Structure and Behaviour
(a roadmap for the Beijing Lectures)

Luís S. Barbosa

[1] CCTC — Centro de Ciências e Tecnologias da Computação
Departamento de Informática
Universidade do Minho
Braga, Portugal

lsb@di.uminho.pt

Modelling & Reasoning

The double face of Formal Methods. Lec 1 slides and & [9]

Modern software design *vs* school physics' basic *problem-solving strategy*:

- *understand* the problem
- *build* a mathematical *model* of it
- *reason* within such a model
- *upgrade* the model whenever necessary
- *calculate* a *solution* and *implement* it

Hence,

The modelling problem
- How are the *right* abstractions for a problem domain *built*?
- Requires *expressive* (often diagramatic) notations (*e.g.*, VDM [24, 20])

The reasoning problem
- To what extent do such abstractions enable *effective reasoning* about engineered systems as well as along the process of their construction?
- Requires *concise* (often cryptic) notations (*e.g.*, pointfree program calculi [18, 3])

Reasoning styles Lec 1 slides & [2]

Guess-and-verify or theorem-followed-by-proof
- oriented to system's *verification*: build a more concrete (*detailed, closer to the programming environment*, ...) model and than *check* it against the abstract model.

Goal-followed-by-construction
- oriented to system's *construction* and *understanding*: emphasis on the *derivation* more concrete models from abstract ones and on *seeking* for *generic* laws by suitable (essentially synthatic) calculations.

A categorial "parenthesis" Lec 1 slides & Note N1

- Categories as a framework to study semantic universes
- Set theory (*elements* and *membership*) *vs* Category theory [27] (*morphisms* and *composition*)
- Three essential notions:
  - *universality*: asserts the existence of a canonical reduction within a family of similar entities; entails both *existence* ($\leadsto$ definitional method) and *uniqueness* ($\leadsto$ proof principle).
  - *functoriality*: uniform transformation of both objects (types) and morphisms (programs).
  - *naturality*: polymorphic morphisms relating structural shapes (functors).
- Some good references: [1] (available free from the web), [16, 35] (computer science oriented), [26] (limited scope but good to build intuitions).

Coalgebras & Coinductive Reasoning

Algebras describe *assembly* processes. The emphasis is on *construction* The *initial algebra* is an algebra whose carrier is the set of terms for the signature captured by the underlying functor, *i.e.*, the *term algebra*, hence a *data type*.

$$[\mathsf{nil}, \mathsf{cons}] : \mathbf{1} + O \times L \longrightarrow L$$

In general:

a *tool box*: 

an *assembly process*:  artifact $\xrightarrow{d}$ artifact

Coalgebras describe *observation* structures (*i.e.*, transition systems). The emphasis is on *observation*. A *final coalgebra* is a coalgebra whose state space is the set of *behaviours* of all coalgebras for the underlying functor. Hence, a *behaviour type*.

$$\langle \mathsf{at}, \mathsf{m} \rangle : U \longrightarrow O \times U$$

In general:

a *lens*: 

an *observation structure*: universe $\xrightarrow{f}$  universe

A structural duality Lec 2 slides & Note N2

- A basic symmetry: *construction vs observation ...*
- ... leading to *informational vs behavioural* structures.
- Such symmetry is mathematically explained by the duality *algebra vs coalgebra*, which, in particular, allows us to go
- ... from *inductive* arguments (over *finite* structures, *e.g.*, A^*) to *coinductive* arguments (over *infinite* structures, *e.g.*, A^ω).
- References: [21] (very good, easy to follow tutorial introduction); [6] (small programming-oriented introduction); [32] (foundational paper); [25] (an introduction to the logic oriented reader); (application to stream calculus) [33, 34]; [23, 22] application to object-orientation.

Coinduction as a Universal Property Lec 2 slides & Note N3

$$\begin{array}{ccc} \nu\mathsf{T} & \xrightarrow{\omega_\mathsf{T}} & \mathsf{T}\nu\mathsf{T} \\ {\scriptstyle [\![p]\!]}\uparrow & & \uparrow{\scriptstyle \mathsf{T}[\![p]\!]} \\ U & \xrightarrow{p} & \mathsf{T}U \end{array}$$

The universal property is equivalently captured by the following law:

$$k = [\![p]\!] \quad \Leftrightarrow \quad \omega_\mathsf{T} \cdot k = \mathsf{T}\ k \cdot p$$

- **Existence** $\equiv$ **definition** principle (co-recursion)
- **Uniqueness** $\equiv$ **proof** principle (co-induction)

From which a number of useful laws can be derived:

$$\begin{array}{rrcl} \textit{cancellation} & \omega_\mathsf{T} \cdot [\![p]\!] & = & \mathsf{T}\ [\![p]\!] \cdot p \\ \textit{reflection} & [\![\omega_\mathsf{T}]\!] & = & \mathsf{id}_{\nu_\mathsf{T}} \\ \textit{fusion} & [\![p]\!] \cdot h & = & [\![q]\!] \quad \text{if} \quad p \cdot h = \mathsf{T}\ h \cdot q \end{array}$$

Example: definition by coinduction (or corecursion)

$$\begin{array}{ccc} O^\omega & \xrightarrow{\langle \mathsf{hd},\mathsf{tl}\rangle} & O \times O^\omega \\ {\scriptstyle \mathsf{rep}}\uparrow & & \uparrow{\scriptstyle \mathsf{id}\times\mathsf{rep}} \\ O & \xrightarrow{\triangle} & O \times O \end{array}$$

$$\mathsf{rep} = [\![\triangle]\!]$$

- $\triangle$ carries the 'genetic inheritance' of the generating process: it is the *one-step* behaviour specification of rep.

- Unfolding the universal equation we arrive to pointwise definition in the format one would expect: the behaviour is specified under all the *observers*:

$$\begin{array}{ll} & (\mathsf{id} \times \mathsf{rep}) \cdot \triangle = \langle \mathsf{hd}, \mathsf{tl} \rangle \cdot \mathsf{rep} \\ = & \quad \{ \text{ } \triangle \text{ definition} \} \\ & (\mathsf{id} \times \mathsf{rep}) \cdot \langle \mathsf{id}, \mathsf{id} \rangle = \langle \mathsf{hd}, \mathsf{tl} \rangle \cdot \mathsf{rep} \\ = & \quad \{ \text{ } \times \text{ absorption and fusion} \} \\ & \langle \mathsf{id}, \mathsf{rep} \rangle = \langle \mathsf{hd} \cdot \mathsf{rep}, \mathsf{tl} \cdot \mathsf{rep} \rangle \\ = & \quad \{ \text{ structural equality} \} \\ & \mathsf{hd} \cdot \mathsf{rep} = \mathsf{id} \ \ \wedge \ \ \mathsf{tl} \cdot \mathsf{rep} = \mathsf{rep} \\ = & \quad \{ \text{ going pointwise} \} \\ & \mathsf{hd}\ (\mathsf{rep}\ a) = a \ \ \wedge \ \ \mathsf{tl}\ (\mathsf{rep}\ a) = \mathsf{rep}\ a \end{array}$$

- Compare with definitions by *induction* (over ***e.g.***, a data type defined by a signature of *constructors*: the inductive operation is specified by recording the result of its application to each constructor. For example, recall function len over A^*:

$$len\,[] \ = \ 0 \ \ \wedge \ \ len\,h : t \ = \ 1 + len\,t$$

Example: proof by coinduction

Lemma: *Pointwise addition of streams of reals is commutative,* i.e., $\sigma + \tau \ = \ \tau + \sigma$

in pointfree notation:

$$+ \cdot \mathsf{s} \ = \ +$$

where $\mathsf{s} = \langle \pi_2, \pi_1 \rangle$ is the swap natural isomorphism.

Proof

$$\begin{array}{ll} & + \cdot \mathsf{s} \ = \ + \\ \equiv & \quad \{ \text{ definition} \} \\ & [\!(\langle + \cdot (\mathsf{hd} \times \mathsf{hd}), \mathsf{tl} \times \mathsf{tl} \rangle)\!] \cdot \mathsf{s} \ = \ [\!(\langle + \cdot (\mathsf{hd} \times \mathsf{hd}), \mathsf{tl} \times \mathsf{tl} \rangle)\!] \\ \Leftarrow & \quad \{ \text{ coinduction fusion law} \} \\ & \langle + \cdot (\mathsf{hd} \times \mathsf{hd}), \mathsf{tl} \times \mathsf{tl} \rangle \cdot \mathsf{s} \ = \ (\mathsf{id} \times \mathsf{s}) \cdot \langle + \cdot (\mathsf{hd} \times \mathsf{hd}), \mathsf{tl} \times \mathsf{tl} \rangle \\ \equiv & \quad \{ \text{ } \times \text{-fusion and absorption laws} \} \\ & \langle + \cdot (\mathsf{hd} \times \mathsf{hd}) \cdot \mathsf{s}, (\mathsf{tl} \times \mathsf{tl}) \cdot \mathsf{s} \rangle \ = \ \langle + \cdot (\mathsf{hd} \times \mathsf{hd}), \mathsf{s} \cdot (\mathsf{tl} \times \mathsf{tl}) \rangle \\ \equiv & \quad \{ \text{ s is natural} \} \\ & \langle + \cdot \mathsf{s} \cdot (\mathsf{hd} \times \mathsf{hd}), \mathsf{s} \cdot (\mathsf{tl} \times \mathsf{tl}) \rangle \ = \ \langle + \cdot (\mathsf{hd} \times \mathsf{hd}), \mathsf{s} \cdot (\mathsf{tl} \times \mathsf{tl}) \rangle \\ \equiv & \quad \{ \text{ arithmetic adition is commutative (i.e., } + \cdot \mathsf{s} = +) \} \\ & \langle + \cdot (\mathsf{hd} \times \mathsf{hd}), \mathsf{s} \cdot (\mathsf{tl} \times \mathsf{tl}) \rangle \ = \ \langle + \cdot (\mathsf{hd} \times \mathsf{hd}), \mathsf{s} \cdot (\mathsf{tl} \times \mathsf{tl}) \rangle \end{array}$$

- Intuitively, to reason about circular definitions over infinite structures, our attention shifts from argument's structural shrinking to the pro- gressive construction of the result which becomes richer in in- formational contents.

- Traditionally, coinductive proofs are made by *bisimulation* (see, *e.g.*, in process algebra references (*e.g.*, [30]) or basic references in coalgebra theory (*e.g.*, [32, 34]). In the lectures we have presented an alternative *calculational* proof style, in the spirit of the so-called Bird-Meertens formalism [18], which is more generic and closer to the (functional) programming practice. This style explores, in a direct way, the *universal property* associated to final coalgebras.

Applications

To *Final* Coalgebras

Streams Lec 2 slides **and Processes** Lec 3 slides

Introduction to a reconstruction of classical process algebra design based on the *calculational* approach to coinduction explained before. The idea is to apply coinductive reasoning principles and calculational style to the design of process calculi, relying on the representation of processes as inhabitants of final coalgebras for suitable Set functors. This is claimed to provide

- Generic design principles. In particular, structural aspects of processes are clearly separated from the *interaction structure* which defines the synchronisation discipline.
- Generic proofs, in a calculational (basically equational and pointfree) style. Explicit construction of *bisimulations*, as in *e.g.* [30], is avoided.
- Main references: [5, 31].
- Interpreters for process calculi become very easy to define in a functional programming language (see [10]).

To *Arbitrary* Coalgebras

Components Lec 4 slides

This lecture was almost entirely skipped. The case study, however, is interesting as an example of applying coalgebraic techniques to a domain which is not *final*: Software components are, actually, defined as *generalized Mealy machines* and encoded as *seeded* concrete coalgebras for functor

$$\mathsf{T} \;=\; \mathsf{B}(\mathsf{Id} \times O)^I$$

where B is an arbitrary *strong monad* (for more on *monads* see Note N4), expressing a *behavioural model*. For example,

- *Partiality*: $\mathsf{B} = \mathsf{Id} + \mathbf{1}$
- *Non determinism*: $\mathsf{B} = \mathcal{P}$
- *Ordered non determinism*: $\mathsf{B} = \mathsf{Id}^*$
- *Monoidal labelling*: $\mathsf{B} = \mathsf{Id} \times M$, with M a monoid.
- *'Metric' non determinism*: $\mathsf{B} = \mathsf{Bag}_M$ based on $\langle M, \oplus, \otimes \rangle$, where $\otimes$ distributes over $\oplus$, both defining Abelian monoids over M.

This also leads to a *bicategorial* structure (in short a bicategory is a category equipped with a categorial structure in its sets of arrows, see Note N5 for details).

Main references

- On the component model: mainly [11] (but also [4, 19])
- On the component calculus: [7]
- On the component refinement (not covered in the slides): [28, 8, 29, 12]
- On the component *prototyping* in a VDM-like language: report to be released in November 2006 (joint work with Joost Visser and Jacome Cunha)

Composition *vs* Coordination

This are new research results on *coordination models*, resorting to relational/coalgebraic semantics.

- Publications so far: [13, 15, 14]
- Recent application to configurations of web-services were presented last month at FOCLASA'06 (paper available on Elsevier ENTCS in November)

Extra tutorial notes (draft)

N1 — A brief introduction to categories
N2 — Algebraic and coalgebraic structures
N3 — Coalgebraic Structures in Program Construction
N4 — Monads (from a mathematical perspective; for the popular functional programming point of view see, *e.g.*, [17] or browse `www.haskell.org`)
N5 — A brief introduction to bicategories

References

[1] J. Adamek, H. Herrlich, and G. E. Strecker. *Abstract and Concrete Categories*. John Wiley & Sons, Inc (revised electronic edition in 2004), 1990.

[2] R. Backhouse. Mathematics and programming. a revolution in the art of effective reasoning. Inaugural Lecture, School of Computer Science and IT, University of Nottingham, 2001.

[3] R. C. Backhouse, P. Jansson, J. Jeuring, and L. Meertens. Generic programming: An introduction. In S. D. Swierstra, P. R. Henriques, and J. N. Oliveira, editors, *Third International Summer School on Advanced Functional Programming, Braga*, pages 28–115. Springer Lect. Notes Comp. Sci. (1608), September 1998.

[4] L. S. Barbosa. Components as processes: An exercise in coalgebraic modeling. In S. F. Smith and C. L. Talcott, editors, *FMOODS'2000 - Formal Methods for Open Object-Oriented Distributed Systems*, pages 397–417. Kluwer Academic Publishers, September 2000.

[5] L. S. Barbosa. Process calculi *à la* Bird-Meertens. In *CMCS'01*, volume 44.4, pages 47–66, Genova, April 2001. Elect. Notes in Theor. Comp. Sci., Elsevier.

[6] L. S. Barbosa. Coalgebraic structures in program construction. In *Proc. of 6th Brazilian Symposium on Programming Languages (invited tutorial)*, PUC, Rio de Janeiro, June 2002.

[7] L. S. Barbosa. Towards a Calculus of State-based Software Components. *Journal of Universal Computer Science*, 9(8):891–909, August 2003.

[8] L. S. Barbosa. A perspective on component refinement. In F. S. de Boer, M. Bonsangue, S. Graf, and W.-P. de Roever, editors, *FMCO'04, Third International Symposium on Formal Methods for Components and Objects – Revised Lectures*, pages 23–48. Springer Lect. Notes Comp. Sci. (3657), 2005.

[9] L. S. Barbosa and M. H. Martinho. Modelling is for reasoning: A challenge to software engineering education. In editor Chris Haines, editor, *Proc. of ICTMA 12 (International Conference on the Teaching of Mathematical Modelling and Applications), Cass Business School, London*, July 2005.

[10] L. S. Barbosa and J. N. Oliveira. Coinductive interpreters for process calculi. In *Proc. of FLOPS'02*, pages 183–197, Aizu, Japan, September 2002. Springer Lect. Notes Comp. Sci. (2441).

[11] L. S. Barbosa and J. N. Oliveira. State-based components made generic. In H. Peter Gumm, editor, *CMCS'03, Elect. Notes in Theor. Comp. Sci.*, volume 82.1. Elsevier, 2003.

[12] L. S. Barbosa and J. N. Oliveira. Transposing partial components: an exercise on coalgebraic refinement. *Theor. Comp. Sci.*, (in print), 2006.

[13] M. A. Barbosa and L. S. Barbosa. A relational model for component interconnection. 10(7):808–823, 2004.

[14] M. A. Barbosa and L. S. Barbosa. An orchestrator for dynamic interconnection of software components. In *Proc. 2nd International Workshop on Methods and Tools for Coordinating Concurrent, Distributed and Mobile Systems (MTCoord'06)*, Bologna, Italy, June 2006. Elsevier.

[15] M.A. Barbosa and L.S. Barbosa. Specifying software connectors. In K. Araki and Z. Liu, editors, *Proc. First International Colloquim on Theoretical Aspects of Computing (ICTAC'04), Guiyang, China*, pages 53–68. Springer Lect. Notes Comp. Sci. (3407), 2004.

[16] M. Barr and C. Wells. *Category Theory for Computer Scientists*. Series in Computer Science. Prentice-Hall International, 1990.

[17] R. Bird. *Functional Programming Using Haskell*. Series in Computer Science. Prentice-Hall International, 1998.

[18] R. Bird and O. Moor. *The Algebra of Programming*. Series in Computer Science. Prentice-Hall International, 1997.

[19] A. Cruz, L. Barbosa, and J. Oliveira. From algebras to objects: Generation and composition. *Journal of Universal Computer Science*, 11(10):1580–1612, 2005.

[20] J. Fitzgerald, P. G. Larsen, P. Mukherjee, N. Plat, and M. Verhoef. *Validated Designs for Object-oriented Systems*. Springer Verlag, 2005.

[21] B. Jacobs and J. Rutten. A tutorial on (co)algebras and (co)induction. *EATCS Bulletin*, 62:222–159, 1997.

[22] Bart Jacobs. Objects and classes, co-algebraically. In B. Freitag and C.B. Jones, C. Lengauer and H.-J. Schek, editor, *Object-Orientation with Parallelism and Persistence*, pages 83–103. Kluwer, 1996.

[23] Bart Jacobs. Exercises in coalgebraic specification. In R. Backhouse, R. Crole, and J. Gibbons, editors, *Algebraic and Coalgebraic Methods in the Mathematics of Program Construction*, pages 237–280. Springer Lect. Notes Comp. Sci. (2297), 2002.

[24] Cliff B. Jones. *Software Development — a Rigorous Approach*. Series in Computer Science. Prentice-Hall International, 1980.

[25] A. Kurz. Coalgebras and modal logic. Technical report, Lecture Notes for ESSLLII'2001, Helsinki, 2001.

[26] F. W. Lawvere and S. H. Schanuel. *Conceptual Mathematics*. Cambridge University Press, 1997.

[27] S. Mac Lane. *Categories for the Working Mathematician*. Springer Verlag, 1971.

[28] Sun Meng and L. S. Barbosa. On refinement of generic software components. In C. Rettray, S. Maharaj, and C. Shankland, editors, *10th Int. Conf. Algebraic Methods and Software Technology (AMAST)*, pages 506–520, Stirling, 2004. Springer Lect. Notes Comp. Sci. (3116). Best Student Co-authored Paper Award.

[29] Sun Meng and L. S. Barbosa. Components as coalgebras: The refinement dimension. *Theor. Comp. Sci.*, 351:276–294, 2005.

[30] R. Milner. *Communication and Concurrency*. Series in Computer Science. Prentice-Hall International, 1989.

[31] P. R. Ribeiro, M. A. Barbosa, and L. S. Barbosa. Generic process algebra: A programming challenge. *Journal of Universal Computer Science*, 12(7):922–937, 2006.

[32] J. Rutten. Universal coalgebra: A theory of systems. *Theor. Comp. Sci.*, 249(1):3–80, 2000. (Revised version of CWI Techn. Rep. CS-R9652, 1996).

[33] J. Rutten. Elements of stream calculus (an extensive exercise in coinduction). Technical report, CWI, Amsterdam, 2001.

[34] J. Rutten. A coinductive calculus of streams. *Math. Struct. in Comp. Sci.*, 15:93–147, 2005.

[35] R. F. C. Walters. *Categories and Computer Science*, volume 28 of *Cambridge Computer Science Texts*. Cambridge University Press, 1991.