JFP **28**, e8, 2 pages, 2018.
doi:10.1017/S0956796818000084

Special Issue on Programming Languages for Big Data Editorial

Ideas from programming languages play an important role in a range of advanced applications of databases, in database system implementation, distributed programming (MapReduce), streaming computation, and high-performance (GPU/multicore) computation. This creative research area is broadening into a subfield of data-centric computation. Although the interaction of databases and programming has a long history (the 16th biennial Database Programming Languages symposium was held in 2017), there has been a recent renewal of interest and broadening of programming language techniques for dealing with data from several quarters in the last few years, including workshops at Microsoft Research (RADICAL 2010), ICFP (XLDI 2012), POPL (DDFP 2013, DCM 2014) and a Dagstuhl Seminar on Programming Languages for Big Data (December 2014). This special issue recognises and encourages the publication of mature research contributions in this area.

The special issue received seven submissions, of which six were ultimately accepted. The topics range from functional and algebraic foundations of database query languages to distributed computation, and highlight the increasing diversity and maturity of programming languages techniques applied to data management.

Techniques for defining and optimising database queries using ideas from functional programming have a long history, e.g., in the Kleisli system by Wong reported in this *Journal* in 2000. In 'Capability-based Localization of Distributed and Heterogeneous Queries', Seco, Ferreira and Lourenço build on this approach by considering a language that permits defining computations that may involve several databases with different capabilities, and give *localisation* algorithms for partitioning the computation into queries that can each be run on a single database, and finally combining their results. Since there may be many ways to partition and optimise such queries, this paper potentially provides a foundation for future work on language-based optimisation of distributed queries.

Ideas from category theory have also long been recognised as a foundation for functional programming, and have had an impact on understanding of database query languages as well. In 'Algebraic Data Integration', Schulz and Wisnesky build upon a recent work by Spivak, Wisnesky and others regarding the use of categorical constructions (including *adjunctions*) as a basis for query languages over categorical data. Whereas a previous work has presented the basic ideas and discussed an implementation called FQL that can execute queries by translation to SQL, this paper develops applications of these ideas to *data integration*, the problem of combining data sources (often with different schemas or constraints).

Functional programming languages, such as Scala, have seen increasing use as a basis for distributed computation in frameworks such as Spark. In 'A Programming Model and Foundation for Lineage-Based Distributed Computation', Haller, Miller and Müller present a new model called *function passing* that provides a substrate for data-centric distributed systems in which failure recovery is supported by replaying function applications and deferred evaluation is used to avoid expensive data transfers. They provide a core language formalising the key ideas and prove a subject reduction result, and there is also an open-source implementation of their approach.

It is the rule rather than the exception that complex workflows in data-intensive research call on pipelines of pre- and post-processors, external code written in a variety of languages, or—more generally—an entire zoo of auxiliary computational black boxes. In 'Computation Semantics of the Functional Scientific Workflow Language Cuneiform', Brandt, Reisig and Leser develop the syntax, type system and semantics of *Cuneiform*, a functional language that can orchestrate such heterogeneous workflows. With Cuneiform's semantics established, the paper focuses on its interpretation and compilation, and discusses the parallel and distributed execution of workflows in the presence of black boxes. Cuneiform has already shown its utility in variety of domains, from bioinformatics to machine learning, and is ready to be downloaded and used today.

Pipelined query evaluation and loop fusion are the database and programming language sides of the same coin. This is a central thesis in 'Push vs. Pull-Based Loop Fusion in Query Engines' by Shaikhha, Dashti and Koch. Their article sheds light on this interesting parallel and then builds on *stream fusion* to derive a new kind of pull-based query engine that combines the advantages of well-known materialisation-avoiding query evaluation strategies. Their paper contains extensive experiments that provide guidance on when it is preferable to push or to pull.

Comprehension syntax has already repeatedly proven to provide an elegant, concise, and versatile language for the specification of queries. In his article 'An Algebra for Distributed Big Data Analytics', Fegaras uses *monoid comprehensions* as the foundation for MRQL, a language that expresses data-intensive computations over distributed sources at a much higher level than established programming frameworks for MapReduce. Beyond declarative query formulation, Fegaras shows that the monoid abstraction also facilitates the optimisation and execution of distributed data analysis tasks.

We thank the authors and referees of these articles for their efforts producing and reviewing these articles within the strict time limits imposed by the special issue publication constraints. We also gratefully acknowledge the support of the JFP editors-in-chief and editorial office.

James Cheney
Laboratory for Foundations of Computer Science, University of Edinburgh
jcheney@inf.ed.ac.uk

Torsten Grust
Department of Computer Science, University of Tübingen
torsten.grust@uni-tuebingen.de