From Categorical Logic to Facebook Engineering

Peter O'Hearn
Facebook & University College London

Abstract

I chart a line of development from category-theoretic models of programs and logics to automatic program verification/analysis techniques that are in deployment at Facebook. Our journey takes in a number of concepts from the computer science logician's toolkit – including categorical logic and model theory, denotational semantics, the Curry-Howard isomorphism, substructural logic, Hoare Logic and Separation Logic, abstract interpretation, compositional program analysis, the frame problem, and abductive inference.

In the 1960s and early 1970s a deep connection between logic, types and categories was uncovered, which has become a cornerstone of logic in computer science. In particular, Lambek described how deductive systems of logic corresponded to categorical structures – most famously for intuitionistic logic and cartesian closed categories with finite coproducts, but also covering various forms of substructural logic connecting to monoidal closed categories [13], [14].

Around the same time, logicians were advancing a form of possible world semantics for relevant logics [28], [26]. Relevant logics typically admitted the structural rule of Contraction but not Weakening, but their semantics could be immediately adapted to cover Contractionless logics and even those without commutativity of conjunction.

And in yet another line of development the denotational semantics of programming languages was being advanced by Scott and Strachey [27].

I did research involving these topics as an academic, and now I work for Facebook Engineering on automated software tools to help engineers make more reliable code. Some of the work we do on tools can be traced directly back to this foundational work in logic and semantics; the talk describes the thread from the abstract semantic concepts through to engineering tools in current deployment, and in this accompanying paper I give a brief description of the main points.

In: *30th IEEE/ACM Symposium on Logic in Computer Science, 2015, pp17-21.* ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=7174865

1) A Denotational Model: The starting point is a particular denotational model. Types A in the model denote functions from finite sets (of locations, or resources) X to sets AX; we think of AX as containing elements that only access the resources in X. We can define operations on types as follows

$$\begin{array}{lcl}
1X & = & \{e\} \\
(A \times B)X & = & AX \times BX \\
(A \Rightarrow B)X & = & AX \Rightarrow BX \\
0X & = & \emptyset \\
(A + B)X & = & AX + BX \\
(emp)X & = & \left\{ \begin{array}{ll} \{e\} & \text{if } X = \emptyset \\ \emptyset & \text{otherwise} \end{array} \right. \\
(A \mathrel{-\!\!*} B)X & = & \Pi_{Y \perp X} AY \Rightarrow B(Y \uplus X) \\
(A * B)X & = & \{(Y, Z, a \in AY, b \in BZ) \mid X = Y \uplus Z\}
\end{array}$$

where $\Rightarrow$, $\times$, $+$ on the right are set-theoretic function space, cartesian product and disjoint union.

The $A \Rightarrow B$ function type is the familiar one from typed λ-calculus. In contrast, $A \mathrel{-\!\!*} B$ can be used to model functions that only ever access disjoint resources from their arguments; the quantification $\Pi_{Y \perp X}$ in the definition is over finite sets Y disjoint from X. $A \times B$ is the cartesian product, the type of arbitrary pairs, where $A * B$ is the type of pairs that access disjoint resources ($Y \uplus Z$ in the definition denoting union of disjoint sets). For instance, a type of locations can be modelled by Loc where $LocX = X$ for each X; then any $(Y, Z, a, b) \in (Loc * Loc)X$ must have a, b being distinct locations from X: they are not "aliases". A version of this model was invented originally [17] to interpret Reynolds's Syntactic Control of Inferference, a novel, early use of substructural type theory to control sharing of resources [23].

Looking at the mathematical structure of this model, in a way that abstracts from its details, leads to a new mathematical logic, Bunched Logic. Category theory is the key tool for performing this abstraction.

2) Doubly-Closed Categories and Bunched Logic: Consider the category in which the objects are functions from finite sets of locations to sets, and where a morphism η from A to B is a collection of set-theoretic functions $\eta X : AX \Rightarrow BX$, indexed by finite sets X. That is, we consider the product category $\mathbf{Set}^C$ where C is the powerset of a given set of locations. The central point is that this category has two closed structures: a cartesian closed structure and a symmetric monoidal closed structure. Writing $A \longrightarrow B$ for the set of morphisms (hom set) from A to B, saying we have these two closed structures is to say that there are isomorphisms of hom sets

$$\frac{\Gamma \times A \longrightarrow B}{\Gamma \longrightarrow A \Rightarrow B} \qquad \frac{\Gamma * A \longrightarrow B}{\Gamma \longrightarrow A \mathrel{-\!\!*} B}$$

for objects Γ, A and B, where we mean an isomorphism here of maps of the form above the line with those below (these isomorphisms additionally need to be what is called natural in Γ and B). So the function types and their corresponding products both fit together in the way that in λ-calculus is understood via Currying. The difference between these correspondences is that the product $*$ does not admit projections or the duplication corresponding to maps of the shape $A * B \longrightarrow A$ or $A \longrightarrow A * A$, maps which are present for the cartesian product $\times$.

This "doubly closed" structure leads to a type theory, bunched type theory [16], [18], [22], where the two isomorphisms just stated correspond to introduction rules for function types. It also leads via the Curry-Howard propositions as types correspondence to a logic, Bunched Logic [19], [22], which mixes intuitionistic logic and what nowadays is called multiplicative intuitionistic linear logic (formerly, BCI logic, or symmetric Lambek calculus). Logically, our isomorphisms of hom sets turn into statements of variants of the deduction theorem of logic.

Summing up, a categorical model of bunched logic is a cartesian closed category with finite coproducts possessing an additional symmetric monoidal closed structure. Any logical formula built using intuitionistic connectives $(1, \wedge, \Rightarrow, 0, \vee)$ plus emp, $*$ and $\mathrel{-\!\!*}$ can be interpreted as an object in such a category, and an entailment judgement $P \vdash Q$ is then interpreted in terms of the existence of a morphism from the object determined by P to that by Q; generally, with an appropriate proof theory, each proof determines such a morphism. This perspective is described briefly in the paper by O'Hearn and Pym introducing Bunched Logic, and more fully in a comprehensive monograph on the logic by Pym [22]. [Bunched Logic is named by reference to tree-like structures (bunches) used in its natural deduction and Gentzen proof theories; we do not need to talk about bunches in this paper.]

If we restrict attention to posets (collapsed categories), models for Bunched Logic can be stated as follows:

> An algebraic model of Bunched Logic is a poset that is a Heyting algebra together with an additional ordered commutative monoid structure $(*, emp)$ which is residuated (having the $\mathrel{-\!\!*}$ adjunct).

We have described how categorical structure extracted from the denotational model described earlier can give rise to a mathematical logic, but we have not said why such a logic might be of interest. This stems from a connection to the notion of shared resources, which was hinted at in the model we began with, and developed further in a form of semantics to which we now turn.

3) Possible Worlds and Resources: In a remarkable coincidence of Australian discoveries around the same time, the "ternary relation" semantics discovered by the relevantists Routley and Meyer [26] was very closely related to a general construction due to Day [9].

$$(A * B)X \;=\; \int^{YZ} AY \times BZ \times \mathcal{P}(X, Y, Z)$$

$$x \models P * Q \;\Leftrightarrow\; \exists yz.\, y \models P \;\wedge\; z \models Q \;\wedge\; Rxyz$$

The denotational model of $A * B$ we gave earlier is an instance of the first formula, which is sometimes referred to as the Day convolution: the co-end $\int^{YZ}$ is a category-theoretic cousin of an existential type, and the particular model earlier gives witnesses of the existential. $\mathcal{P}(X, Y, Z)$ is what is called a promonoidal structure (in our partiular case indicating that $X = Y \uplus Z$). The second formula is the interpretation of the relevantists' "fusion" connective in the ternary relation semantics, where $R \subseteq W^3$ is a ternary relation and the $x, y, z \in W$ are possible worlds.

Doubly-closed categories model the proof theory of (intuitionistic) Bunched Logic using a variant on the Heyting view of proofs as functions. Converting over to a possible world semantics following the relevantists' work leads to a declarative semantics of Bunched Logic, advanced by Pym, which he then justified intuitively by appeal to a notion of resource: $P * Q$ is true of a resource just when it is possible to decompose the resource into parts that make the conjuncts true. Pym gives explanations of all the intuitionistic connectives as well as emp, $*$ and $\mathrel{-\!\!*}$ in terms of resource [22], [21]. Besides giving an inuitive declarative reading of formulae, Pym's semantics opened the way for a semantics of a boolean variant as well.

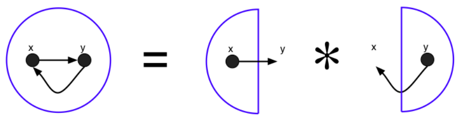
> An algebraic model of boolean Bunched Logic is a poset that is a boolean algebra together with an additional ordered commutative monoid structure $(*, emp)$ which is residuated (having the $\mathrel{-\!\!*}$ adjunct).

Any ternary relation model satisfying axioms forcing commutativity and associativity of $*$ gives rise to a model of boolean Bunched Logic.

Separation Logic is an extension of Hoare Logic based on a particular ternary-relation model of Bunched Logic where the "resources" are "heaplets", portions of global program heaps.

4) Separation Logic: Separation Logic was developed originally in three papers by O'Hearn, Reynolds, Ishtiaq and Yang [24], [12], [15]. The early work on Separation Logic up until 2002 was summed up systematically in an influential survey paper by Reynolds, published in the LICS'02 proceedings [25].

Mathematically, we can say that a heaplet $h : L \rightharpoonup_f V$ is a finite partial function from locations to values. The ternary relation Rhh_1h_2 says that heaplets h_1 and h_2 have disjoint domains and that h is their union. The ternary-relation semantics above then specializes to the conjunction of Separation Logic. With this semantics we can describe heap partitionings as in the following picture: the formula $x \mapsto y * y \mapsto x$ of a separating conjunction of points-to facts corresponds to



Separation Logic provided a new modular way of reasoning about programs with pointers and dynamic allocation. Its key inference rule

$$\frac{\{P\}C\{Q\}}{\{P * F\}C\{Q * F\}} \text{ Frame Rule}$$

allows a description of unaltered state F to be tacked on to a specification, in away that lets the specifications themselves to concentrate on only those cells that a program touches (the footprint). The rule is named after the frame problem from artificial intelligence, where F is the frame (evoking the idea of the frame as the unchanging part in an animation).

A typical instance of the frame rule

$$\frac{\{\texttt{tree}(i)\}\texttt{DispTree}(i)\{emp\}}{\{\texttt{tree}(i) * \texttt{tree}(j)\}\texttt{DispTree}(i)\{emp * \texttt{tree}(j)\}}$$

is used during the proof of a recursive procedure to dispose all the nodes in a tree, where correctness relies on the fact that the recursive call on one subtree, described by assertion $\texttt{tree}(i)$, does not affect the other, described by the frame $\texttt{tree}(j)$.

5) Automation, Frame Inference and Abduction: The first Separation Logic verification tool, Smallfoot [1], [2], used a fragment of Separation Logic (symbolic heaps) shown to be decidable by Berdine and Calcagno. Proofs of imperative statements in Smallfoot themselves worked imperatively, utilizing the separating conjunction to update formulae in-place in a way reminiscent of the imperative in-place update of concrete program execution.

Crucially, in-place reasoning worked modularly, for applications of entire procedures, and not only for individual heap operations. Essential for this was the identification by Calcagno and O'Hearn of the notion of frame inference, as a way of automating the use of the frame rule.

Frame Inference: find F making $A \vdash B * F$ true.

In automating a proof of tree disposal as above, the question would be

$$\texttt{tree}(i) * \texttt{tree}(j) \vdash \texttt{tree}(i) * F$$

an evident solution of which is $F = \texttt{tree}(j)$. Calcagno designed and implemented the first frame-inferring theorem prover based on using information from failed entailment proofs to find frames. Frame inference is the workhorse of a number of automated reasoning tools, particularly for interprocedural program analysis.

Smallfoot required the user to provide preconditions and postconditions for procedures, as well as loop invariants. The first step towards higher automation was the inference of loop invariants, following the usual strategy of abstract interpretation [8]. This was done by adapting ideas from Distefano's PhD thesis [10] to the assertion language of Smallfoot [11]

The next step forward for automation was to infer preconditions. The basic strategy came from O'Hearn's notion of local reasoning [15]: the idea was to aim for a canonical precondition, describing the footprint of a piece of code, and then using a standard forwards abstract interpretation to obtain a postcondition. A beginning was made by Yang, based on the idea of synthesizing $\mapsto$-facts (describing heap cells) from failed program-proof attempts and using these to discover preconditions to allow the proofs to go through [5]. Then, a breakthrough was made by Distefano who showed how to use

Abduction: find M making $A * M \vdash B$ true

in concert with frame inference to automatically stitch together specifications in a bottom-up interprocedural program analysis. Calcagno, Distefano, O'Hearn and Yang then worked on the implementation and the theory of a compositional program analysis method based on abduction and frame inference [6].

Abduction was originally formulated by the philosopher Charles Peirce as part of his explanation of the scientific process: abductive inference concerned the generation of hypotheses. In retrospect, it seems a natural fit to use abduction to generate preconditions for computer programs.

6) Industrial Application: With abduction plus frame inference, scalability to large code bases became possible. Procedures could be analyzed in isolation with compact specifications concentrating on the footprints, and changes to the large bases could be analyzed incrementally. Because preconditons and loop invariants were inferred, analysis could be applied to bare code, without waiting for the programmer to insert specifications.

Spurred by these advances, Calcagno and Distefano made the brave decision to form a startup company, Monoidics, to push the ideas further, and they pursuaded O'Hearn to join them. Monoidics' INFER static analyzer [3] is now being further developed inside Facebook, following the aquisition of Monoidics in 2013 [7].

As of this writing INFER is deployed and running to verify select properties of modifications to Facebook's mobile code (the properties include null dereferences and resource and memory leaks). Because of the compositional way that the analysis works, a code change can be analyzed without re-analyzing an entire codebase. In a typical month there are thousands of such code modifications analyzed by INFER, during which time there are millions of calls to an internal theorem prover that solves frame inference and abduction problems for a fragment of Separation Logic. Hundreds of potential bugs reported by INFER are fixed each month by developers before they ever get committed to Facebook code and deployed to mobile phones.

See [4] for more information on the deployment of INFER within Facebook.

In this short paper I have described a journey from abstract theory through to industrial application. A message I wish to convey is that logic in computer science is a rich area with a breadth of ideas, and drawing upon this breadth can boost impact beyond what might be possible from subareas in isolation. In the particular story the key concepts underpinning application are Separation Logic, Frame Inference, Abduction, Abstract Interpretation, and Compostionality, with category-theoretic models and logic, and the Curry-Howard isomorphism connecting propositions and types, playing earlier enabling roles. I make no claim whatsoever that every bit of the theory at each stage is necessary to the application. In fact, it is natural and even desirable in linking theory and practice that not all of the ideas present one stage of development are necessary to refer to when working at another. In any case, we have here but one story of applying ideas from logic in computer science; there are and will be many others.

References

[1] J. Berdine, C. Calcagno, and P.W. O'Hearn. Symbolic execution with separation logic. In K. Yi, editor, *APLAS 2005*, volume 3780 of *LNCS*, 2005.

[2] J. Berdine, C. Calcagno, and P.W. O'Hearn. Smallfoot: Automatic modular assertion checking with separation logic. In *4th FMCO*, pp115-137, 2006.

[3] C. Calcagno and D. Distefano. Infer: An automatic program verifier for memory safety of C programs. In *NASA Formal Methods - Third International Symposium, NFM 2011, Pasadena, CA, USA, April 18-20, 2011. Proceedings*, pages 459–465, 2011.

[4] C. Calcagno, D. Distefano, J. Dubreil, D. Gabi, P. Hooimeijer, M. Luca, P.W. O'Hearn, I. Papakonstantinou, J. Purbrick, and D. Rodriguez. Moving fast with software verification. In *NASA Formal Methods - 7th International Symposium, NFM 2015, Pasadena, CA, USA, April 27-29, 2015, Proceedings*, pages 3–11, 2015.

[5] C. Calcagno, D. Distefano, P. O'Hearn, and H. Yang. Footprint analysis: A shape analysis that discovers preconditions. In *SAS: Static Analysis, 14th International Symposium*, volume 4634 of *LNCS*, pages 402–418, 2007.

[6] C. Calcagno, D. Distefano, P. W. O'Hearn, and H. Yang. Compositional shape analysis by means of bi-abduction. *J. ACM*, 58(6):26, 2011. Preliminary versin in POPL'09.

[7] Josh Constine. Facebook acquires assets of UK mobile bug-checking software developer Monoidics. http://techcrunch.com/2013/07/18/facebook-monoidics.

[8] P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. 4th POPL, pp238-252, 1977.

[9] B. J. Day. On closed categories of functors. In S. Mac Lane, editor, *Reports of the Midwest Category Seminar*, volume 137 of *Lecture Notes in Mathematics*, pages 1–38. Springer-Verlag, Berlin-New York, 1970.

[10] D. Distefano. *On model checking the dynamics of object-based software: a foundational approach*. PhD thesis, University of Twente, 2003.

[11] D. Distefano, P. O'Hearn, and H. Yang. A local shape analysis based on separation logic. 16th TACAS, pp287–302, 2006.

[12] S. Isthiaq and P. W. O'Hearn. BI as an assertion language for mutable data structures. In *28th POPL*, pages 36–49, 2001.

[13] J. Lambek. Deductive Systems and Categories I,II, and III. J. Math. Systems Theory; 1968, 1969, 1972.

[14] J. Lambek and P. Scott. *Introduction to Higher-Order Categorical Logic*. Cambridge University Press, 1986.

[15] P. O'Hearn, J. Reynolds, and H. Yang. Local reasoning about programs that alter data structures. In *15th CSL*, pp1–19, 2001.

[16] P. W. O'Hearn. Resource interpretations, bunched implications and the $\alpha\lambda$-calculus. In *Typed λ-calculus and Applications*, J-Y Girard editor, L'Aquila, Italy, April 1999. Lecture Notes in Computer Science 1581.

[17] P. W. O'Hearn. A model for syntactic control of interference. *Mathematical Structures in Computer Science*, 3(4):435–465, 1993.

[18] P. W. O'Hearn. On bunched typing. *Journal of Functional Programming*, 2003. 13(4): 747-796.

[19] P. W. O'Hearn and D. J. Pym. The logic of bunched implications. *Bulletin of Symbolic Logic*, 5(2):215–244, June 99.

[20] P. W. O'Hearn and R. D. Tennent, editors. *Algol-like Languages*. Two volumes, Birkhauser, Boston, 1997.

[21] D. Pym, P. W. O'Hearn, and H. Yang. Possible worlds and resources: The semantics of BI. *Theoretical Computer Science*, 315(1):257–305, May 2004.

[22] D.J. Pym. *The Semantics and Proof Theory of the Logic of Bunched Implications*. Applied Logic Series. Kluwer Academic Publishers, 2002.

[23] J. C. Reynolds. Syntactic control of interference. In *5th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 39–46, Tucson, Arizona, January 1978. ACM, New York. Also in [20], vol 1.

[24] J. C. Reynolds. Intuitionistic reasoning about shared mutable data structure. In Jim Davies, Bill Roscoe, and Jim Woodcock, editors, *Millennial Perspectives in Computer Science*, pages 303–321, Houndsmill, Hampshire, 2000. Palgrave.

[25] J. C. Reynolds. Separation logic: A logic for shared mutable data structures. In *17th LICS*, pp 55-74, 2002.

[26] R. Routley and R. K. Meyer. The semantics of entailment, I. In H. Leblanc, editor, *Truth, Syntax and Modality*, pages 199–243. North-Holland, 1973.

[27] D. S. Scott and C. Strachey. Toward a mathematical semantics for computer languages. pages 19–46. Proceedings of the Symposium

on Computers and Automata. Also Technical Monograph PRG-6, Oxford University Computing Laboratory, Programming Research Group, Oxford.

[28] A. Urquhart. Semantics for relevant logics. *Journal of Symbolic Logic*, pages 1059–1073, 1972.