Language-integrated query using comprehension syntax: state of the art, open problems, and work in progress

James Cheney
University of Edinburgh
jcheney@inf.ed.ac.uk

Sam Lindley
University of Edinburgh
Sam.Lindley@ed.ac.uk

Philip Wadler
University of Edinburgh
wadler@inf.ed.ac.uk

Abstract

Comprehension syntax has proved to be a powerful tool for embedding query language features into strongly-typed functional languages. This work may also be applicable to other programming models (data-parallel, GPU, MapReduce) and deserves to be better-known to the data-centric programming community. This talk will give a technical overview of the highlights in the development of monadic comprehensions, particularly focusing on language designs, theory, and systems, identify current open problems, and discuss some current work.

1. The state of the art

Ideas from programming language theory, including monads and comprehensions, have played an important role in the development of current approaches to language-integrated query, such as Microsoft's LINQ. The Nested Relational Calculus (NRC) of Buneman et al. [1] provides an elegant foundation for query languages based on monadic comprehensions over collection types such as sets, lists and bags. We will survey language designs for query integration, underlying conservativity and normalization results, and implemented systems in this area. This abstract briefly summarizes related work, omitting technical detail that will be presented in the talk.

1.1 Language designs

NRC's use of collection types helps mitigate the impedance mismatch between programming language and database, but there is still a semantic mismatch: there are some things that programs can do that queries cannot (and potentially vice versa). For example, we cannot print to the console from inside a query, and we cannot use recursion within a query (though we may want to use recursion to *build* queries).

One default solution is to simply disallow allow recursion/effects (Kleisli), or allow them but fail at runtime if expectations are violated (F#, C#). These systems do not provide a guarantee that a given piece of code generates a single or statically bounded number of queries. This can lead to *query avalanches* (or the $N + 1$ query problem), where for example to evaluate a relational join of two tables, the program loads the contents of one table and then issues N queries on the second table, instead of issuing a single join query. Ferry [10] does provide a bound on the number of queries that will be generated if query generation succeeds, but if the query expression attempts to use recursion or some other forbidden feature, it can still fail at run time.

The approach taken in C# for LINQ supports query-like operations as expressions, which are syntactic sugar for (quoted) operations on collection types, which in turn are transformed to SQL by LINQ implementations. Similarly, LINQ in F# 3.0 is implemented explicitly using quotation [19] and *computation expressions* [18].

The Links system [6] initially followed an ad hoc approach similar to Kleisli. Currently it uses a type-and-effect system [5, 14], where the "effects" characterize the environment in which a piece of code can be run (database, host language, or either). The programmer can require the typechecker to verify that an expression will generate exactly one query at run time, using the `query` keyword. This feature is important for obtaining predictable performance.

Other language designs that have been explored (not explicitly based on comprehensions) include defining ad hoc language extensions to include SQL or query-like syntax explicitly within the programming language, as in SML# [16], or using row types to embed SQL-like operators as a domain-specific embedded language, as in Ur [4].

1.2 Expressiveness and conservativity results

NRC allows writing queries that are not literally translatable to SQL, because they involve intermediate nested structures. Conservativity results for NRC mean that any *flat–flat* query expression (one that maps flat input data to flat output data) is equivalent to a query that only manipulates flat tables. Such results are important because flat–flat NRC queries can be translated easily to SQL.

Wong's proof of conservativity [22] was not the first proof of such a property, but in contrast to previous proofs based on semantic arguments [17], Wong's proof gave a straightforward rewriting algorithm for normalizing a flat–flat query to a form isomorphic to SQL; this idea formed the basis of the Kleisli system [23].

Since then, conservativity has been extended in several ways. Van den Bussche gave a proof of simulation of nested relational algebra by flat relational algebra [21], showing that any nested query can be simulated by n flat queries, where n is the number of collection types appearing in the result of the query. The proof is constructive and gives an explicit translation, but to our knowledge this approach has never been implemented or experimentally evaluated, nor extended to bags or lists.

Wong also considered sum types and simple (first-order) lambda abstraction, but his proof did not handle higher-order terms or allow sum types or functions as query results. Wong's normalization approach was extended to allow higher-order functions within the query by Cooper [5], and this approach has been refined in other work [3, 14]. More recently, Giorgidze et al. [9] have considered another representation for algebraic data types, and Grust and Ulrich proposed handling function values using defunctionalization [13]. These techniques do allow functions and algebraic data type values as query results.

1.3 Systems

Wong [23] introduced Kleisli, a system for data integration based on comprehension syntax. Kleisli was used to solve the twelve so-called 'impossible queries' identified by a US Department of En-

System	Language	Model	Sums?	Higher-order?	Nested?	Grouping?	Bounded?	Proof?
Kleisli [23]	Kleisli	Set, bag, list	0	-	-	0	0	+
Links [6]	Links	Bag	0	+	-	-	+	+
LINQ [15, 19]	C#, F#	Bag	-	0	-	-	-	-
Ferry (LINQ) [11]	C#, F#	List	0	0	+	+	-	-
Ferry (Links) [20]	Links	List	0	+	+	+	+	-
Database-Supported Haskell [8, 9]	Haskell	List	+	0	+	+	+	-
P-LINQ [3]	F#	Bag	-	+	-	+	-	-
T-LINQ [3]	F#	Bag	-	+	-	-	+	+
Links + Shredding (in progress)	Links	Bag	0	+	+	-	+	+

Table 1. Summary of comprehension-based systems for language-integrated query.

ergy Bioinformatics Summit. A related system called K2 became part of a popular biomedical data integration product [7].

Links [6] initially followed the Kleisli approach, and attempted to identify parts of programs that could be turned into queries automatically. Cooper [5] showed how to generalize this approach to handle queries that use higher-order functions in Links, and Lindley and Cheney [14] presented an alternative approach based on row types (which is the current approach adopted in Links). We subsequently [3] showed how to adapt this approach to LINQ as implemented in F#, using a quotation-based language design to embed queries instead of effects. The T-LINQ calculus in that paper provides strong guarantees but does not handle grouping and aggregation, while the P-LINQ library shows that our approach works well with LINQ in practice (including grouping and aggregation) but does not yet offer a formal guarantee.

Grust et al. [10] have introduced Ferry, an implementation of nested relational calculus that employs a *loop-lifting* technique originally developed for XQuery to LINQ. The Ferry approach uses a SQL:1999 query optimizer called Pathfinder [12] as a backend, and has been adapted to LINQ [11] and Links [20]. There is no published proof of correctness of either Pathfinder or Ferry's loop-lifting algorithm.

Table 1 summarizes the features of the different systems and characterizes their support for features such as set, bag, and list collections; higher-order queries; nested query results; grouping and aggregation; bounded-query guarantees; and availability of correctness proofs. As the table suggests, no one system combines all of the potentially desirable features. Among the systems, only Ferry provides support for nested query results, and no current system supports all collection types (set, bag, list), higher-order queries, nesting, and grouping.

2. Open problems and current work

We are interested in relating effect-based and quotation-based code, as investigated in a recent paper [2], and more generally, in compilation of the effect-based approach used in Links. We are currently investigating a *shredding* algorithm for handling nested data based on normalization; extending this approach to handle aggregation and grouping is an open problem. Proving correctness of Ferry's loop lifting, and formalizing and verifying Pathfinder, are also open problems. Measuring the usability of these approaches is also an interesting open problem. Finally, it may be of interest to adapt these techniques to cover other computational models, such as data parallelism, GPU programming or MapReduce instead of SQL.

References

[1] P. Buneman, S. Naqvi, V. Tannen, and L. Wong. Principles of programming with complex objects and collection types. *Theor. Comput. Sci.*, 149(1), 1995.

[2] J. Cheney, S. Lindley, G. Radanne, and P. Wadler. Effective quotation. In *PEPM*, 2014. To appear.

[3] J. Cheney, S. Lindley, and P. Wadler. A practical theory of language-integrated query. In *ICFP*, 2013.

[4] A. J. Chlipala. Ur: statically-typed metaprogramming with type-level record computation. In *PLDI*, 2010.

[5] E. Cooper. The script-writer's dream: How to write great SQL in your own language, and be sure it will succeed. In *DBPL*, 2009.

[6] E. Cooper, S. Lindley, P. Wadler, and J. Yallop. Links: web programming without tiers. In *FMCO*, 2007.

[7] S. B. Davidson, J. Crabtree, B. P. Brunk, J. Schug, V. Tannen, G. C. Overton, and C. J. Stoeckert, Jr. K2/Kleisli and GUS: Experiments in integrated access to genomic data sources. *IBM Syst. J.*, 40(2):512–531, Feb. 2001.

[8] G. Giorgidze, T. Grust, T. Schreiber, and J. Weijers. Haskell boards the Ferry - database-supported program execution for Haskell. In *IFL*, number 6647 in LNCS, pages 1–18. Springer-Verlag, 2010.

[9] G. Giorgidze, T. Grust, A. Ulrich, and J. Weijers. Algebraic data types for language-integrated queries. In *DDFP*, pages 5–10, 2013.

[10] T. Grust, M. Mayr, J. Rittinger, and T. Schreiber. Ferry: Database-supported program execution. In *SIGMOD*, June 2009.

[11] T. Grust, J. Rittinger, and T. Schreiber. Avalanche-safe LINQ compilation. *PVLDB*, 3(1), 2010.

[12] T. Grust, J. Rittinger, and J. Teubner. Pathfinder: XQuery off the relational shelf. *IEEE Data Eng. Bull.*, 31(4), 2008.

[13] T. Grust and A. Ulrich. First-class functions for first-order database engines. In *DBPL*, 2013. `http://arxiv.org/abs/1308.0158`.

[14] S. Lindley and J. Cheney. Row-based effect types for database integration. In *Proceedings of the 8th ACM SIGPLAN workshop on Types in language design and implementation*, TLDI '12, 2012.

[15] Microsoft. Query expressions (F# 3.0 documentation), 2013. `http://msdn.microsoft.com/en-us/library/vstudio/-hh225374.aspx`, accessed March 18, 2013.

[16] A. Ohori and K. Ueno. Making Standard ML a practical database programming language. In *ICFP*, pages 307–319. ACM, 2011.

[17] J. Paredaens and D. V. Gucht. Converting nested algebra expressions into flat algebra expressions. *ACM Trans. Database Syst.*, 17(1), 1992.

[18] T. Petricek and D. Syme. Syntax Matters: Writing abstract computations in F#. Pre-proceedings of TFP, 2012. `http://www.cl.cam.ac.uk/~tp322/drafts/notations.pdf`.

[19] D. Syme. Leveraging .NET meta-programming components from F#: integrated queries and interoperable heterogeneous execution. In *ML*, 2006.

[20] A. Ulrich. A Ferry-based query backend for the Links programming language. Master's thesis, University of Tübingen, 2011.

[21] J. Van den Bussche. Simulation of the nested relational algebra by the flat relational algebra, with an application to the complexity of evaluating powerset algebra expressions. *Theor. Comput. Sci.*, 254(1-2), 2001.

[22] L. Wong. Normal forms and conservative extension properties for query languages over collection types. *J. Comput. Syst. Sci.*, 52(3), 1996.

[23] L. Wong. Kleisli, a functional query system. *J. Funct. Program.*, 10(1), 2000.