Dynamic coalgebraic modalities

Dynamic Coalgebraic Modalities

Helle Hvid Hansen
Eindhoven University of Technology,
Centrum Wiskunde & Informatica, Amsterdam

Raul Andres Leal
Universiteit van Amsterdam

With this work we aim to place dynamic modal logics such as Propositional Dynamic Logic (PDL) [1] and Game Logic (GL) [4] in a uniform coalgebraic framework. In our view, a dynamic system $\mathbb{S}$ consists of the following ingredients:

1. A set S which represents the global states of $\mathbb{S}$.
2. An algebra L of labels (denoting actions, programs, games, ...).
3. An interpretation of labels as G-coalgebras on the state space S.
4. A collection of labelled modalities $[\alpha]$, for $\alpha \in L$, where intuitively $[\alpha]\varphi$ reads: "after α, φ holds".

Formally, the interpretation of labels is a map $\sigma\colon L \to (GS)^S$ which describes how actions change the global system state. The algebraic structure on L describes how one can compose actions into more complex ones. The same type of algebraic structure should be carried by $(GS)^S$, and we say that σ is *standard*, if σ is an algebra homomorphism, which means that the semantics of actions is compositional. By considering the exponential adjoint $\widehat{\sigma}\colon S \to (GS)^L$ we obtain a behavioural description of the system in the form of a G^L-coalgebra. These two (equivalent) views of a dynamic system form the basis of our modelling. In short, σ describes structure and dynamics, and $\widehat{\sigma}$ describes behaviour and induces modalities.

$\sigma\colon L \to (GS)^S$	$\widehat{\sigma}\colon S \to (GS)^L$
(algebraic view: structure, dynamics)	(coalgebraic view: behaviour, modalities)

A similar observation was made in [2] in the context of Java semantics, which we will return to later.

PDL (without tests) can be seen as an instance of the above by taking as L the set of program expressions, and as G the covariant powerset functor $\mathscr{P}$. The algebraic structure on $(\mathscr{P}S)^S$ is given by the operations on relations that define a standard PDL frame (cf. [1]). Similarly, GL (without tests) is obtained by taking G to be the monotonic neighbourhood functor $\mathscr{M}$. Observe that the PDL-modalities are just labelled versions of the usual $\mathscr{P}$-modality $\Box$. Generalising this construction, we obtain G^L-modalities by labelling G-modalities. More precisely, given $\alpha \in L$ and a predicate lifting $\lambda\colon \mathscr{Q} \to \mathscr{Q}G$ (where $\mathscr{Q}$ denotes the contravariant power set functor), we define the *α-labelling of λ* to be the predicate lifting with S-component

$$\lambda^\alpha_S\colon \mathscr{Q}S \to \mathscr{Q}(GS)^L; \quad U \mapsto \lambda^\alpha_S(U) = \{\delta \in (GS)^L \mid \delta(\alpha) \in \lambda_S(U)\}$$

Also the modalities in Game Logic and Java semantics (cf. [2]) arise as labelled modalities. Every predicate lifting $\lambda^\alpha \in \Lambda^L_G$ induces a predicate transformer $m_{\lambda^\alpha} = \sigma(\alpha)^{-1} \circ \lambda_S\colon \mathscr{Q}S \to \mathscr{Q}S$ on a system $\mathbb{S}$, and the truth set of a modal formula $[\alpha]_\lambda(\phi)$ in $\mathbb{S}$ is defined as $[\![[\alpha]_\lambda(\phi)]\!] = m_{\lambda^\alpha}([\![\phi]\!])$.

The structure of standard PDL frames is axiomatised by formulas such as $[\alpha;\beta]\phi \leftrightarrow [\alpha][\beta]\phi$ (SEQ) and $[\alpha \cup \beta]\phi \leftrightarrow [\alpha]\phi \wedge [\beta]\phi$ (CHOICE). These axioms display an interaction between the $\Box$-modality for $\mathscr{P}$ and the algebraic structure on programs. We would like to understand such axioms in our abstract setting. Our focus so far has been on the axiom (SEQ) for sequential composition. It is rather well known (see e.g. [3]) that sequential composition can be more generally understood as Kleisli composition for a monad. Indeed, for PDL (resp. GL) we have that $\sigma(\alpha;\beta) = \sigma(\alpha) * \sigma(\beta)$, where $*$ denotes Kleisli composition for $\mathscr{P}$ (resp. $\mathscr{M}$). Hence if G is a monad, then an operation ; can be defined on $(GS)^S$ as Kleisli composition for G. Since $[\alpha]_\lambda$ is interpreted by a predicate lifting λ^α for G^L, (SEQ) should be read parametric in the underlying predicate lifting λ. This is made explicit in the following formulation of (SEQ) in terms of predicate transformers.

Definition 1. A predicate lifting $\lambda : \mathscr{Q} \to \mathscr{Q}G$ for a monad G *captures sequential composition* if for all standard $\sigma : L \to (GS)^S$ and all $\alpha, \beta \in L$: $m_{\lambda^{\alpha;\beta}} = m_{\lambda^\alpha} \circ m_{\lambda^\beta}$.

We point out that Definition 1 is not vacuous. For example, the constant predicate lifting λ for $\mathscr{P}$ with value $\lambda_S(U) = \{\emptyset\}$ for all $U \subseteq S$ does not capture sequential composition. To see this, for a state $s \in S$ we have: $s \in m_{\lambda^{\alpha;\beta}}(U)$ iff $\sigma(\alpha;\beta)(s) = \emptyset$, while $s \in m_{\lambda^\alpha}(m_{\lambda^\beta}(U))$ iff $\sigma(\alpha)(s) = \emptyset$. Hence the inclusion from left to right fails in general. (Take for example $\sigma(\alpha) = S \times S$ and $\sigma(\beta) = \emptyset$.) A second (easy) observation is that λ captures sequential composition iff its boolean dual $\neg\lambda\neg$ does. Our main result up to now is the following characterisation.

Theorem 2. Let G be a monad. A predicate lifting $\lambda : \mathscr{Q} \to \mathscr{Q}G$ captures sequential composition iff its adjoint $\widehat{\lambda} : G \to \mathscr{Q}\mathscr{Q}$ is a monad morphism.

From this theorem, we can reprove the validity of (SEQ) for Game Logic: If $\lambda : \mathscr{Q} \to \mathscr{Q}\mathscr{M}$ is the predicate lifting for the monotonic box, then $\widehat{\lambda} : \mathscr{M} \to \mathscr{Q}\mathscr{Q}$ is simply the natural inclusion which is a monad morphism. Note that checking whether a natural transformation is a monad morphism is an algorithmic procedure.

Our dynamic systems can be seen as a special case of the framework presented in [2] to give semantics to Java programs. The main difference with PDL and GL is that Java programs manipulate data as well as effecting state change. In [2], a Java program with input in A and output in B is formalised as a map $A \to F(S,B)^S$ where $F : \mathit{Set} \times \mathit{Set} \to \mathit{Set}$ is a bifunctor such that for a fixed S, the functor $F(S,-)^S$ is a monad. In particular, sequential composition is Kleisli composition for $F(S,-)^S$. PDL is obtained in this framework by taking $A = B = 1$ (PDL programs have trivial input/output), and $F(S,B) = \mathscr{P}(B \times S)$. Taking the exponential adjoint, a Java program is also a coalgebra $S \to F(S,B)^A$, and thus gives rise to labelled $F(-,B)^A$-modalities and predicate transformers. However, since Java program operations are subject to typing conditions, and hence not totally defined, a standard interpretation can no longer be defined as an algebra homomorphism, and the collection of all programs does not form a labelled $F(-,B)^A$-coalgebra. Still we can generalise the notion of standardness and Definition 1 such that Theorem 2 still holds.

There are many questions we would like to address in the future. We have seen that if G is a monad, then $(GS)^S$ supports a notion of sequential composition. We would like to investigate the general relationship between properties of the functor G, operations supported by $(GS)^S$, and compositionality axioms. We also would like to develop new examples of dynamic modalities involving probabilistic systems. An issue that we left open was how to include the program/game construction of tests in our framework.

Acknowledgements We thank Bart Jacobs for suggesting this topic to us, and both Bart Jacobs and Yde Venema for stimulating discussions and helpful comments.

References

[1] R. Goldblatt. *Logics of Time and Computation*. Number 7 in CSLI Lecture Notes. Center for the Study of Language and Information, 2nd edition, 1992.

[2] B. Jacobs and E. Poll. Coalgebras and monads in the semantics of Java. *Theoretical Computer Science*, 291:329–349, 2003.

[3] E. Moggi. Notions of computation and monads. *Information and Computation*, 93(1):55–92, 1991.

[4] R. Parikh. The logic of games and its applications. In *Topics in the Theory of Computation*, number 14 in Annals of Discrete Mathematics. Elsevier, 1985.