CALCO Young Researchers Workshop

CALCO-jnr 2011

29 August 2011

Abstracts for Presentations

edited by

Corina Cîrstea, Monika Seisenberger, and Toby Wilkinson

UNIVERSITY OF Southampton

LONDON MATHEMATICAL SOCIETY

Preface

CALCO brings together researchers and practitioners to exchange new results related to foundational aspects and both traditional and emerging uses of algebras and coalgebras in computer science.

This is a high-level, bi-annual conference formed by joining the forces and reputations of CMCS (the International Workshop on Coalgebraic Methods in Computer Science), and WADT (the Workshop on Algebraic Development Techniques). Previous very successful CALCO conferences were held 2005 in Swansea, Wales, 2007 in Bergen, Norway, and 2009 in Udine, Italy. This fourth event takes place in Winchester, UK.

The CALCO Young Researchers Workshop, CALCO-jnr, is a CALCO satellite event dedicated to presentations by PhD students and by those who completed their doctoral studies within the past few years. The workshop is open to all - many CALCO conference participants attend the CALCO-jnr workshop (and vice versa).

CALCO-jnr presentations have been selected on the basis of submitted 2-page abstracts, by the CALCO-jnr PC. This booklet contains the abstracts of the accepted contributions. After the workshop, the author(s) of each presentation will be invited to submit a full 10-15 page paper on the same topic. They will also be asked to write (anonymous) reviews of papers submitted by other authors on related topics. Additional reviewing and the final selection of papers will be carried out by the programme committee. The volume of selected papers from the workshop will be published as a technical report, available online. Authors will retain copyright, and are also encouraged to disseminate the results reported at CALCO-jnr by subsequent publication elsewhere.

We would like to thank the CALCO 2011 local organisers for their great efforts to make this event successful. We are grateful to the London Mathematical Society, the British Logic Colloquium and the University of Southampton for their financial support. Special thanks go to John Power and Magne Haveraaen for their splendid work on the programme committee and their continuous support of CALCO-jnr.

August 2011

Corina Cîrstea
Monika Seisenberger
Toby Wilkinson

Table of Contents

A Category Theoretic View of Nondeterministic Recursive Program Schemes

Daniel Schwencke

Technische Universität Braunschweig,
Institut für Theoretische Informatik,
Mühlenpfordtstraße 22–23, D-38106 Braunschweig, Germany
schwencke@iti.cs.tu-bs.de
http://www.tu-braunschweig.de/iti

1 Introduction

Recursive program schemes serve as a tool for giving semantics e. g. to programming languages. Originally investigated in the 1970's, they gained renewed interest in the 2000's when a considerably generalizing category theoretic approach was found [2, 4].

However, this approach does not cover the nondeterministic version of recursive program schemes which appears in classical work as well, see e. g. [1]. Our research aims to fill this gap: we give a category theoretic definition of a nondeterministic recursive program scheme and obtain as our main result an (uninterpreted) semantics for such schemes. Our definitions and results capture and generalize those from loc. cit. and relate to the ones from [4].

2 A Category Theoretic Notion

Recursive program schemes define new function symbols by recursive equations using given function symbols. The corresponding signatures give rise to polynomial functors V (new symbols) and H (given symbols) on the category Set of sets and functions. The sets $F^{H+V}X$ of finite terms over variables from X built from given and new symbols carry the free algebras $\phi_X^{H+V} : (H+V)F^{H+V}X \to F^{H+V}X$ on X. Moreover, these free algebras form a natural transformation ϕ^{H+V} and give rise to the free monad F^{H+V}. Similarly, the sets T^HX of infinite terms over variables from X built from given symbols carry the free completely iterative algebras $\tau^H : HT^HX \to T^HX$ on X which give rise to the free completely iterative monad T^H with universal arrow $\kappa^H : H \to T^H$, see [3].

In the case of nondeterministic schemes (from now on abbreviated NDRPS) nondeterministic choice is allowed on the right-hand sides of the recursive equations. We model this by the nonempty powerset monad $(\mathcal{P}^+, \eta^+, \mu^+)$. Additionally, there is a canonical construction making also $\mathcal{P}^+T^H$ into a monad. This is important in the following definition where $\alpha^\#$ denotes the unique monad morphism extending α and where inl denotes the left coproduct injection.

Definition 1. *Let H and V be a polynomial* Set*-endofunctors. A* nondeterministic recursive program scheme *is a natural transformation $e : V \to \mathcal{P}^+F^{H+V}$. It is called* guarded *if it factors*

$$e \equiv (V \xrightarrow{e'} \mathcal{P}^+HF^{H+V} \xrightarrow{\mathcal{P}^+\mathsf{in}\mathsf{l}F^{H+V}} \mathcal{P}^+(H+V)F^{H+V} \xrightarrow{\mathcal{P}^+\phi^{H+V}} \mathcal{P}^+F^{H+V}) .$$

An uninterpreted solution *of e is a natural transformation $e^\dagger : V \to \mathcal{P}^+T^H$ such that*

$$e^\dagger = (V \xrightarrow{e} \mathcal{P}^+F^{H+V} \xrightarrow{\mathcal{P}^+[\eta^+T^H\cdot\kappa^H, e^\dagger]^\#} \mathcal{P}^+\mathcal{P}^+T^H \xrightarrow{\mu^+T^H} \mathcal{P}^+T^H) .$$

Every classical NDRPS as defined in [1] can be translated into a NDRPS in the sense of Definition 1; and Definition 1 collapses to a special case of the existing category theoretic notion of a (deterministic) recursive program scheme from [4] if $(\mathcal{P}^+, \eta^+, \mu^+)$ is replaced by the identity monad $(\mathrm{Id}, \mathrm{id}, \mathrm{id})$.

3 A Semantics for NDRPS's

An (uninterpreted) semantics for NDRPS's is given by attaching (uninterpreted) solutions to them. Thus our following main result provides an uninterpreted semantics for guarded NDRPS's.

Theorem 2. *Every guarded NDRPS has a canonical greatest uninterpreted solution.*

The proof is rather long and technical; it is based on the fact that $\eta^+(HT^H + \mathrm{Id}) \cdot [\tau^H, \eta^H]^{-1} : T^H \to \mathcal{P}^+T^H$ is a weakly final coalgebra for some functor on some Kleisli category. But it is then not difficult to complete this proof in order to see that uninterpreted solutions of a NDRPS can only differ in infinite terms which are suprema of ω-chains of terms. In other words: uninterpreted solutions of guarded NDRPS's even are nearly unique.

4 Future Work

Due to the category theoretic approach we see several possibilities for future work: a generalization from polynomial functors to the wider class of analytic functors and from finite to infinite terms, the treatment of composite recursive program schemes (replacing the monad $\mathcal{P}^+$ by the monad $(-)^E$) and the investigation of interpreted solutions of NDRPS's.

References

1. Arnold, A., Nivat, M.: Formal computations of non deterministic recursive program schemes. Math. Systems Theory 13, 219–236 (1980)
2. Ghani, N., Lüth, C., Marchi, F.D.: Solving algebraic equations using colagebra. Theor. Inform. Appl. 37, 301–314 (2003)
3. Milius, S.: Completely iterative algebras and completely iterative monads. Inform. and Comput. 196, 1–41 (2005)
4. Milius, S., Moss, L.S.: The category theoretic solution of recursive program schemes. Theoret. Comput. Sci. 366, 3–59 (2006)

Coalgebraic Semantics of Recursive Computation on Circular Data Structures

Baltasar Trancón y Widemann

University of Bayreuth, Germany

This is a summary of the theoretical content of the author's PhD thesis, completed in 2007 at the Technical University of Berlin, Germany. Preliminary results [1–3] have been published in the functional programming community, meeting with scepticism regarding the use of coalgebra. The complete thesis [4] is available in German only. The purpose of this article is to bring the work to the attention of the (co)algebra community.

Nested data structures, such as the algebraic datatypes of functional programming (FP), are usually implemented as directed graphs of cells and pointers in memory. On one hand, the FP principle of referential transparency dictates that the actual numerical values of pointers and identities of cells be hidden from the user, revealing only the content of the cells pointed to. This allows, for instance, to share a common subgraph rather than have multiple copies. On the other hand, it is natural for pointer graphs to contain cycles. In a transparent setting, these are indistinguishable from infinite periodic data. Here, an evaluation strategy for FP in the presence of cycles is developed. By replacing the traditional algebraic semantics of data structures with a finitary coalgebraic one, the issue of actually infinite data is evaded. Consequently, some important decision problems remain tractable, as opposed to settings with non-strict semantics, for instance in the lazy FP language Haskell.

The key idea is to specify cell layout by a signature functor, and to regard each state of memory as a coalgebra of that functor, with address space as the carrier and pointer dereferencing as the operation. Referential transparency is established by presenting only the final coalgebra semantics of a memory state to the user. Nevertheless, the actual addresses are available to the runtime system, allowing for a variety of implicit techniques for cycle detection and handling. The main theoretical contribution of this work is to relate those techniques to rigorous semantic concepts.

The proposed cycle detection technique is based on a traditional eager evaluation model, where pending function incarnations are accumulated as frames on a call stack. It is assumed that functional computations traverse the data graph by recursion, creating a function incarnation for each reached cell. A circular computation is then indicated by the presence of two nested frames of identical content on the call stack, detectable at runtime by stack inspection. Of course, once a cycle is detected, it needs to be handled in an alternative, non-circular way to obtain the correct function result; otherwise an infinite regression, or rather inevitable stack overflow, occurs. By maintaining a cheap marking invariant, the cost of stack inspection can be reduced greatly, at the price of recognizing cycles only up to bisimilarity.

One way to handle cycles is to ensure that each function incarnation produces the root cell of its result first, and delegates the initialization of nested pointers to subsequent recursive incarnations; cf. the optimization technique of tail calls "modulo constructor". In this case, it suffices to share the root cell between the two endpoint incarnations of the cycle by duplicating a pointer. This handling technique can be proven to coincide with primitive corecursion or unfolding of suitably defined step coalgebras, restricted to finite input and output. Unlike in non-strict implementations of corecursion, the cycle structure of the input is retained.

The finiteness of input can be exploited to give an effective implementation of recursive search problems, which cannot be handled in non-strict environments where cycles are unfolded lazily ad infinitum. A detected cycle in a search problem indicates a redundant search that need not be performed, due to the idempotency of logical operations. It can immediately abort with the default value true/false for universal/existential search, resp., without affecting the result. The default values of subproblems can be chosen independently under mild consistency conditions. This technique can be proven to coincide with the choice of an instance from the lattice of fixpoints, due to Tarski's theorem, of a monotonic deduction operator.

The above techniques have been applied to a variety of areas, yielding nontrivial and outright surprising extensions of well-known algorithms from cycle-free to circular data: The first application concerns positional number systems, where all elementary arithmetic and relational operations can be lifted from finite precision to exact rationals. The second application concerns list structures, where many basic operations such as insertion, deletion, comparison, mapping, filtering and even sorting can be lifted to circular lists. Last but not least, the third application concerns dynamic structural subtyping on mutually recursive coalgebraic datatypes, realized in analogy to the "vtable" approach to object-oriented method dispatch, where dynamic cast between recursive types is an inherently and nontrivially circular computation.

References

1. B. Trancón y Widemann. Stacking cycles: Functional transformation of circular data. In T. Arts and R. Peña, editors, *Implementation of Functional Languages (IFL 2002), Revised Selected Papers*, volume 2670 of *LNCS*, pages 150–164. Springer, 2003.
2. B. Trancón y Widemann. Advanced strict corecursion. In G. Michaelson and P. Trinder, editors, *Proc. Implementation of Functional Languages (IFL 2003)*. Heriott–Watt University Edinburgh, 2004.
3. B. Trancón y Widemann. V→M: A virtual machine for strict evaluation of (co)recursive functions. In C. Grelck, F. Huch, G. Michaelson, and P. Trinder, editors, *Implementation and Application of Functional Languages (IFL 2004), Revised Selected Papers*, volume 3474 of *LNCS*, pages 90–107. Springer, 2005.
4. B. Trancón y Widemann. *Strikte Verfahren zyklischer Berechnung*. Dissertation, Technische Universität Berlin, 2007.

Internal Models for Coalgebraic Modal Logics

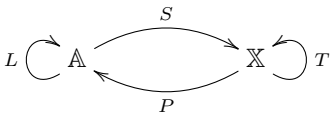
Toby Wilkinson *

University of Southampton, UK
stw08r@ecs.soton.ac.uk

We present ongoing work into the systematic study of the use of dual adjunctions in coalgebraic modal logic. We introduce a category of so called internal models for a modal logic. These are constructed from syntax, and yield a generalised notion of canonical model. Further, expressiveness of a modal logic is seen to relate to coproducts in this category of internal models.

Dual Adjunction Framework

We work in a dual-adjunction framework for coalgebraic modal logic [6, 4, 3].



Briefly this consists of two categories $\mathbb{A}$ and $\mathbb{X}$, and two contravariant functors P and S that form a dual adjunction i.e. there exists a natural isomorphism

$$\Phi\colon \mathbb{A}(-_1, P(-_2)) \Rightarrow \mathbb{X}(-_2, S(-_1))$$

The category $\mathbb{X}$ represents a collection of state spaces, and a collection of generalised transition systems is defined on these state spaces as coalgebras for an endofunctor T. Similarly, the category $\mathbb{A}$ represents a collection of base logics to which modal operators are to be added. These are introduced via an endofunctor L, and the corresponding modal logics are the L-algebras. The semantics of these modal logics is given in two stages. First the dual adjunction gives a semantics for the base logics in terms of the state spaces, and then secondly, a natural transformation

$$\delta\colon LP \Rightarrow PT$$

gives the semantics of the modal operators in terms of the transition structures introduced by T [5, 7].

Models and Internal Models

Let (A, α) denote an L-algebra. Now, not all T-coalgebras are models for the modal logic that (A, α) represents. We can therefore define a category $\mathbf{Mod}(A, \alpha)$ with objects given by pairs

$$((X, \gamma), f\colon X \to S(A))$$

* Research supported by an EPSRC Doctoral Training Account.

where (X, γ) is a T-coalgebra, and f is a theory map, such that $((X, \gamma), f)$ is a model for (A, α). The morphisms are then the T-coalgebra morphisms such that if $g\colon ((X_1, \gamma_1), f_1) \to ((X_2, \gamma_2), f_2)$ then $f_1 = f_2 \circ g$. Note: f is a theory map if its adjunct under the base dual adjunction is an L-algebra homomorphism $(A, \alpha) \to (P(X), P(\gamma) \circ \delta_X)$. These are the valuations in the models of [2]. Theory maps are also related to the twisted coalgebra homomorphisms of [8].

Now given a class M of monomorphisms in $\mathbb{X}$, we define $\mathbf{IntMod}(A, \alpha)$ to be the full subcategory of $\mathbf{Mod}(A, \alpha)$ where the theory maps are in M, and write

$$R\colon \mathbf{IntMod}(A, \alpha) \to \mathbf{Mod}(A, \alpha)$$

for the corresponding inclusion functor.

The intuition is that $S(A)$ is to be thought of as the collection of all possible theories of (A, α), and the objects of $\mathbf{IntMod}(A, \alpha)$, the **internal models**, are those models constructed from subsets of the set of all theories i.e. from the syntax. We parameterise by the class M, as for some categories $\mathbb{X}$, only a subclass of all monomorphisms represents what we would consider "inclusions" or "embeddings". For example in **Top**, M is the class of regular monomorphisms.

IntMod(A, α) and Properties of Coalgebraic Modal Logic

The category $\mathbf{IntMod}(A, \alpha)$ is closely related to many interesting properties of coalgebraic modal logic. For example, sufficient conditions for a dual adjunction to exist between $\mathbf{Alg}(L)$ and $\mathbf{CoAlg}(T)$, are that for all L-algebras (A, α) the following hold:

1. for all X in $\mathbf{Mod}(A, \alpha)$ there exists a $g\colon X \to R(I)$ for some object I in $\mathbf{IntMod}(A, \alpha)$,
2. $\mathbf{IntMod}(A, \alpha)$ has a final object.

The final internal model of condition 2 can be thought of as analogous to the canonical models of Kripke semantics [1].

By selecting $\mathbb{X}$ so that the objects are sets with some additional structure, and the morphisms have underlying functions, we can talk about the individual states of a model. Then given two models X_1, X_2 in $\mathbf{Mod}(A, \alpha)$, and $x_1 \in X_1$, $x_2 \in X_2$, we say x_1 and x_2 are **behaviourally equivalent** if there exists in $\mathbf{Mod}(A, \alpha)$ a cospan

$$X_1 \xrightarrow{f_1} X_3 \xleftarrow{f_2} X_2$$

such that $f_1(x_1) = f_2(x_2)$.

The forgetful functor from $\mathbf{Mod}(A, \alpha)$ to $\mathbf{CoAlg}(T)$ yields the usual definition of behavioural equivalence as a cospan in $\mathbf{CoAlg}(T)$, and the forgetful functor to $\mathbb{X}$ yields a condition that the theory maps are compatible c.f. the definition of bisimulation in [1]. Note: [2] has a similar definition, but requires f_1, f_2 to be surjective.

We say (A, α) is **expressive**, if for all models in $\mathbf{Mod}(A, \alpha)$, states have the same theories if and only if they are behaviourally equivalent [4, 3]. Now

if we choose M to be some subclass of the class of monos in $\mathbb{X}$ with injective underlying functions, then sufficient conditions for expressiveness are:

1. same as condition 1 above,
2. for every pair I_1, I_2 in $\mathbf{IntMod}(A, \alpha)$ there is a cospan $I_1 \to I_3 \leftarrow I_2$ in $\mathbf{IntMod}(A, \alpha)$.

Suppose now that a class E of morphisms in $\mathbb{X}$ exists such that $\mathbb{X}$ has an (E, M)-factorisation system. We also know that for every choice of $\delta\colon LP \Rightarrow PT$ the base dual adjunction gives a natural transformation $\delta^*\colon TS \Rightarrow SL$. So if

$$m \in M \Rightarrow \delta^*_A \circ T(m) \in M$$

then the following hold

1. condition 1 above,
2. the forgetful functor $U\colon \mathbf{IntMod}(A, \alpha) \to \mathbb{X}$ detects small colimits.

Thus if in addition $\mathbb{X}$ has binary coproducts, then (A, α) is expressive. Alternatively, if $\mathbb{X}$ is M-wellpowered and has small coproducts, then $\mathbf{IntMod}(A, \alpha)$ has a final object, and if this is true for all (A, α), there is a dual adjunction between $\mathbf{Alg}(L)$ and $\mathbf{CoAlg}(T)$.

Future Work

The category $\mathbf{IntMod}(A, \alpha)$ is not yet fully understood, and indeed, an obvious question is when the sufficient conditions for the above results are actually necessary. It is also anticipated, that like canonical models in Kripke semantics, internal models may have something to say about completeness.

Acknowledgements

The author would like to thank Corina Cîrstea for the many interesting discussions and her invaluable guidance.

References

1. Blackburn, P., de Rijke, M., Venema, Y.: Modal Logic. Cambridge University Press (2001)
2. Doberkat, E.-E.: Stochastic Coalgebraic Logic. Springer Berlin Heidelberg (2009)
3. Jacobs, B., Sokolova, A.: Exemplaric Expressivity of Modal Logics. J Logic Computation (2009)
4. Klin, B.: Coalgebraic modal logic beyond sets. Electronic Notes in Theoretical Computer Science 173 (2007) 177–201
5. Kupke, C., Kurz, A., Pattinson, D.: Algebraic semantics for coalgebraic logics. Electronic Notes in Theoretical Computer Science 106 (2004) 219–241
6. Kupke, C., Kurz, A., Venema, Y.: Stone coalgebras. Theoretical Computer Science 327(1–2) (2004) 109–134
7. Kupke, C., Kurz, A., Pattinson, D.: Ultrafilter extensions for coalgebras. Lecture Notes in Computer Science Vol.3629, (2005) 263–277
8. Pavlovic, D., Mislove, M., Worrell J.: Testing Semantics: Connecting Processes and Process Logic. Lecture Notes in Computer Science Vol.4019, (2006) 308–322

A nondeterministic probabilistic monad with nondeterminism for coalgebraic trace semantics

Tetsuya Sato

Research Institute for Mathematical Sciences, Kyoto University, Kyoto 606-8502, Japan E-mail : satoutet@kurims.kyoto-u.ac.jp

Introduction. In the study of coalgebraic trace semantics of Hasuo et al. [1, 2], finite trace semantics for the powerset monad $\mathcal{P}$ as well as the (sub)distribution monad $\mathcal{D}$ can be derived from an initial algebra; the initial F-algebra $\alpha\colon FA \to A$ in **Set** gives rise to the final F-coalgebra in the Kleisli category of each monad. In this work we construct a probabilistic monad with nondeterminism $\mathcal{P}_+\mathcal{I}D$ based on indexed valuations of Varacca and Winskel [5]. We show that *weakly final* coalgebras in the Kleisli category of this monad captures a finite trace semantics for a Segala automaton [4].

Indexed valuations. We recall the notion of indexed valuations following [5]. Let X be a set. A *discrete indexed valuation* on X is a pair (Ind, v) of two functions $\mathrm{Ind}\colon I \to X$ and $v\colon I \to [0,\infty]$, for some set I. The equivalence relation $\sim$ on two indexed valuations is defined as follows: given two indexed valuations on X, (Ind, v) and (Ind', v'), $(\mathrm{Ind}, v) \sim (\mathrm{Ind}', v')$ if and only if there is a bijection $h\colon \mathrm{Supp}(v) \to \mathrm{Supp}(v')$ such that $\mathrm{Ind}(i) = \mathrm{Ind}'(h(i))$ and $v(i) = v'(h(i))$ for all $i \in \mathrm{Supp}(v)$. Here, $\mathrm{Supp}(v) = \{i \in I | v(i) \neq 0\}$.

For a set X, the set of all indexed valuations on X is denoted by $\mathcal{I}V X$. The mapping $X \mapsto \mathcal{I}V X$ is functorial, and can be extended to a monad on **Set**. Let $\mathcal{P}_+$ be a nonempty powerset monad. The composition $\mathcal{P}_+\mathcal{I}V$ also forms a monad on **Set**, via a distributive law between $\mathcal{P}_+$ and $\mathcal{I}V$.

Our Results. Similarly, for any set X, we write $\mathcal{I}DX$ for the set of all indexed valuations (Ind, v) on X such that $\sum_i v(i) \leq 1$. $\mathcal{I}D$ extends to a monad $\mathcal{I}D$ on **Set**, and the composition $\mathcal{P}_+\mathcal{I}D$ also forms a monad on **Set**, again via a distributive law.

Proposition 1. *For any set X, $\mathcal{I}DX$ has an ω-complete partial order with the least element $\underline{0}$, which is the equivalence class of (Ind, v) such that $v = 0$.*

Proposition 2. *Monad $\mathcal{P}_+\mathcal{I}D$ is a commutative strong monad on* **Set**.

Proposition 3. *The Kleisli category $\mathbf{Set}_{\mathcal{P}_+\mathcal{I}D}$ is enriched with the trivial inclusion order.*

Let F be an endofunctor $1 + (A \times id)$ on **Set** (A is an arbitrary set). A polynomial endofunctor on **Set** E has a lifting $\overline{E}$ on the Kleisli category $\mathbf{Set}_T$ of a

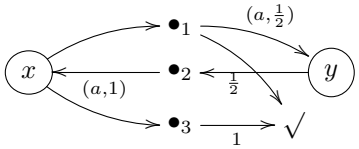
commutative strong monad T. See also [2]. Thus, by proposition 2 the functor F has a lifting $\overline{F}$ in $\mathbf{Set}_{\mathcal{P}_+ ID}$. By simple translation a Segala automaton can be captured by an $\overline{F}$-coalgebra in $\mathbf{Set}_{\mathcal{P}_+ ID}$.

Theorem 1. *1. Let $\alpha\colon FA^* \to A^*$ be an initial F-algebra. Then the $\overline{F}$-coalgebra $\eta \circ \alpha^{-1}\colon A^* \to \overline{F}A^*$ is weakly final, where η is the unit of the monad.*

2. Consider a Segala automata ($\overline{F}$-coalgebra) $c\colon X \to \overline{F}X$. The finite trace semantics of c is captured by the maximum coalgebra morphism $c \to \eta \circ \alpha^{-1}$ with respect to the inclusion order of proposition 3.

The key of the proof of this theorem is proposition 1. Thus we can construct the trace semantics as the pointwise limit of functions.

Example. Consider the functor $1+(A\times id)$. The initial F-algebra $[nil; cons]\colon FA^* \to A^*$ consists of the finite lists over A. The following is an example of a coalgebra $c\colon X \to \mathcal{P}_+ IDFX$.



By Theorem 1 we obtain a coalgebra morphism $\mathbf{Trace}\colon c \to \eta \circ \alpha$ via weak finality and taking the maximum. The trace semantics of the state x is

$$\mathbf{Trace}(x) = \{(\langle\rangle, 1), (\langle\rangle, \frac{1}{2}) \oplus (a, \frac{1}{2}), (\langle\rangle, \frac{1}{2}) \oplus (a, \frac{1}{4}) \oplus (aa, \frac{1}{4}), \ldots\}$$

Note that we express indexed valuations as $(x_0, p_0) \oplus (x_1, p_1) \oplus \cdots \oplus (a_k, p_k) \oplus \cdots$, where $\sum_{k=0}^{\infty} p_k \leq 1$. For instance $(\langle\rangle, \frac{1}{2}) \oplus (a, \frac{1}{2})$ is equivalent to the indexed valuation $\mathrm{Ind}\colon \mathbf{2} \to A^*, v\colon \mathbf{2} \to [0, \infty]$ defined by $\mathrm{Ind}(0) = \langle\rangle$, $\mathrm{Ind}(1) = a$, $v(0) = \frac{1}{2}$ and $v(1) = \frac{1}{2}$.

Related work The comparison between Jacob's work using convex subsets of distributions [3] and ours is an important future work.

References

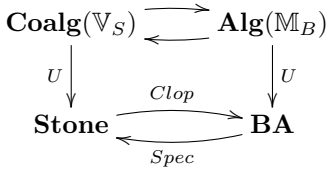
1. Ichiro Hasuo. **Tracing Anonymity with Coalgebras.** *PhD thesis, Radboud University Nijmegen. Defended on March 10, 2008.*
2. Ichiro Hasuo, Bart Jacobs and Ana Sokolova. **Generic Trace Semantics via Coinduction.** *Logical Methods in Computer Science,* 3(4:11), 2007.
3. Bart Jacobs.**Coalgebraic trace semantics for combined possibilitistic and probabilistic systems.***Electronic Notes in Theoretical Computer Science*, 203(5):131-152, 2008.
4. Roberto Segala. **A compositional trace-based semantics for probabilistic automata.** *In International Conference on Concurrency Theory (CONCUR '95),* pages 234-248. Springer-Verlag, 1995
5. Daniele Varacca and Glynn Winskel. **Distributing Probability over Nondeterminism.** *Mathematical Structures in Computer Science* 16(1) (2006)

A final Vietoris coalgebra beyond compact spaces and a generalized Jónsson-Tarski duality

Liang-Ting Chen* and Achim Jung

School of Computer Science, University of Birmingham
{L.Chen,A.Jung}@cs.bham.ac.uk

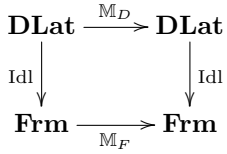
In a coalgebraic perspective, Kripke frames and models can be viewed as $\mathcal{P}$-coalgebras and $(\mathcal{P} \times \mathcal{P}\Phi)$-coalgebras respectively where Φ is the set of atoms. Furthermore, descriptive Kripke frames are Vietoris coalgebras over Stone spaces as shown in [3], and the classical Jónsson-Tarski duality is a duality between the category $\mathbf{Alg}(\mathbb{M}_B)$ of $\mathbb{M}_B$-algebras and the category $\mathbf{Coalg}(\mathbb{V}_S)$ of Vietoris coalgebras, i.e.



where $\mathbb{V}_S$ is the Vietoris construction over Stone spaces and $\mathbb{M}_B$ is the modal algebra construction over Boolean algebras. The duality also holds for positive normal modal logic, i.e. the duality between modal algebras over distributive lattice denoted by $\mathbb{M}_D$ and the Vietoris topology over Priestley spaces (isomorphic to coherent spaces [1]) as shown in [4].

Following these dualities, we generalise it to the duality between $\mathbf{Coalg}(\mathbb{V})$ and $\mathbf{Alg}(\mathbb{M}_F)$ over stably locally compact spaces (see [1]) where $\mathbb{V}$ is the Vietoris topology construction taking compact lens (intersections of closed sets and open sets) and $\mathbb{M}_F$ is the modal algebra construction over frames. These constructions can be found in [2].

As for the final $\mathbb{V}$-coalgebra, we start from the algebraic view instead of the coalgebraic view and obtain an initial $\mathbb{M}_F$-algebra concretely via the initial sequence. The calculation relies on the coherence preservation of $\mathbb{M}_F$, i.e.



commutes.

By Stone duality and the natural isomorphism between $\mathbb{V}\,\mathrm{Pt}$ and $\mathrm{Pt}\,\mathbb{M}_F$ for stably locally compact spaces, we find the final $\mathbb{V}$-coalgebra from the initial $\mathbb{M}_F$-algebra. Routinely we further generalise results to coherent Vietoris polynomial functors.

* Supported by a Doctoral Training Grant funded by EPSRC and the School.

References

1. P. T. Johnstone, *Stone spaces*. ,Cambridge Studies in Advanced Mathematics, no 3, Cambridge University Press 1982.
2. P. T. Johnstone, Vietoris Locales and Localic Semilattices. *Continuous lattices and their Applications* (R.-E. Hoffmann, ed.), Pure and Applied Mathematics, no.101, Marcel Dekker, 1985, pp. 155–18.
3. C. Kupke and A. Kurz and Y. Venema, Stone coalgeras. *Theoret. Comput. Sci.*, 327, 2004.
4. N. Bezhanishvili and A. Kurz, Free modal algebras: A coalgebraic perspective. In *CALCO 2007*, volume 4624 of *LNCS*, pp. 143-157. Springer-Verlang 2007.

Realizable Institutions

Michal R. Przybylek

Faculty of Mathematics, Informatics and Mechanics University of Warsaw, Poland
{mrp@mimuw.edu.pl}

Language is the most fundamental tool of speech. Logic is the most fundamental tool of reasoning. Therefore, to speak on reasoning, we need a logic over a language. If we mean by "logic" a model-theoretic logic — that is — two sets: a set of formulae and a set of models, together with a relation saying what formulae are true in what models, then perhaps the most successful description for the concept of "logic (varying) over a language" was given by Goguen and Burstall [5–7]. They called such entities *institutions*[1]:

Definition 1 (Institution). *An institution I is a quadruple $\langle \mathbf{Sign}, Sen, Mod, \models \rangle$, where:*

- $\mathbf{Sign}$ *is a category, whose objects are called signatures*
- Sen *is a functor* $\mathbf{Sign} \to \mathbf{Cat}$ *sending signatures to formulae (sentences),*
- Mod *is a functor* $\mathbf{Sign}^{op} \to \mathbf{Cat}$ *sending signatures to models,*
- $\models$ *is a family of "satisfaction" relations* $\models^{\Sigma \in \mathbf{Sign}} \subset \|Mod(\Sigma)\| \times \|Sen(\Sigma)\|$, *where* $\| - \|$ *indicates the class of objects of a given category*

such that for every morphism $\sigma \colon \Sigma \to \Sigma'$ the following condition holds: $M \models^{\Sigma'} Sen(\sigma)(\phi) \Leftrightarrow Mod(\sigma)(M) \models^{\Sigma} \phi$.

The definition roughly says that formulae change *covariantly* with changes of the language, models change *contravariantly* with changes of the language and logical values are compatible with these changes. Institution theory has proven its usefulness in a variety of contexts, including: software specification and formal software development [11], semantics of databases and ontologies [9], cognitive linguistics and cognitive semantics [8], universal algebra [3, 4, 14].

One may summarize institutions in the picture 1, where $BRel$ is the class of all binary relations on sets, and $\models$ has to be compatible with morphisms in $\mathbf{Sign}$. In this paper we shall try to replace picture 1 with picture 2, where p is a module fibration; obtaining, what we shall call, *realizable institution*. Let us elaborate more on this concept.

Definition 2 (Module). *A module [1, 2] Ψ from a category $\mathbb{X}$ to a category $\mathbb{Y}$, denoted by $\Psi \colon \mathbb{X} \nrightarrow \mathbb{Y}$, is a functor $\mathbb{Y}^{op} \times \mathbb{X} \to \mathbf{Set}$. We will write Ψ^* for $\mathbb{X}$ and $\Psi_!$ for $\mathbb{Y}$.*

Definition 3 (Module Morphism). *A morphism H from a module Ψ to a module Φ consists of the following data:*

[1] Actually what we describe is the strongest variant of institutions, sometimes called "**Cat**/**Cat** institutions" — instead of sets of formulae and sets of models there are categories of formulae and categories of models.

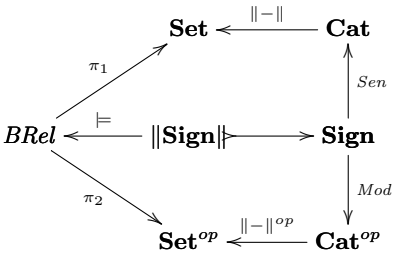


Fig. 1.

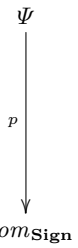


Fig. 2.

- *a functor* $H^*\colon \Psi^* \to \Phi^*$
- *a functor* $H_!\colon \Psi_! \to \Phi_!$
- *a natural transformation* $\hat{H}\colon \Psi \to \Phi \circ ({H_!}^{op} \times H^*)$

Definition 4 (Module Transformation). *A module transformation* τ *from a module morphism* $H\colon \Psi \to \Phi$ *to a module morphism* $K\colon \Psi \to \Phi$ *is a pair of natural transformations* $\tau^*\colon H^* \to K^*$, $\tau_!\colon K_! \to H_!$ *satisfying for all objects* $A \in \Psi^*$, $B \in \Psi_!$ *the following smoothness condition:* $\hat{K}_{B,A} = \Phi({\tau_!}_B, {\tau^*}_A) \circ \hat{H}_{B,A}$.

It is easy to verify that modules, module morphisms and module transformations constitute a 2-category, and that this category has weighted limits (so, comma objects). Since it has comma objects, we can speak of fibrations [12, 13] over modules, which we shall call *profibrations*. If a profibration comes equipped with a cleavage (i.e. with a choice of a cartesian morphism for every morphism from a base module) than we shall call it a *cloven profibration*. Just like ordinary cloven fibrations have external forms in the shape of **Cat**-valued pseudofunctors (i.e. indexed categories), cloven profibrations have their external forms in the shape of *Dist*-valued module pseudomorphisms[2], which we shall call *indexed modules*.

Definition 5 (Indexed module). *A module indexed over a module* Φ *is a module pseudomorphism* $\Theta\colon \Phi \to Dist$ *consisting of the following data:*

- *a pseudofunctor* $\Theta^*\colon \Phi^* \to \mathbf{Cat}^{op}$
- *a pseudofunctor* $\Theta_!\colon \Phi_! \to \mathbf{Cat}$
- *a pseudonatural transformation* $\hat{\Theta}\colon \Phi \to Dist \circ ({\Theta_!}^{op} \times \Theta^*)$

Definition 6 (Realizable institution). *A realizable institution* I *over a category* $\mathbb{B}$ *is a profibration* $I\colon \Psi \to Hom_{\mathbb{B}}$. *An element* $r \in \Psi(Y, X)$ *over a morphism* $\sigma\colon B \to A$ *in* $\mathbb{B}$ *will be denoted by* $Y \models^r_\sigma X$. *We shall write* $Y \models^r X$ *for* $Y \models^r_{id} X$.

Realizable institutions emerged from two observations. Firstly, one may argue that the theory of institutions is about linking the logic of models with the logic

[2] *Dist* stays here for a canonical module over the bicategory of distributors.

of theories — since existence of homomorphisms between particular models has exactly nothing to do with either of the logics, the theory of institution should not be willing to embrace such entities. Instead, it should be willing to embrace the notions of "logic of models" (which says how rich a model is; think for example of sets of the standard models as of models — if $M \subset N$ then M is richer than N) and "logic of proofs" (which says what formulae entail what formulae). In addition, both of the logics should smoothly compose with the satisfaction relation (i.e. if $M \subset N, N \models \phi, \phi \to \psi$ then $M \models \psi$). Secondly, just like standard logicians bother with *particular* proofs between formulae and not merely with their *existence*, the institution theoreticians should bother with *particular* witnesses of the fact that some models force some formulae.

Nonetheless, the primary reason for introducing realizable institutions is to make an intermediate step toward development of a uniform framework suitable for analyzing versatile of model-theoretic logics. The next step is then in enriching the presented results with a cosmos (i.e. complete and cocomplete symmetric monoidal closed category) [10]. We show how this general setting incorporates external (i.e. model-theoretic) with internal (i.e. proof-theoretic) logic.

References

1. J. Bénabou, *Introduction to Bicategories*, Reports of the Midwest Category Seminar, Lecture Notes in Math. 47, Springer (1967)
2. J. Bénabou, *Distributors at Work*, Lecture notes of a course given by Jean Bnabou in June 2000 at TU Darmstadt about Distributors and Generalized Fibrations, `http://www.mathematik.tu-darmstadt.de/~streicher`
3. R. Diaconescu *Institution-Independent Model Theory*, Birkhuser (2008)
4. D. Gaina, A. Popescu *An institution-independent generalisation of Tarski's elementary chain theorem*, Journal of Logic and Computation 16(6) (2006)
5. J. A. Goguen, R. M. Burstall, *Introducing Institutions*, Lecture Notes in Computer Science 164 (1984)
6. J. A. Goguen, R. M. Burstall, *Institutions: Abstract Model Theory for Specification and Programming*, Journal of the Association for Computing Machinery 39 (1992)
7. J. A. Goguen and G. Rosu, *Institution morphisms*, Formal aspects of computing 13, (2002)
8. J. A. Goguen, *Information Integration in Institutions*, `http://cseweb.ucsd.edu/users/goguen/pps/ifi04.pdf` (2004)
9. J. A. Goguen, *Data, Schema, Ontology and Logic Integration*, Logic Journal of IGPL 13(6) (2005)
10. G. M. Kelly, *Basic Concepts of Enriched Category Theory*, London Mathematical Society Lecture Note Series No.64 (1982)
11. D. Sannella, A. Tarlecki, *Foundations of Algebraic Specification and Formal Software Development*, Springer, (hopefully 2011)
12. R. Street, *Fibrations in Bicategories*, Cahiers Topologie et Gomtrie Diffrentielle Catgoriques, 21 (1980)
13. R. Street, *Correction to "Fibrations in Bicategories"*, Cahiers Topologie et Gomtrie Diffrentielle Catgoriques, 28 (1987)
14. A. Tarlecki *Quasi-varieties in abstract algebraic institutions*, Journal of Computer and System Sciences 33(3) (1986)

Modelling and Analysis of Real Time Systems with Logic Programming and Constraints *

Gourinath Banda

National Aerospace Laboratories, Bangalore, India
Email: gourinath@nal.res.in

Embedded systems are increasingly being deployed in a wide variety of applications. Most, if not all, of these applications involve an electronic controller with discrete behaviour controlling a continuously evolving plant. Because of their hybrid behaviour (discrete and continuous) and reactive behaviour, the *formal verification*[29] of embedded systems pose new challenges.

Research efforts undertaken to improve confidence in the correctness of embedded systems have resulted in various modelling and programming languages, verification techniques, frameworks, etc. as surveyed in [23] [15] [27] [20] [8] [28] [24]. Linear Hybrid Automata (LHA)[1] is a language for specifying systems with linear hybrid behaviour. *Abstract interpretation*[9] [10] is a formal theory for approximating the semantics of programming languages. *Model checking*[7] [25] is a technique to verify the reactive behaviour of concurrent systems. Computation Tree Logic (CTL)[12] is a temporal property specification language. *Logic programming*[22] is a general purpose programming language based on predicate logic.

1 Proposed framework

In this recently concluded doctoral work, which is reported in five publications [5] [4] [3] [17] [13] and my doctoral dissertation, the LHA models are verified by encoding them as constraint logic programs. We model the reachable state semantics of an LHA. Such constraint logic program (CLP) encoding an LHA model is first specialised and then a concrete minimal model (or possibly an abstract minimal model) is computed by subjecting the residual program to static analysis. The abstract minimal model is computed by applying the theory of abstract interpretation. This concrete (or abstract) minimal model being a *finite* representation of infinite reachable state space, forms the basis for verifying the LHA model. We consider two techniques to verify the reactive properties specified as CTL formulas: (i) reachability analysis and (ii) model checking. Figure 1 shows the proposed formal verification framework.

* Work undertaken while the author was a doctoral fellow supervised by Professor John Gallagher at Department of Computer Science, Roskilde University. This work was supported by: (i) a phd-fellowship jointly sponsored by the Roskilde University and the EU-IST project *ASAP: Advanced Specialisation and Analysis for Pervasive Systems* and (ii) the Danish Natural Science Research Council project *SAFT: Static Analysis Using Finite Tree Automata*.

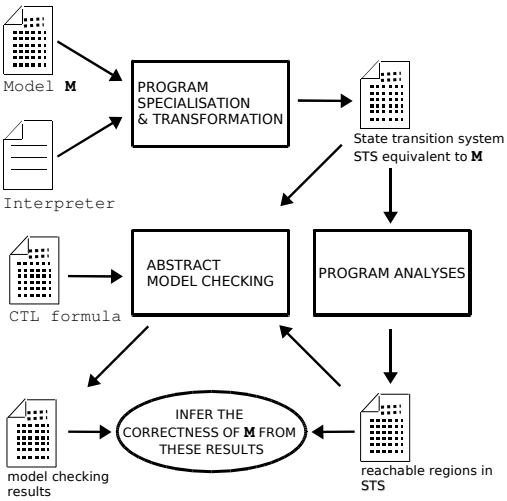


Fig. 1. A framework for verifying embedded systems

A systematic translation of LHA models into constraint logic programs is defined. This is mechanised by a compiler. To facilitate forward and backward reasoning, two different ways to model an LHA are defined[3]. A framework consisting of general purpose constraint logic program tools is presented to accomplish the reachability analysis to verify a class of safety and liveness properties. A tool to compute the concrete minimal model is implemented. The model checking of CTL is defined as a concrete CTL-semantic function. Since model checking of infinite state systems, which LHAs are, does not terminate, we apply the theory of abstract interpretation to model checking that ensures termination at the cost of loss in precision. An abstract CTL-semantic function is constructed as an abstract interpretation of the CTL-semantic function. This abstract CTL-semantic function is implemented using a SMT solver resulting in an abstract model checker (AMC). We consider two abstract domains: (i) the domain of constraints and (ii) the domain of convex polyhedra, for both abstract model checking and abstract minimal model computation. The abstract model checking is presented in [5].

1.1 Framework demonstration

The applicability of the proposed theory (with the implemented tool framework) to verify real time systems has been demonstrated by considering several examples taken from literature. The list of systems verified include: *Train-gate-controller*[2], Task scheduler[2], *Gas burner controller*[26] [19], *Fischer protocol*[21], *Temperature controller*[19] and *Water-level controller*[16]. The results are presented in [5], [4], [3].

2 Novelties

First, a systematic translation scheme from LHA to CLP is defined capturing the reachable and reaching state semantics. We apply general purpose program analyses tools meant for CLP programs to analyse LHA models. Furthermore, AMC, unlike other existing model checking tools (such as Uppaal[6], HyTech[18], *etc.*) that cannot handle nested temporal formulas, can model check any well formed CTL-formula. As we report in [5], our approach makes it possible to verify temporal properties expressed in the modal-μ logic, which is the most expressive temporal logic. On the other hand, unlike other abstract model checking techniques (such as [11] [14]) that mandate constructing an abstract kripke structure or modal transition system, thanks to both our *CTL-semantics function* definition and the framework of abstract interpretation, with AMC it is not necessary to explicitly construct such abstract systems.

References

1. R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theor. Comput. Sci.*, 138(1):3–34, 1995.
2. R. Alur, T. A. Henzinger, and P.-H. Ho. Automatic symbolic verification of embedded systems. *IEEE Trans. Software Eng.*, 22(3):181–201, 1996.
3. G. Banda and J. P. Gallagher. Analysis of linear hybrid systems in clp. In M. Hanus, editor, *LOPSTR*, volume 5438 of *Lecture Notes in Computer Science*, pages 55–70. Springer, 2008.
4. G. Banda and J. P. Gallagher. Constraint-based abstraction of a model checker for infinite state systems. In A. Wolf and U. Geske, editors, *WLP '09: Proc. 23rd Workshop on (Constraint) Logic Programming*, pages 109–124. Universität Potsdam, 2009.
5. G. Banda and J. P. Gallagher. Constraint-based abstract semantics for temporal logic: A direct approach to design and implementation. In E. M. Clarke and A. Voronkov, editors, *LPAR (Dakar)*, volume 6355 of *Lecture Notes in Computer Science*, pages 27–45. Springer, 2010.
6. G. Behrmann, A. David, and K. G. Larsen. A tutorial on Uppaal. In M. Bernardo and F. Corradini, editors, *SFM-RT 2004*, number 3185 in Lecture Notes in Computer Science, pages 200–236. Springer, September 2004.
7. E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In D. Kozen, editor, *Logic of Programs*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71. Springer, 1981.
8. E. M. Clarke and J. M. Wing. Formal methods: State of the art and future directions. *ACM Comput. Surv.*, 28(4):626–643, 1996.
9. P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL*, pages 238–252, 1977.
10. P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *POPL*, pages 269–282, 1979.
11. D. Dams, R. Gerth, and O. Grumberg. Abstract interpretation of reactive systems. *ACM Trans. Program. Lang. Syst.*, 19(2):253–291, 1997.

12. E. A. Emerson and E. M. Clarke. Using branching time temporal logic to synthesize synchronization skeletons. *Sci. Comput. Program.*, 2(3):241–266, 1982.
13. J. P. Gallagher, K. S. Henriksen, and G. Banda. Techniques for scaling up analyses based on pre-interpretations. In M. Gabbrielli and G. Gupta, editors, *ICLP*, volume 3668 of *Lecture Notes in Computer Science*, pages 280–296. Springer, 2005.
14. P. Godefroid, M. Huth, and R. Jagadeesan. Abstraction-based model checking using modal transition systems. In K. G. Larsen and M. Nielsen, editors, *CONCUR 2001*, volume 2154 of *Lecture Notes in Computer Science*, pages 426–440. Springer, 2001.
15. A. Gupta. Formal hardware verification methods: A survey. *Formal Methods in System Design*, 1(2/3):151–238, 1992.
16. N. Halbwachs, Y.-E. Proy, and P. Roumanoff. Verification of real-time systems using linear relation analysis. *Formal Methods in System Design*, 11(2):157–185, 1997.
17. K. S. Henriksen, G. Banda, and J. P. Gallagher. Experiments with a convex polyhedral analysis tool for logic programs. *CoRR*, abs/0712.2737, 2007.
18. T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. HYTECH: A model checker for hybrid systems. In *Computer Aided Verification, 9th International Conference, CAV '97, Haifa, Israel, June 22-25, 1997, Proceedings*, volume 1254 of *Lecture Notes in Computer Science*, pages 460–463. Springer, 1997.
19. T. A. Henzinger and V. Rusu. Reachability verification for hybrid automata. In T. A. Henzinger and S. Sastry, editors, *HSCC*, volume 1386 of *Lecture Notes in Computer Science*, pages 190–204. Springer, 1998.
20. J. C. Kelly and K. Kemp. *Formal methods specification and verification guidebook for software and computer systems*, volume I: Planning and technology insertion. Office of Safety and Mission Assurance, NASA, National Aeronautics and Space Administration, Washington, DC 20546, USA, July 1995.
21. L. Lamport. A fast mutual exclusion algorithm. *ACM Trans. Comput. Syst.*, 5(1):1–11, 1987.
22. J. W. Lloyd. *Foundations of logic programming; (2nd extended ed.)*. Springer-Verlag New York, Inc., New York, NY, USA, 1987.
23. J. S. Ostroff. Formal methods for the specification and design of real-time safety critical systems. *J. Syst. Softw.*, 18(1):33–60, 1992.
24. M. R. Prasad, A. Biere, and A. Gupta. A survey of recent advances in SAT-based formal verification. *STTT*, 7(2):156–173, 2005.
25. J.-P. Queille and J. Sifakis. Specification and verification of concurrent systems in CESAR. In M. Dezani-Ciancaglini and U. Montanari, editors, *Symposium on Programming*, volume 137 of *Lecture Notes in Computer Science*, pages 337–351. Springer, 1982.
26. A. P. Ravn, H. Rischel, and K. M. Hansen. Specifying and verifying requirements of real-time systems. *IEEE Trans. Software Eng.*, 19(1):41–55, 1993.
27. D. Sannella. A survey of formal software development methods. In R. Thayer and A. McGettrick, editors, *Software Engineering: A European Perspective*, pages 281–297. IEEE Computer Society Press, 1993. Originally published as report ECS-LFCS-88-56, Dept. of Computer Science, Univ. of Edinburgh, 1988.
28. F. Wang. Formal verification of timed systems: A survey and perspective. In *Proceedings of the IEEE*, volume 92, pages 1283–1307, Los Alamitos, CA, USA, 2004. IEEE Computer Society.
29. J. M. Wing. A specifier's introduction to formal methods. *Computer*, 23(9):8–23, 1990.

Domain-theoretic Modelling of Functional Programming Languages

Tie Hou and Ulrich Berger

Department of Computer Science, Swansea University, UK
{cshou,u.berger}@swansea.ac.uk

This paper is part of a research project aiming at a semantical foundation for program extraction from proofs [1]. It contributes to a soundness proof for a Language of Realiser (LoR) of proofs involving inductive and coinductive definitions. The natural LoR for inductive and coinductive definitions is a typed lambda calculus with types modelling initial algebras and final coalgebras, and terms modelling structural recursion and corecursion. In this paper we study the relation between Curry-style and Church-style type assignments for such a system. The difference between the two styles is that in Church-style type assignment each bound variable is assigned a unique type, as in $\lambda x : \rho.M$, while in Curry-style the binding is untyped, as in $\lambda x.M$.

We give a domain-theoretic semantics for the Curry- and the Church-style system, and prove that they coincide under certain conditions. The proof uses hybrid logical relations, which are related to Tait's computability method and Girard's method of reducibility candidates. The reason for studying this domain-theoretic semantics is that it allows for very simply and elegant proofs of computational adequacy, and hence the correctness of program extraction.

We consider a prototypical functional programming language LoR that is interpreted in a domain which satisfies the following recursive domain equation $\mathrm{D} \simeq (\mathbf{1}+\mathrm{D}+\mathrm{D}+\mathrm{D}\times\mathrm{D}+[\mathrm{D}\to\mathrm{D}])_\perp$, where $\mathbf{1}$ denotes the sole-element domain $\{\star\}$, and $+, \times, [\_ \to \_]$ represent the usual continuous domain operations: separated sum, Cartesian product, and continuous function space. Thus, an element in D is in one of the following forms: $\perp$, $\star$, $\mathrm{Left}(a)$ or $\mathrm{Right}(a)$, $\mathrm{Pair}(a,b)$ and $\mathrm{Fun}(f)$, where a and b range over D, and f is a continuous function mapping from D to D.

We denote the powerset of D by $\wp(\mathrm{D})$, and if X is an entity in LoR, its interpretation by $[\![X]\!]$.

Let x range over a set of variables and C a set of constructors. The syntax of *Curry-style* LoR terms is given by the following grammar

$$\begin{aligned} M, N, R_i ::= {} & x \mid \lambda x.M \mid MN \mid C(M_1, \ldots, M_n) \mid \\ & \text{case } M \text{ of } \{C_i(x_i) \to R_i\}_{i\in\{1,\ldots,n\}} \end{aligned} \tag{1}$$

where in the term case M of $\{C_i(x_i) \to R_i\}_{i\in\{1,\ldots,n\}}$ all constructors C_i are distinct and each x_i is a vector of distinct variables. We consider the constructors Nil (nullary), Left, Right (unary), Pair(binary) and In (unary).

In the following we take the function space and abstraction as examples for illustration. For every environment $\eta : \mathbf{Var} \to \mathrm{D}$, we define

$$[\![MN]\!]\eta := \begin{cases} f([\![N]\!]\eta) & \text{if } [\![M]\!]\eta = \mathrm{Fun}(f), \\ \bot & \text{otherwise.} \end{cases}$$

$$[\![\lambda x.M]\!]\eta := \mathrm{Fun}(f) \text{ where } f \in [\mathrm{D} \to \mathrm{D}] \text{ s.t. } f(a) := [\![M]\!]\eta[x := a](a \in \mathrm{D}).$$

For Curry-style LoR terms, LoR's set of types, **Type**, is defined as follows:

$$\mathbf{Type} \ni \rho, \sigma, \tau ::= \alpha \mid \rho \to \sigma \mid \mathbf{1} \mid \rho \times \sigma \mid \rho + \sigma \mid \mu\alpha.\rho \mid \nu\alpha.\rho \tag{2}$$

where in $\mu\alpha.\rho$ and $\nu\alpha.\rho$, ρ is positive in α.

We define $[\![\rho \to \sigma]\!]\xi \subseteq \mathrm{D}$ for every environment $\xi : \mathbf{TVar} \to \wp(\mathrm{D})$ as $\{\mathrm{Fun}(f) \mid f \in [\mathrm{D} \to \mathrm{D}] \text{ s.t. } f([\![\rho]\!]\xi) \subseteq [\![\sigma]\!]\xi\}$.

For a type context $\Gamma = x_1 : \rho_1, \ldots, x_n : \rho_n$, we set $\mathrm{dom}(\Gamma) = \{x_1, \ldots, x_n\}$. Let $\eta \in [\![\Gamma]\!]\xi$ mean $\Gamma(x_i) = \rho_i \wedge \eta(x_i) \in [\![\rho_i]\!]\xi$ for all $i = 1, \ldots, n$.

Theorem 1 (Soundness For Curry-style Terms).
If $\Gamma \vdash M : \rho$ and $\eta \in [\![\Gamma]\!]\xi$, then $[\![M]\!]\eta \in [\![\rho]\!]\xi$.

For *Church-style* LoR terms, the term syntax differs from (1) in the case: $\lambda x : \rho.M$. For every environment $\zeta : [\mathrm{D} \to \mathrm{D}]^{\mathbf{TVar}}$ and $\eta : \mathbf{Var} \to \mathrm{D}$, we define

$$[\![MN]\!]^\zeta\eta := \begin{cases} f([\![N]\!]^\zeta\eta) & \text{if } [\![M]\!]^\zeta\eta = \mathrm{Fun}(f), \\ \bot & \text{otherwise.} \end{cases}$$

$$[\![\lambda x : \rho.M]\!]^\zeta\eta := \mathrm{Fun}(f) \text{ where } f \in [\mathrm{D} \to \mathrm{D}] \text{ s.t. } f(a) := [\![M]\!]^\zeta\eta[x := \langle\rho\rangle\zeta(a)].$$

In the definition of **Type** (2) we replace the constructs $\mu\alpha.\rho$ and $\nu\alpha.\rho$ by fixed-point type fix $\alpha.\rho$. We call the resulting type system **recursive types**. We define the semantics for recursive types by means of *finitary projections* [2] ($\langle\rho\rangle\zeta$), and then set $[\![\rho]\!]\zeta := (\langle\rho\rangle\zeta)(\mathrm{D})$ for all $\rho \in \mathbf{Type}$. For example, for every environment $\langle\rho\rangle : [[\mathrm{D} \to \mathrm{D}]^{\mathbf{TVar}} \to [\mathrm{D} \to \mathrm{D}]]$ and for all $a \in \mathrm{D}$, we define

$$(\langle\rho \to \sigma\rangle\zeta)(a) := \begin{cases} \mathrm{Fun}(g) \text{ where } g : [\mathrm{D} \to \mathrm{D}] \text{ s.t. } g = \langle\sigma\rangle\zeta \circ f \circ \langle\rho\rangle\zeta & \text{if } a = \mathrm{Fun}(f), \\ \bot & \text{otherwise.} \end{cases}$$

Theorem 2 (Soundness For Church-style Terms).
If $\Gamma \vdash M : \rho$ and $\eta \in [\![\Gamma]\!]\zeta$, then $[\![M]\!]^\zeta\eta \in [\![\rho]\!]\zeta$.

We define a hybrid logical relation to indicate two elements are equivalent elements of a certain type, e.g. $\sim^{\mathrm{R},\zeta}_{\rho\to\sigma} := \{(\bot, \bot)\} \cup \{(\mathrm{Fun}(f), \mathrm{Fun}(g)) \mid \forall a, b \in \mathrm{D}(a \sim^{\mathrm{R},\zeta}_\rho b \Rightarrow f(a) \sim^{\mathrm{R},\zeta}_\sigma g(b)) \wedge \ \langle\sigma\rangle\zeta \circ f \circ \langle\rho\rangle\zeta = \langle\sigma\rangle\zeta \circ g \circ \langle\rho\rangle\zeta\}$.

Let M be a Church-style term, M^- the corresponding Curry-style term and ρ a recursive type.

Lemma 1. *Assume $\mathrm{FV}(M) \subseteq \mathrm{dom}(\Gamma)$. Let $\eta \sim^{\mathrm{R},\zeta}_\Gamma \eta'$ denote the following: for all $x \in \mathrm{dom}(\Gamma)$, if $\Gamma(x) = \sigma$, then $\eta(x) \sim^{\mathrm{R},\zeta}_\sigma \eta'(x)$.*

$$\Gamma \vdash M : \rho, \eta \sim^{\mathrm{R},\zeta}_\Gamma \eta' \Rightarrow [\![M]\!]^\zeta\eta \sim^{\mathrm{R},\zeta}_\rho [\![M^-]\!]\eta'.$$

Theorem 3 (Coincidence).
If $\Gamma \vdash M : \rho$ *and* $\eta \in [\![\Gamma]\!]\zeta$, *then* $[\![M]\!]^{\zeta}\eta = \langle\rho\rangle\zeta([\![M^{-}]\!]\eta)$.

Theorem 3 continues to hold where limited form of recursion are added (guarded (co)recursion, structural recursion). This is the subject of ongoing work.

References

1. Berger, U., Seisenberger, M.: Proofs, programs, processes. LNCS **6158** (2010) 39–48
2. Amadio, R.M., Bruce, K.B., and Longo, G.: The finitary projection model for second order lambda calculus and solutions to higher order domain equations. In First Annual IEEE Symposium on Logic in Computer Science, IEEE Computer Society Press (1986) 122–130

Two Notions of Globular Setoids in Type Theory

Darin Morrison
dwm@cs.nott.ac.uk

University of Nottingham

Recent results have shown a relationship between higher dimensional category theory, type theory, and homotopy theory. The fundamental idea is that identity types in Martin-Löf type theory exhibit the structure of a weak groupoid [hof98] and if one considers their higher iterations, they in fact yield a weak ω-groupoid [lum09] [gar10]. This shows similarities to the notion of the fundamental groupoid of paths and higher homotopies of a topological space [awo09].

To explore these implications, we require a precise definition of weak ω-groupoids. To date, all definitions are quite complex, making reasoning difficult without sacrificing subtle but important detail. In the spirit of Voevodsky's Univalent Foundations programme [vov10] we aim to formalize these definitions in a proof assistant, easing the mathematician's burden of bookkeeping when reasoning.

We have begun by programming category theory in the Coq proof assistant. However, Coq is based on an intensional type theory and does not provide native support for extensional reasoning. We remedy this by building a theory around setoids (Bishop sets) and setoid morphisms (respectful functions), working directly with setoids where we would otherwise refer to the type of sets. We use this extensional core to create a library for reasoning about categories weakly enriched in setoids.

Category theory in type theory is an interesting problem and various parts of its story have already been told [hue98]. Our formalisation is new but inspired by previous efforts. It differs from past efforts in that we aim to build a library easy to understand, practical to use, and modular enough to admit future extension. We intend to use it for a wide variety of categorical studies.

To give a flavour of our library, we show our definition of category and use it to define and prove correct the product category. We use `ssreflect` [gon08] and `Russell` [soz08] extensively to shorten our code and improve readability.

```
Record Cat : Type :=
  { obj       :> Type
  ; hom       :  obj -> obj -> ESet
  ; one       :  forall A, EFun One (hom A A)
  ; cmp       :  forall A B C, EFun (hom B C × hom A B) (hom A C)
  ; unit_id_l :  Unit_Id_L one cmp
  ; unit_id_r :  Unit_Id_R one cmp
  ; assoc_cmp :  Assoc_Cmp     cmp }.

Program Definition Prod (ℂ₁ ℂ₂ : Cat) : Cat :=
  {| obj := ℂ₁ * ℂ₂
   ; hom := fun π₁ π₂    => hom (fst π₁) (fst π₂) × hom (snd π₁) (snd π₂)
   ; one := fun π        => const _ (one (fst π) tt , one (snd π) tt)
   ; cmp := fun π₁ π₂ π₃ => {| map := fun πf => ( cmp (fst (fst πf) , fst (snd πf))
                                                , cmp (snd (fst πf) , snd (snd πf))) |} |}.
Next Obligation.
  rewrite /Stable /= => a b φ.
  elim φ => φ₁ φ₂; elim φ₁ => φ₁₁ φ₁₂; elim φ₂ => φ₂₁ φ₂₂; clear φ φ₁ φ₂.
  by [ split; apply stb ]. Qed.
```

```
Next Obligation.
  rewrite /Unit_Id_L /= => A B f. by [ split; apply unit_id_l ]. Qed.
Next Obligation.
  rewrite /Unit_Id_R /= => A B f. by [ split; apply unit_id_r ]. Qed.
Next Obligation.
  rewrite /Assoc_Cmp /= => A B C D h g f. by [ split; apply assoc_cmp ]. Qed.
```

As a first step toward using our library for reasoning about weak ω-groupoids and eventually homotopy type theory, we focus on implementing globular setoids. Globular sets are used to define weak ω-groupoids in the style of Leinster [lei04]. In our type theoretic setting, we have two versions of globular setoids: one based on the idea of setoid-valued presheafs on the globe category 𝔾 and the other on a direct coinductive definition of a kind of setoid-carrying ω-quiver.

For the type theorist, the second version is much simpler when working in a foundation that allows for coinductive definitions. Coq, based on the coinductive calculus of constructions, allows for such definitions:

```
CoInductive ωQuiver : Type :=
  { obj :> ESet
  ; hom :  obj -> obj -> ωQuiver }.
```

We would prefer to use this version in building up to the full definition of weak ω-groupoids. Our task is to show that the category of ω-quiver style globular setoids is equivalent to the category of presheaf style globular setoids, since it is the presheaf version that most closely follows the existing literature. We intend to do so by showing they are isomorphic objects in the appropriate notion of the bicategory of weakly enriched setoid categories [wil05].

References

[awo09] Awodey, S., Warren, M. *Homotopy Theoretic Models of Identity Types*. Mathematical Proceedings of the Cambridge Philosophical Society (2009)

[lei04] Leinster, T. *Higher Operads, Higher Categories*. London Mathematical Society Lecture Note Series, vol. 298, Cambridge University Press, Cambridge (2004)

[gar10] Garner, R., van den Berg, B. *Types are Weak ω-groupoids*. Proc. Lond. Math. Soc. (2010)

[gon08] Gonthier, G., Mahboubi, A., Tassi, E. *A Small Scale Reflection Extension for the Coq system* Inria Research Report RR-6455 (2008)

[hof98] Hofmann, M., Streicher, T. *The Groupoid Interpretation of Type Theory*. Twenty-five Years of Constructive Type Theory, Venice (1995)

[hue98] Huet, G., Saïbi, A. *Constructive Category Theory*. In proc. Joint CLICS-TYPES Workshop on Categories and Type Theory, Göteborg (1998)

[lum09] Lumsdaine, P. F. *Weak ω-Categories from Intensional Type Theory*. TCLA 2009, Brasília, Logical Methods in Computer Science, Vol. 6, issue 23, paper 24 (2009)

[soz08] Sozeau, M. *Un Environnement pour la Programmation avec Types Dépendants*. PhD Thesis, Université de Paris-Sud 11 (2008)

[vov10] Voevodsky, V. *Univalent Foundations of Mathematics* `http://www.math.ias.edu/~vladimir/Site3/Univalent_Foundations.html`

[wil05] Wilander, K. O. *An E-Bicategory of E-Categories*. TYPES 2004, U.U.D.M. Report 2005:48 (2005)

Elimination Principles for Initial Dialgebras

Fredrik Nordvall Forsberg*
csfnf@swansea.ac.uk
Department of Computer Science, Swansea University, UK

Joint work with Thorsten Altenkirch, Peter Morris and Anton Setzer.

Initial algebra semantics provides a concise and uniform way to model inductive data types. The data type is regarded as the carrier of the initial algebra $(\mu F, \mathrm{in})$, i.e. $\mathrm{in} : F(\mu F) \to \mu F$, of an endofunctor $F : \mathbb{C} \to \mathbb{C}$ for some suitable category $\mathbb{C}$. Initiality gives an iteration operation $\mathrm{fold}(h) : \mu F \to A$ for any other algebra (A, h). However, working in dependent type theory [8][1] , we would prefer to have dependent recursion – i.e. an "induction principle" under the propositions-as-types interpretation:

$$\frac{x : \mu F \vdash P(x) : \mathrm{Set} \qquad x : F(\mu F), \widetilde{x} : \Box_F(P, x) \vdash \mathrm{step}(x, \widetilde{x}) : P(\mathrm{in}(x))}{\mathrm{elim}(P, \mathrm{step}) : (x : \mu F) \to P(x)} \tag{1}$$

Here, $\Box_F(P, x)$ is the set of proofs that P holds for the elements of μF that x is built up from, i.e. the induction hypothesis. It is defined in such a way that there is an isomorphism $\varphi_F : F((\Sigma x : \mu F) P(x)) \to (\Sigma x : F(\mu F)) \Box_F(P, x)$ with $\pi_0 \circ \varphi_F = F(\pi_0)$.

Ghani, Johann and Fumex [4] recently extended work by Hermida and Jacobs [6], showing that such an eliminator can be defined for every initial F-algebra (under some mild assumptions on $\mathbb{C}$). There are, however, other meaningful forms of data types which are not covered by these results [9]. In order to accommodate these more exotic definitions, we move to the setting of dialgebras, as introduced by Hagino [5] in his thesis:

Definition 1 (Dialgebras). *Let $F, G : \mathbb{C} \to \mathbb{D}$ be functors. The category $\mathit{Dialg}(F, G)$ has as objects pairs (A, f) where $A \in \mathbb{C}$ and $f : F(A) \to G(A)$. A morphism from (A, f) to (A', f') is a morphism $h : A \to A'$ in $\mathbb{C}$ such that $G(h) \circ f = f' \circ F(h)$.*

We will mostly be interested in the case when G is some kind of forgetful functor, for example $G : \mathrm{Dialg}(F, G') \to \mathbb{C}$, $G(A, f) = A$. Note also that with G the identity functor on $\mathbb{C}$, we regain ordinary algebras as a special case.

Our goal is to show that one can define an eliminator for every initial dialgebra. This requires us to generalise the concept of an eliminator to the new setting. In order to state what an eliminator for a (F, G)-dialgebra is, where F, $G : \mathbb{C} \to \mathbb{D}$,

* Supported by EPSRC grant EP/G033374/1, Theory and applications of induction-recursion.

[1] We will write $(x : A) \to B(x)$ for the dependent function space (consisting of functions f such that $f(a) : B(a)$ for every $a : A$) and $(\Sigma x : A) B(x)$ for the type of dependent pairs with elements $\langle a, b \rangle$, where $a : A$ and $b : B(a)$.

we require that both $\mathbb{C}$ and $\mathbb{D}$ have some extra structure, namely that they are categories with families [3, 7], i.e. categorical models of dependent type theory. First, let us introduce the category of families of sets:

Definition 2 (Families of sets). *The category Fam(Set) has as objects pairs (I, X) where I is a set (the* index set*) and $X : I \to Set$ is a functor (the set I is here regarded as a discrete category), i.e. a family of I-indexed sets. A morphism from (I, X) to (J, Y) is a pair (f, g) where $f : I \to J$ and $g : X \to Y \circ f$ is a natural transformation, i.e. a family of I-indexed functions $g_i : X(i) \to Y(f(i))$.*

Equivalently (in the technical sense), one could work with the arrow category $\mathrm{Set}^{\to}$. From a type theoretic perspective, however, Fam(Set) seems more natural.

Consider a functor $F : \mathbb{C} \to \mathrm{Fam(Set)}$. Unfolding the definitions, we see that its object part is given by a mapping sending X in $\mathbb{C}$ to a set $F_0(X)$ and a family $F_1(X) : F_0(X) \to \mathrm{Set}$ of $F_0(X)$-indexed sets. Similarly, the morphism part splits up into two components and we write $F = (F_0, F_1)$ for such a functor.

Dybjer [3] used this structure for categorical models of dependent type theory. Roughly, a base category $\mathbb{C}$ is treated as a category of contexts, and a functor $F = (\mathrm{Ty}, \mathrm{Tm}) : \mathbb{C} \to \mathrm{Fam(Set)}$ gives, for each context Γ, a set $\mathrm{Ty}(\Gamma)$ of types in context Γ, together with a family $\mathrm{Tm}(\Gamma) : \mathrm{Ty}(\Gamma) \to \mathrm{Set}$ with $\mathrm{Tm}(\Gamma, \sigma)$ the set of terms of type σ in context Γ. The morphism part of F gives substitution of both types and terms. We also need a comprehension operation $\Gamma \cdot \sigma$ to extend a context Γ with a new variable of some well-formed type σ in context Γ. More precisely, we have:

Definition 3 (Categories with families). *A* category with families *is given by*

- *A category $\mathbb{C}$ (the* category of contexts*) with a terminal object (the* empty context*);*
- *A functor $F = (Ty, Tm) : \mathbb{C}^{op} \to Fam(Set)$ (the* types and terms functor*). For the morphism part, we introduce the notation $\_\{\cdot\}$ for both types and terms, i.e. if $f : \Delta \to \Gamma$ then $\_\{f\} : Ty(\Gamma) \to Ty(\Delta)$ and for every $\sigma \in Ty(\Delta)$ we have $\_\{f\} : Tm(\Delta, \sigma) \to Tm(\Gamma, \sigma\{f\})$;*
- *For each object Γ in $\mathbb{C}$ and $\sigma \in Ty(\Gamma)$ an object $\Gamma \cdot \sigma$ in $\mathbb{C}$ (the* context comprehension*) together with a morphism $\boldsymbol{p} : \Gamma \cdot \sigma \to \Gamma$ (the* first projection*) and a term $\boldsymbol{v} \in Tm(\Gamma \cdot \sigma, \sigma\{\boldsymbol{p}\})$ (the* second projection*) satisfying a universal property (omitted here).*

Example 4 (Set as a category with families). The category Set can be extended to a category with families by defining

$$
\begin{aligned}
\mathrm{Ty}(\Gamma) &= \{A \mid A : \Gamma \to \mathrm{Set} \text{ is a } \Gamma\text{-indexed family of (small) sets}\} \ , \\
A\{f\} &= A \circ f \in \mathrm{Ty}(\Delta) \qquad (f : \Delta \to \Gamma) \ , \\
\mathrm{Tm}(\Gamma, A) &= (x : \Gamma) \to A(x) \ , \\
a\{f\} &= a \circ f \in \mathrm{Tm}(\Delta, A\{f\}) \qquad (f : \Delta \to \Gamma) \ , \\
\Gamma \cdot A &= (\Sigma x : \Gamma) A(x) \ .
\end{aligned}
$$

In light of Example 4, we are however interested in a slightly different interpretation of the family structure of a category with families. We see $A \in \mathrm{Ty}(\Gamma)$ as a property of Γ, $g \in \mathrm{Tm}(\Gamma, \sigma)$ as a kind of dependent function and $\Gamma \cdot \sigma$ as a kind of dependent pair, with projections $\mathbf{p}$ and $\mathbf{v}$.

Guided by this, we can now define what the type of the induction hypothesis and the elimination principle for an (F, G)-dialgebra $(\mu FG, \mathrm{in})$ should be: in the simple situation in (1), $\Box_F$ lifted properties of μF to properties of $F(\mu F)$ such that there was an isomorphism $\varphi_F : F((\Sigma x{:}\mu F)P(x)) \to (\Sigma x{:}F(\mu F))\Box_F(P, x)$ with $\pi_0 \circ \varphi_F = F(\pi_0)$. In fact, this uniquely determines $\Box_F$ up to isomorphism.

Abstracting to an arbitrary category with families, we get: If $P \in \mathrm{Ty}_{\mathbb{C}}(\mu FG)$, then $\Box_F(P) \in \mathrm{Ty}_{\mathbb{D}}(F(\mu FG))$ is the unique-up-to-isomorphism type such that there is an isomorphism $\varphi_F : F(\mu FG \cdot P) \to F(\mu FG) \cdot \Box_F(P)$ with $\mathbf{p} \circ \varphi_F = F(\mathbf{p})$. In particular, this means that $\Box_{\mathrm{id}}(P) = P$. More concretely, $\Box_F$ can be constructed explicitly from sigma types and extensional identity types, and every category with finite limits can be extended to a category with families supporting these [2].

We can now translate the elimination rule (1) to our more abstract setting. The generic elimination rule becomes

$$\frac{P \in \mathrm{Ty}(\mu FG) \qquad \mathrm{step} \in \mathrm{Tm}(F(\mu FG) \cdot \Box_F(P), \Box_G(P)\{\mathrm{in} \circ \mathbf{p}\})}{\mathrm{elim}(P, \mathrm{step}) \in \mathrm{Tm}(G(\mu FG), \Box_G(P))} \tag{2}$$

For $G = \mathrm{id} : \mathbb{C} \to \mathbb{C}$, this reduces to the ordinary elimination rules for ordinary algebras (interpreted in $\mathbb{C}$). For $\mathbb{C} = \mathrm{Set}$, we get exactly the usual elimination rule. However, we can also treat other forms of inductive data types, such as induction-induction (see our CALCO paper [1] for details).

The proof of the equivalence between initiality and the elimination principle can still be carried out in this more general setting. Thus, we get a uniform proof for different sorts of induction such as ordinary induction, induction-induction, induction-recursion, ...

References

1. Altenkirch, T., Morris, P., Nordvall Forsberg, F., Setzer, A.: A categorical semantics for inductive-inductive definitions. In: CALCO (2011)
2. Clairambault, P., Dybjer, P.: The biequivalence of locally cartesian closed categories and Martin-Löf type theories. In: TLCA (2011)
3. Dybjer, P.: Internal type theory. LNCS 1158, 120–134 (1996)
4. Ghani, N., Johann, P., Fumex, C.: Fibrational induction rules for initial algebras. In: Computer Science Logic. pp. 336–350. Springer (2010)
5. Hagino, T.: A Categorical Programming Language. Ph.D. thesis, Edinburgh (1987)
6. Hermida, C., Jacobs, B.: Structural induction and coinduction in a fibrational setting. Information and Computation 145(2), 107 – 152 (1998)
7. Hofmann, M.: Syntax and semantics of dependent types. In: Semantics and Logics of Computation, pp. 79 – 130. Cambridge University Press (1997)
8. Martin-Löf, P.: Intuitionistic type theory. Bibliopolis Naples (1984)
9. Nordvall Forsberg, F., Setzer, A.: Inductive-inductive definitions. In: Computer Science Logic. LNCS, vol. 6247, pp. 454–468 (2010)

Author Index