Testing Semantics: Connecting Processes and Process Logics

Dusko Pavlovic[1] and Michael Mislove[2]

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

Abstract. Early approaches to semantics utilized denotational models to reason abstractly about programming languages. However, there also was the need to provide an operational view of processes, which was captured early on by Plotkin's SOS style. More recently this approach has been put on an equal footing with denotational semantics with the realization that coalgebras capture this view of process behavior quite precisely.
What remains missing from these approaches is an effective method for capturing the interactions of a machine represented in a denotational model and the data it manipulates. In this paper, we propose a methodology based on testing as a framework to capture these interactions. Using a duality that models machines on the one hand, and the data they manipulate on the other, testing is then 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 $\varXi$ of systems by a family $\varTheta$ of tests, or process logic formulas, is a map

$$\varXi \times \varTheta \xrightarrow{\mathbb{T}} \varOmega$$

where $\varOmega$ is the type of observations, or truth values. The simplest case is $\varOmega = \{0, 1\}$, where 1 represents "accept", or "succeed", or "truth", and 0 is "reject", or

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

"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 presented in this fashion 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 $R \in \varXi$ from an ideal reference system $S \in \varXi$, or to demonstrate that they 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.[3] 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 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

[3] Excellent surveys of testing methodologies (albeit a bit outdated in applications) are [2, 3].

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.

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:

Stone duality: $\mathcal{T}$ are Boolean algebras (viewed as propositional theories), whereas $\mathcal{S}$ are Stone spaces (whose points are the ultrafilters, i.e. models of Boolean propositional theories)

Topological spaces and complete Heyting algebras: $pt \dashv \mathcal{O} : \mathsf{Esp}^{op} \longrightarrow \mathsf{Frm}$ [10]

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}),

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 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 $Obs = |P1| = |M1|$.[4] For 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 isomorphim 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[5] both in $\mathcal{S}$ and in

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

[5] 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.

$\mathcal{T}$. 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 means that it passes the test that the formula represents.

The first and probably best known process logic is the 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 is given with a canonical algebraic structure $T\Omega \longrightarrow \Omega$, which lifts to all $TPX \longrightarrow PX$ along the inclusion $PX \rightarrowtail \Omega^X$.

3.1 Test theories

Test theories are obtained by extending T-algebras ("propositional theories") by the modal operators generated by Σ. A test theory is thus a weak algebra for either of the functors

$$F_0 X = TX + \Sigma \times X \qquad \text{or} \qquad F_1 X = T(\Sigma \times X)$$

The difference between the two is that each F_1-word is always a T-proposition, whereas an F_0-word can be a mere modal formula. In both cases, though, the universal test theory is obtained as the initial weak algebra

$$\Theta_i = \mu X.\ F_i X$$

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. It follows that Θ_1 is an algebra for the monad T, whereas Θ_0 is just a weak algebra for the functor T. 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* $\Sigma \times A \stackrel{\cdot}{\longrightarrow} A$*, 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

The choice constructors are represented by the monad $S : \mathcal{S} \longrightarrow \mathcal{T}$.

Definition 2. *A (state)* machine *with the inputs from* Σ*, the outputs from* Γ *and the 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.[6]

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].[7]

[6] 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.

[7] In Turing machines, some authors distinguish the accepting states, rejecting states and halting states; others allow families of initial states. All such families are given by suitable predicates.

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 is given 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

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

which gives for each state $x \in X$ the observation $a =\!\!| x$. 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 $F, G : \mathbb{A} \longrightarrow \mathcal{A}$ 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 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 $\nu X.GX \xrightarrow{\models} M\mu X.FX$ from the distributivity $FP \longrightarrow PG$, 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. [8] *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.*

(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*

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

[8] 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.

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

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

(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 $\varLambda\partial \xrightarrow{f'} M\alpha$ has a regular epi-mono factorisation, which induces a coalgebra $\ell : L \longrightarrow MFPL$ as the image of $\varLambda\partial$.*

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

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

Comment. The correspondence $\mathcal{T}/P \cong \mathcal{S}/M^{op}$ [9] thus lifts to ${}_F\mathcal{T}/\widehat{P} \cong \varLambda/M$, 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.*

[9] See the Appendix A for the relevant definitions.

(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* $\ell : L \longrightarrow MPL$ *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 Metrics on processes and tests

So far, testing semantics has been used to identify behaviorally equivalent processes, i.e. those $x, y \in X$ where $(x \models t) = (y \models t)$ holds for all tests $t \in A$. When the type Ω of observations has additional structure, testing can actually provide significantly more information. For example, if Ω supports a sup operation, then we can exploit this fact to define metrics on both the family of processes and on the family of tests. In the possibilistic setting, however, the metric captures the same information as the representation by behaviors. However, if the testing regime is extended to include probabilistic tests, then more information can be obtain. See Appendix B for further discussion.

5.3 Specifying semantics

Given a coalgebra $X \xrightarrow{\partial} GX$ and the initial test algebra $F\Theta \xrightarrow{\varrho} \Theta$, we define a semantics $X \xrightarrow{\models} MA$ 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 MA 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 $X \xrightarrow{\partial} (SX)^{\Sigma}$, and $\models$ extends along $X \xrightarrow{\models} SX \longrightarrow M\Theta \longrightarrow \Omega^{\Theta}$.

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

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 monid 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.

A more complete discussion of all three of these examples is contained in Appendix C

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

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

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 $\Theta = \mu A.FA$.

Weak bisimulation To model communication, we take[11] $\Sigma = \Sigma_+ + \Sigma_- + \{\tau\}$, and consider coalgebras for $GX = G_0X + G_1X$

$$G_0X = (\wp_f X)^\Sigma$$
$$G_1X = X \times X$$

For a coalgebra $X \xrightarrow{\partial} GX$, $\partial x \in G_0X$ means that x branches to $x \ \stackrel{a_0}{x}_0$, $x \ \stackrel{a_1}{x}_1 \ldots x \stackrel{a_n}{\to} x_n$, whereas $\partial x \in G_1X$ means that $x = x_0 \otimes x_1$. The modified part of semantics is now

$$(x \models a.t) = \begin{cases} \bigvee_{x \xrightarrow{a} y} (y \models t) & \text{if } \partial x \in G_0X \\ (x_0' \otimes x_1 \models t) & \text{if } \partial x = x_0 \otimes x_1 \wedge x_0 \stackrel{a}{\to} x_0' \\ \cdots & \end{cases}$$

7 Summary

This work has explored the interplay of algebra and coalgebra in theory and practice of process specification. Processes are often represented as coalgebras: state transitions are represented by a coalgebra's structure maps. Since coalgebra homomorphisms thus preserve and reflect state transitions, they capture computational behaviors. The final coalgebra thus gives a canonical representation of computational behaviors.

But this does not capture important models of computation that are richer than mere processes, but that also specify nontrivial interactions with data: e.g., Turing machines read and write on tapes, various classes of automata react to or generate strings of symbols, in many different ways. In some cases, the specified computational interactions induce constraints on the structure of processes, so that not all coalgebras correspond to actual machines. In other cases, the same process model underlies several different models of computation, realized through different interactions with data; the other way around, within the same model, different processes often have indistinguishable computational behaviors. To address this, we have found a precise representation methodology that captures the distinctions and the identifications that arise from the variety of computational models needed to explain real computational phenomena. This has been accomplished by combining coalgebraic descriptions of process with algebraic specification of the data they operate on, allowing an analysis of their interactions. Our Main Theorem 1 is key to this approach. A lot remains to do here. For example, presenting probabilistic systems from this perspective is a major goal, e.g., to understand better the work from [16] combining probability and nondeterminism.

[11] This is what is traditionally done. A conceptually more telling, albeit more complicated way, might be to take computational paths from the free *group*, rather than monoid, over a simple alphabet Σ.

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. Varacca, D.: The powerdomain of indexed valuations. In: Proceedings 17th IEEE Symposium on Logic in Computer Science, IEEE Press (2002)
17. Hyland, M.: A small complete category. Annals of Pure and Applied Logic **40** (1988) 135–165
18. Pavlovic, D.: On completeness and cocompleteness in an around small categories. Annals of Pure and Applied Logic **74** (1995) 121–152
19. Choquet, G.: Theory of capacities. Ann. Inst. Fourier (Grenoble) **5** (1953) 131–295
20. Jacobs, B.: A bialgebraic review of regular expressions, deterministic automata and languages. (In: Joseph Goguen Festschrift)
21. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, Reading, Massachusetts (1979)
22. Papadimitriou, C.M.: Computational Complexity. Addison-Wesley, Reading, Massachusetts (1994)

Appendix

A The Structure of connections

The total categories of a connection are

$$\begin{tikzcd}
\mathcal{T}/\Omega \arrow[d, "\cong"] & & \mathcal{S}/\Omega \arrow[d, "\cong"] \\
1/M^{op} \arrow[dr, hook] & & 1/P \arrow[dl, hook'] \\
& \mathcal{S}/M^{op} \cong \mathcal{T}/P \arrow[dl] \arrow[dr] & \\
\mathcal{S} & & \mathcal{T}
\end{tikzcd}$$

When $\mathcal{T} = \mathsf{Set}$, then $1/P$ is the "category of elements" of the presheaf $P : \mathcal{S}^{op} \longrightarrow \mathsf{Set}$, and the projection $1/P \longrightarrow \mathcal{S}$ the discrete fibration induced by this presheaf.

If the types in $\mathcal{T}$ carry some sort of category structure (e.g. they are monoids, or posets, or even discrete) then $P : \mathcal{S}^{op} \longrightarrow \mathcal{T}$ is an indexed category. If $1/P \cong \mathcal{S}/\Omega$ is taken to be a *lax* comma (in this case opslice), then the projection $\mathcal{S}/\Omega \longrightarrow \mathcal{S}$ is the Grothendieck fibration induced by the indexed category $P : \mathcal{S}^{op} \longrightarrow \mathcal{T}$, which is the "externalisation" [17, 18] of Ω as an internal category in $\mathcal{S}$.

B Testing Metrics

Iif the connection structure is viewed as a suitable completion construction, then Ω is a certain completion; at least a complete lattice. Using its supremum operation, the metric of Ω readily lifts to a metric on $MA \rightarrowtail \Omega^A$

$$d(f, g) = \bigvee_{t \in A} d(ft, gt)$$

which then lifts to a pseudometric on processes along the process semantics map $X \xrightarrow{\models} MA$, by setting $d(x, y) = d\big((x \models), (y \models)\big)$. The behavior coalgebra L, constructed in theorem 1, is just the quotient of X where this pseudometric becomes a metric. This *testing metric* provides valuable information about behaviors. *Even if two processes are not behaviorally indistinguishable, how different are they?* — The testing metric provides answers to such questions.

Note, however, that the testing pseudometric on processes induced by possibilistic testing captures exactly the same information as the representation of behaviors, since $d(x, y) \iff x \sim y$, i.e. if the processes x and y are indistinguishable. The testing metric on behaviors is thus trivial, with $d(x, y) = 0 \iff x = y$ and $d(x, y) = 1 \iff x \neq y$. The testing metric provides essential new information when it is induced by probabilistic testing, and more generally when $\Omega \neq \{0, 1\}$.

Studying the way the distance between processes is approximated by evaluating them on the various tests provides guidance for test selection and generation. In general every set of tests $U \subset A$ induces a *relative* testing pseudometric on processes

$$d_U(x,y) = \bigvee_{t \in U \subseteq A} d\big((x \models t), (y \models t)\big)$$

In an informal sense, the value

$$m(U) = \bigvee_{x,y \in X} \frac{d_U(x,y)}{d_A(x,y)}$$

measures the distinguishing power of the set of tests U. To formalize this, one could begin by restricting it to the σ-algebra on A which makes $A \xrightarrow{\models} PX$ measurable. The σ-algebra on $PX \rightarrowtail \Omega^X$ is induced from Ω. Although continuous, the map m will in general still not be a measure, but perhaps a Choquet capacity [19]. In any case, it carries the information useful for estimating the error of various testing suites.

Process distance on tests Dually, a test semantics map $A \xrightarrow{\models} PX$ induces a *process pseudometric* on tests $d(s,t) = \bigvee_{x \in X} d\big((x \models s), (x \models t)\big)$. By abstract category theory, it can be shown that the quotient of the test algebra A where the process pseudometric becomes a metric induces the same equivalence X as A does; and that the behavior coalgebra L induces the same equivalence on A as X. So identifying tests (or processes) at distance 0 does not change the equivalence of processes (resp. tests) induced on the other side. On the other hand, *increasing* the mutual distances of tests in a testing suite increases its distinguishing power. That is why the process pseudometric on tests can be useful for test suite construction, and for estimating the error in its results: *how closely do they approximate the true distance between processes?* If a test suite did not find any bugs (i.e. manage to distinguish the the real process from its idealized specification), how much do we know about the actual distance between these processes?

C Details of trace semantics

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)

$$(x \models \top) = \Phi(x) \tag{8}$$

$$(x \models a.t) = \bigvee_{x \xrightarrow{a} y} (y \models t) \tag{9}$$

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.

Representation. Let $\mathsf{Aut} \longrightarrow G(\mathsf{Aut})$ be final for all *finite* G-coalgebras, for $GX = 2 \times \wp_f(\Sigma \times X) \cong \wp_f(1 + \Sigma \times X)$. The elements of the underlying set Aut can be viewed as bisimulation classes of finite-state automata, or as finite Σ-labelled hypersets with a distinguished subset of finite states.

The trace semantics $\mathsf{Aut} \xrightarrow{\models} \wp\Sigma^*$ maps each automaton $x \in \mathsf{Aut}$ to the language $L_x = \{\sigma \in \Sigma^* \mid x \models \sigma\}$. The distribution $FP \longrightarrow PG$ is in this case the map

$$\begin{aligned} 1 + \Sigma \times \wp X &\longrightarrow \wp(2 \times \wp_f(\Sigma \times X)) \\ \top &\longmapsto \{\langle 1, \emptyset \rangle\} \\ \langle a, U \rangle &\longmapsto \left\{ \langle 0, \{a\} \times V \rangle \mid V \in \wp_f U \right\} \end{aligned}$$

i.e. a relation in which a pair $\langle a, U \rangle \in \Sigma \times \wp X$ corresponds with a finite set $\{\langle b, x \rangle \in \Sigma \times X\}$ if and only if all $b = a$ and $x \in U$. Postcomposing the coalgebra structure $\mathsf{Aut} \longrightarrow 2 \times \wp_f(\Sigma \times \mathsf{Aut})$ with the transpose $2 \times \wp_f(\Sigma \times X) \longrightarrow \wp\left(1 + \Sigma \times \wp(\mathsf{Aut})\right)$ of this relation, we get

$$\begin{aligned} \mathsf{Aut} &\longrightarrow \wp\left(1 + \Sigma \times \wp(\mathsf{Aut})\right) \\ x &\longmapsto \{\top \mid \Phi(x)\} \cup \{\langle a, U \rangle | \delta(a, x) \subseteq U\} \end{aligned}$$

The image of the semantics map $\mathsf{Aut} \xrightarrow{\models} \wp\Sigma^*$ is the set $\mathsf{Reg} \hookrightarrow \wp\Sigma^*$ of regular languages, which comes with the coalgebra structure

$$\mathsf{Reg} \longrightarrow \wp\left(1 + \Sigma \times \wp(\mathsf{Reg})\right) \quad \text{and} \quad L \longmapsto \{\top | \langle\rangle \in L\} \cup \{\langle a, U \rangle \,| \partial_a(L) \in U\}$$

where $\partial_a(L) = \{\sigma \in \Sigma^* |\ a.\sigma \in L\}$.

$$\begin{tikzcd}
\wp\left(1 + \Sigma \times \wp(\mathsf{Aut})\right) \arrow[r] & \wp\left(1 + \Sigma \times \wp(\mathsf{Reg})\right) \arrow[r] & \wp(1 + \Sigma \times \Sigma^*) \\
2 \times \wp_f(\Sigma \times \mathsf{Aut}) \arrow[u, "\lambda'"] & & \\
\mathsf{Aut} \arrow[u, "\partial"] \arrow[r, two heads] & \mathsf{Reg} \arrow[uu, "r"] \arrow[r, hook] & \wp\Sigma^* \arrow[uu, "\wp\alpha"']
\end{tikzcd}$$

Instantiating (1-3) again, we get canonical semantics for Reg

$$\big(L \models \top\big) = \begin{cases} \top & \text{if } \langle\rangle \in L \\ \bot & \text{otherwise} \end{cases} \quad \text{and} \quad \big(L \models a.t\big) = \bigwedge_{\langle a, U \rangle \in r(L)} \bigvee_{M \in U} M \models t$$

While Reg can be obtained directly, without any excursion into testing, as the final coalgebra for a suitable family of deterministic automata [20], the point here is to derive regular expressions as the behaviors of a class of automata which can be interpreted (tested) in many other ways, leading to different behaviors.

Pushdown automata While the various 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, $\Theta = \Sigma^*$.

The semantics of a pushdown automaton $X \times \Gamma^* \xrightarrow{\langle \Phi, \partial \rangle} 2 \times \wp_f (X \times \Gamma^*)^{\Sigma+1}$ will be

$$\begin{aligned}
\big(x, \gamma \models \langle\rangle\big) &= \Phi(x) \\
\big(x, \langle\rangle \models a.t\big) &= \bot \\
\big(x, Z.\gamma \models a.t\big) &= \bigvee_{y,\phi \in \partial(x,Z.\gamma)(a)} \big(y, \phi \cdot \gamma \models t\big) \\
&\vee \bigvee_{y,\phi \in \partial(x,Z.\gamma)(\sqcup)} \big(y, \phi \cdot \gamma \models a.t\big)
\end{aligned}$$

The induced T-homomorphism $\Sigma^* \xrightarrow{\dashv} PX$ transposes to $X \xrightarrow{\models} M(\Sigma^*)$. Taking X to be the final coalgebra $\mathsf{Psa} = \nu X.\ (\wp_f X \times \Gamma^*)^{\Sigma+1}$, we get the coalgebra Cfl of context free languages

$$\begin{tikzcd}
\wp\big(1 + \Sigma \times \wp(\mathsf{Psa})\big) \arrow[r] & \wp\big(1 + \Sigma \times \wp(\mathsf{Cfl})\big) \arrow[r] & \wp(1 + \Sigma \times \Sigma^*) \\
2 \times \Big(\wp_f(\mathsf{Psa} \times \Gamma^*)\Big)^{\Sigma+1} \arrow[u, "\lambda'"] & & \\
\mathsf{Psa} \arrow[u, "\langle\Phi,\partial\rangle"] \arrow[r, two heads] & \mathsf{Cfl} \arrow[uu] \arrow[r, hook] & \wp\Sigma^* \arrow[uu, "M\alpha"']
\end{tikzcd}$$

Note that the representation eliminates the non-terminals Γ, since the distributivity λ', connects the next states of a pushdown automaton with the derivatives of its accepted language. Specifying this distribution directly seems cumbersome, but it can also be obtained by precomposing with the inverse of the isomorphism $\langle \Phi, \partial \rangle$ the more intuitive coalgebra structure, induced by the accepted languages

$$\begin{aligned}
\mathsf{Psa} &\longrightarrow \wp\big(1 + \Sigma \times \wp(\mathsf{Psa})\big) \\
x &\longmapsto \{\top | \Phi(x)\} \cup \{\langle a, U\rangle \,|\exists y \in U.\ \partial_a L(x) = L(y)\}
\end{aligned}$$

Remark. Context free grammars have been studied in [9] as coalgebras for the functor

$$GX = \wp_f\big((\Sigma + X)^*\big)$$

Such coalgebras are however not interpreted as processs: the elements of the underlying set $x \in X$ are not states, but rather the nonterminal symbols from our Γ. The structural difference is that state transitions output (or input) symbols as they go, whereas rewrite rules output terminal symbols only at the end. This is reflected in the fact that processs are coalgebras for functors extending $\Sigma \times X$, whereas grammars are coalgebras for functors extending $\Sigma + X$.

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[12]; 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 $X \xrightarrow{\langle\Phi,\partial\rangle} 2 \times \wp_f(X \times \Gamma \times \{\triangleleft, \triangleright\})^\Gamma$, where $\Gamma \supseteq \Sigma + \{\sqcup\}$. As before, the component $X \xrightarrow{\Phi} 2$ marks the final states, whereas the transition function $X \times \Gamma \xrightarrow{\delta} \wp_f(X \times \Gamma \times \{\triangleleft, \triangleright\})$ 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.

However, to simplify definitions (avoid clauses dealing with popping from the empty stack) it is convenient to assume that the stacks are padded by $\sqcup$s to infinity. That is why they are usually thought of as infinite tapes. (In fact, taking just one of them to be infinite suffices [21].)

In order to write down the semantics of a Turning machine more conveniently, we adjoin the "infinite stacks"/"tapes" to the coalgebraic description of the machine:

$$\frac{X \times \Gamma \xrightarrow{\delta} \wp_f(X \times \Gamma \times \{\triangleleft, \triangleright\})}{X \times \Gamma^\oplus \xrightarrow{\delta^\oplus} \wp_f(X \times \Gamma^\oplus \times \{\triangleleft, \triangleright\})}$$

where

$$\Gamma^\oplus = \{\gamma \in \Gamma^\omega | \; |\gamma \setminus \sqcup| < \omega\}$$

i.e. each $\gamma \in \Gamma^\oplus$ has only a finite number of non-blank symbols.

$$\delta^\oplus(x, a.\gamma) = \{\langle y, b.\gamma, \Diamond\rangle | \langle y, b, \Diamond\rangle \in \delta(x, a)\}$$

[12] In some versions of Turing machines, the tapes are infinite only in one direction, but the head can still move in both directions, and read and write on both sides, which makes direct induction over the tape impossible.

The semantics is now[13]

$$
\begin{aligned}
(x, b.\gamma \models d.\varphi) = \quad & \Phi(x) \ \vee \\
\vee & \bigvee_{\langle y,c,\lhd\rangle \in \delta(x,b)} (y, \gamma \models c.d.\varphi) \\
\vee & \bigvee_{\langle y,c,\rhd\rangle \in \delta(x,b)} (y, c.b.\gamma\rangle \models \varphi)
\end{aligned}
$$

A trace semantics of a final coalgebra TM of the functor $GX = 2\times\wp_f(\Gamma\times X\times\{\lhd,\rhd\})^\Gamma$ is the set RE of recursively enumerable languages

$$
\begin{tikzcd}
\wp\left(1+\Sigma\times\wp(\mathsf{TM})\right) \arrow[rr] & & \wp\left(1+\Sigma\times\wp(\mathsf{RE})\right) \arrow[r] & \wp(1+\Sigma\times\Sigma^*) \\
2\times\left(\wp_f(\mathsf{TM}\times\Gamma\times\{\lhd,\rhd\})\right)^\Gamma \arrow[u, "\lambda'"] & & & \\
\mathsf{TM} \arrow[u, "\langle\Phi,\partial\rangle"] \arrow[rr, two heads] & & \mathsf{RE} \arrow[uu, "r"] \arrow[r, hook] & \wp\Sigma^* \arrow[uu, "M\alpha"']
\end{tikzcd}
$$

with the language coalgebra structure r as before

$$
\begin{aligned}
\mathsf{RE} & \longrightarrow \wp\left(1+\Sigma\times\wp(\mathsf{RE})\right) \\
L & \longmapsto \{\top|\langle\rangle\in L\}\cup\{\langle a,U\rangle\,|\partial_a(L)\in U\}
\end{aligned}
$$

Nondeterministic time-bounded Turing machines are based on state spaces in the form $X\times\mathbb{N}$. As explained in section 6.1, such objects can be viewed as monoid actions, i.e. algebras for the monad $\mathbb{N}\times(-)$. The requirement is that every computation from state $\langle x,n\rangle$ must halt after at most n steps. This requirement can be enforced by transforming the coalgebraic form of the machine

$$
\frac{X \xrightarrow{\langle\Phi,\partial\rangle} 2\times\wp_f(X\times\Gamma\times\{\lhd,\rhd\})}{X\times\mathbb{N} \xrightarrow{\langle\Phi_<,\partial_<\rangle} \wp_f(X\times\mathbb{N}\times\Gamma\times\{\lhd,\rhd\})}
$$

to

$$
\begin{aligned}
\Phi_<(x,n) &= \Phi(x) \\
\partial_<(x,0)(a) &= \emptyset \\
\partial_<(x,n+1)(a) &= \{\langle y,n,b,\Diamond\rangle|\langle y,b,\Diamond\rangle\in\partial(x)(a)\}
\end{aligned}
$$

The semantics of TM now induces a semantics of $\mathsf{TM}_<$

$$
\begin{aligned}
(x,0,\gamma \ \models \ \varphi) = \quad & \Phi(x) \\
(x,n+1,b.\gamma \ \models \ d.\varphi) = \quad & \Phi(x) \ \vee \\
\vee & \bigvee_{\langle y,c,\lhd\rangle\in\delta(x,b)} (y,n,\gamma \ \models \ c.d.\varphi) \\
\vee & \bigvee_{\langle y,c,\rhd\rangle\in\delta(x,b)} (y,n,c.b.\gamma \ \models \ \varphi)
\end{aligned}
$$

[13] We assume that the states are in the form $\langle x,\gamma\rangle\in X\times\Gamma^\oplus$, but write the semantic definition using the transition function δ, rather than $\delta^\oplus$.

Remark. A *precisely* bounded Turing machine is required to halt *precisely* n steps after a state $\langle x, n\rangle$. This is captured by setting

$$\Phi_<(x, n) = \begin{cases} \Phi(x) & \text{if } n = 0 \\ \perp & \text{otherwise} \end{cases} \tag{10}$$

and consequently dropping the "$\Phi(x)$" part from the "$n + 1$"-clause of the semantics. Precise and imprecise semantics distinguish the same class of languages [22, prop. 7.1].

Nondeterministic polynomial-time Turing machines[14] are based on state spaces in the form $X \times \mathbb{Z}[u]$, where $\mathbb{Z}[u]$ is the ring of polynomials with integer coefficients. The requirement is now that every computation from state $\langle x, p\rangle$ over a word α must halt after at most $p(|\alpha|)$ steps, where $|\alpha|$ is the length of α. Each Turing machine and every polynomial induce a polynomially-bounded machine

$$\frac{X \xrightarrow{\langle \Phi, \partial\rangle} 2 \times \wp_f(X \times \Gamma \times \{\lhd, \rhd\}) \qquad p(u) \in \mathbb{Z}[u]}{X \times \mathbb{Z}[u] \xrightarrow{\langle \Phi_{\mathsf{NP}}, \partial_{\mathsf{NP}}\rangle} \wp_f(X \times \mathbb{Z}[u] \times \Gamma \times \{\lhd, \rhd\})} \tag{11}$$

to

$$\begin{aligned} \Phi_{\mathsf{NP}}(x, p) &= \Phi(x) \\ \partial_{\mathsf{NP}}(x, p)(a) &= \{\langle y, p - 1, b, \Diamond\rangle | \langle y, b, \Diamond\rangle \in \partial(x)(a)\} \end{aligned}$$

with a semantics based on the time bounded semantics definition:

$$(x, p, \gamma \;\models\; \varphi) = \big(x, p(|\varphi|), \gamma \;\models\; \varphi\big)$$

Remark. The functions used to limit the complexity of computations need to be constrained: they need to be easily computable themselves, positive, etc. For simplicity, we just consider polynomials. Clearly, those that are not monotone increasing, or take negative values, induce trivial, unsatisfiable constraints on machines.

The coalgebra of the NP-languages is now obtained by testing the coalgebra[15] $\mathsf{TM}_{\mathsf{NP}} = \mathsf{TM} \times \mathbb{Z}[u]$

$$\begin{tikzcd}
\wp\left(1 + \Sigma \times \wp(\mathsf{TM}_{\mathsf{NP}})\right) \arrow[r] & \wp\left(1 + \Sigma \times \wp(\mathsf{NP})\right) \arrow[r] & \wp(1 + \Sigma \times \Sigma^*) \\
2 \times \left(\wp_f(\mathsf{TM}_{\mathsf{NP}} \times \Gamma \times \{\lhd, \rhd\})\right)^{\Gamma} \arrow[u, "\lambda'"] & & \\
\mathsf{TM}_{\mathsf{NP}} \arrow[u, "\langle \Phi_{\mathsf{NP}}, \partial_{\mathsf{NP}}\rangle"] \arrow[r, two heads] & \mathsf{NP} \arrow[uu, "r"'] \arrow[r, hook] & \wp\Sigma^* \arrow[uu, "M\alpha"']
\end{tikzcd}$$

---

[14] Deterministic polynomial-time Turing machines are in the form $X \xrightarrow{\langle \Phi, \partial\rangle} 2 \times (X \times \Gamma \times \{\lhd, \rhd\})^{\Gamma}$. Adjoining the time bounds and the polynomial bounds, in exactly the same way as to nondeterministic machines, leads to the language class P.

[15] This is a coalgebra in the Kleisli category for the monad $(-) \times \mathbb{Z}[u]$. It is not final, but the semantics definition does not require finality.

The structure $\mathsf{TM}_{\mathsf{NP}} \xrightarrow{\langle \Phi_{\mathsf{NP}}, \partial_{\mathsf{NP}} \rangle} 2 \times \left(\wp_f(\mathsf{TM}_{\mathsf{NP}} \times \Gamma \times \{\lhd, \rhd\}) \right)^{\Gamma}$ is derived from the structure $\langle \Phi, \partial \rangle$ by transformation (11).