf{Set} \longrightarrow \mathsf{Set}$. So both the space of the choices $\wp_f X$ and the logic of tests $\wp_f A$ are free semilattices. But the two lattices will be used differently: the former as a join semilattice (because the process can continue with this computation *or* with that computation...), and the latter as a meet semilattice (because the testing formula is a conjunction).

Remark. The same class of computational behaviors could be formalized by taking either of the monads T and S, or both of them, to be the diagonal functor $\Delta X = X \times X$. This would just mean that nondeterministic branching would always be binary, and that tests would be just binary conjunctions. Associativity, commutativity and idempotence of these operations would be imposed later. The intermediary options would be to take the functor $\wp_{\leq 2} X = \{x_0, x_1\}$ of (at most) two-element subsets, imposing commutativity and idempotence, and leaving out associativity.

Two-way simulation In the simplest case $TA = \wp_f A$. The tests are thus in the form

$$t ::= \top \mid t \wedge t \cdots \wedge t \mid a.t$$

where $\wedge$ is an associative, commutative, idempotent operation with unit $\top$. The semantics (1-3) becomes

$$\begin{aligned}(x \models \top) &= \top \\ (x \models \bigwedge_{i=1}^{n} t_i) &= \bigwedge_{i=1}^{n} (x \models t_i) \\ (x \models a.t) &= \bigvee_{y \in \delta_\kappa(x,a)} (y \models t)\end{aligned}$$

The functors generating data and processes are thus

$$FA = \wp_f A + \Sigma \times A$$
$$GX = \wp_f(\Sigma \times X)$$

Proposition 2. *Let Θ be the initial F-algebra and Ξ the final G-coalgebra. Let the partial order on X be defined by*

$$x \leq y \iff \forall t \in \Theta.\ (x \models t) \leq (y \models t) \tag{6}$$

Then the process x can be simulated by the process y if and only if $x \leq y$, i.e.

$$x \leq y \iff \forall a \in \Sigma \forall x' \in \delta(x,a) \exists y' \in \delta(y,a).\ x' \leq y' \tag{7}$$

Bisimulation Adding negation to the logic

$$t ::= \top \mid t \wedge t \cdots \wedge t \mid \neg t \mid a.t$$

i.e. testing by

$$FA = \wp_f A + A + \Sigma \times A$$

the semantics is extended by the clause

$$(x \models \neg t) = \neg(x \models t)$$

This gives an interesting strengthening of the testing power.

Proposition 3. *The equivalence*

$$x \sim y \iff \forall t \in \Theta.\ (x \models t) = (y \models t)$$

means just that the processes x and y are bisimilar

$$\begin{aligned} x \sim y \iff & \forall a \in \Sigma \\ & (\forall x' \in \delta(x,a) \exists y' \in \delta(y,a).\ x' \sim y') \wedge \\ & (\forall y' \in \delta(y,a) \exists x' \in \delta(y,a).\ x' \sim y') \end{aligned}$$

Strong bisimulation by Stone duality Strong bisimilarity is classified by the final coalgebra $\Xi \longrightarrow \wp_f(\Sigma \times \Xi)$. Using the restriction of the Stone duality $S \dashv C : \mathsf{Set}^{op} \longrightarrow \mathsf{caBa}$ from Stone spaces and Boolean algebras to sets (discrete spaces) and complete atomic Boolean algebras, allows applying Corollary 1. Setting $FA = C\wp_f(\Sigma \times SA)$ allows a representation of the bisimulation classes as characters of the boolean algebra $\Theta = \mu A.FA$.

7 Summary, Related and Future Work

We have proposed combining algebras as denotational models of processes with coalgebras as models of their testing regimes via logical connections between the two, in order to realize models that capture the interactions of processes with the data on which they operate, as well as to model the processes themselves. Our Main Thereom 1 is the key to this approach. In addition, we have illustrated our approach with standard examples: finite automata, pushdown automata and Turing machines, as well as results that show how our approach captures simulation and strong bisimulation of processes in concurrency.

Of course, there is a wealth of work using duality theory – especially Stone Duality – emanating from the seminal work of Abramsky [16] and the work on coalgebras exemplified by Rutten's [17] and the work of Plotkin and Turi [18]. None of these works has the same aims as our work. Indeed, the work along this line has been aimed at achieving a setting in which both operational and denotational models of the same language or process algebra could be presented and related.

The closest work – in theory at least – to what we have presented is that of Kupke, Kurz and Pattinson [19] and of Bonsangue and Kurz [20]. The former uses similar theoretical machinery – but restricted to duality theories, and applies it to the study of finitary modal logics as specification languages for Set-coalgebras. The latter also models transition systems as coalgebras, and then uses a duality to arrive at a logic for the transition system. Their main result is the soundness, completeness and expressiveness of the logic. They also extend to the setting of Vietoris coalgebras on topological spaces, and apply it to derive adequate logics on posets, sets, spectral spaces and Stone spaces. The logics in these works employ the usual modalities, possibility and necessity. In addition, the results rely heavily on the duality theory to transfer initial algebras to final coalgebras and back.

As we stated above, our approach has a rather different goal, and employs weaker assumptions. Our goal is to understand the interactions of a state machine and the data on which it operates during computation. These are fundamentally different objects – programs are executed, but data are not. Our work is based on logical connections, and does not require a duality theory. In fact, one could argue that our results begin when the connection used is not a duality. In addition, we are dealing with process logics where process formulas are a possible interpretation of tests: a process modality $\langle a \rangle$ is a test assigned to the action $a \in \Sigma$. That said, we believe the present paper has just begun to scratch the surface, and a lot remains to do. A primary goal is to present probabilistic systems from this perspective, and, in particular to apply it to extend the work in [4], as well as to explore the relationship between probability and nondeterminism, as presented, e.g., in [21].

References

1. Moore, E.F.: Gedanken experiments on sequential machines. In: Automata Studies, Princeton (1956) 129–153

2. Holzmann, G.J.: Design and Validation of Computer Protocols. Software Series. Prentice Hall, London (1991)
3. Lee, D., Yannakakis, M.: Principles and methods of testing finite state machines - A survey. In: Proceedings of the IEEE. Volume 84. (1996) 1090–1126
4. Mislove, M., Ouaknine, J., Pavlovic, D., Worrell, J.: Duality for labelled Markov processes. In Walukiewicz, I., ed.: Proceedings of FoSSaCS 2004. Volume 2987 of Lecture Notes in Computer Science., Springer Verlag (2004) 393–407
5. Pavlovic, D., Smith, D.R.: Composition and refinement of behavioral specifications. In: Automated Software Engineering 2001. The Sixteenth International Conference on Automated Software Engineering, IEEE (2001)
6. Pavlovic, D., Smith, D.R.: Guarded transitions in evolving specifications. In Kirchner, H., Ringeissen, C., eds.: Proceedings of AMAST 2002. Volume 2422 of Lecture Notes in Computer Science., Springer Verlag (2002) 411–425
7. Pavlovic, D., Pepper, P., Smith, D.R.: Colimits for concurrent collectors. In Dershowitz, N., ed.: Verification — Theory and Practice. Essays Dedicated to Zohar Mana on the Occasion of His 64th Birthday. Volume 2772 of Lecture Notes in Computer Science., Springer Verlag (2003) 568–597
8. Jacobs, B.: Trace semantics for coalgebras. In: Coalgebraic Methods in Computer Science (CMCS) 2004. Volume 106 of Electr. Notes in Theor. Comp. Sci. (2004)
9. Hasuo, I., Jacobs, B.: Context-free languages via coalgebraic trace semantics. In Fiadeiro, J., Harman, N., Roggenbach, M., Rutten, J., eds.: Algebra and Coalgebra in Computer Science (CALCO'05). Volume 3629 of Lecture Notes in Computer Science., Berlin, Springer-Verlag (2005) 213–231
10. Johnstone, P.: Stone Spaces. Number 3 in Cambridge Studies in Advanced Mathematics. Cambridge University Press (1982)
11. Abramsky, S.: Domain theory in logical form. Annals of Pure and Applied Logic **51** (1991) 1–77
12. Milner, R.: Communication and concurrency. International Series in Computer Science. Prentice Hall, London (1989)
13. Rutten, J.: Behavioral differential equations: a coinductive calculus of streams, automata and power series. Theor. Comp. Sci. **308** (2003) 1–53
14. Glabbeek, R., Smolka, S.A., Steffen, B.: Reactive, generative, and stratified models of probabilistic processes. Inf. Comput. **121**(1) (1995) 59–80
15. DeNicola, R., Hennessy, M.: Testing equivalences for processes. Theoretical Computer Science **34** (1984) :83–133
16. Abramsky, S.: A domain equation for bisimulation. Information and Computation **92** (1991) 161–218
17. Rutten, J.: Universal coalgebra: a theory of systems. Theor. Comp. Sci. **249** (2000) 3–80
18. Plotkin, G.D., Turi, D.: Towards a mathematical operational semantics. (In: Proceedings of LICS)
19. C. Kupke, A.K., Pattinson, D.: Ultrafilter extensions for coalgebras. In: Proceedings of CALCO. Volume 3629 of Lecture Notes in Computer Science. (2005) 263–277
20. Bonsangue, M., Kurz, A.: Duality for logics of transition systems. In: Proceedings of FoSSaCS. Volume 3441 of Lecture Notes in Computer Science. (2005) 455–469
21. Varacca, D.: The powerdomain of indexed valuations. In: Proceedings 17th IEEE Symposium on Logic in Computer Science, IEEE Press (2002)