The five articles in this special issue are the extended, journal versions of the papers first presented at the ACM SIGPLAN 2012 Workshop on Partial Evaluation and Program Manipulation. PEPM'12 took place in January 2012 in Philadelphia, Pennsylvania, USA. Out of 19 papers presented at PEPM 2012, the program committee selected six and invited their authors to submit an extended and improved version to the Special issue. Each submission was reviewed by three reviewers according to the rigorous journal standards. Five have been recommended for publication.

The PEPM Symposium/Workshop series is about the theory and practice of program transformation understood broadly, ranging from program manipulation such as partial evaluation, to program analyses in support of program manipulation, to treating programs as data objects (metaprogramming). PEPM focuses on techniques, supporting theory, tools, and applications of the analysis and manipulation of programs. PEPM specifically stresses that each technique or tool of program manipulation should have a clear, although perhaps informal, statement of desired properties, along with an argument how these properties could be achieved. The papers included in this special issue reflect the entire scope of PEPM, its interplay of theory and practice, and its stress on rigor and clarity.

The first article by Kazutaka Matsuda, Kazuhiro Inaba and Keisuke Nakano on *Polynomial-Time Inverse Computation for Accumulative Functions with Multiple Data Traversals* is directly about program manipulation. Specifically, the topic of the article is program inversion: transforming a program to 'run in reverse', enumerating the possible inputs to the original program that result in the given output. Not only does the transformed, inverted program always terminate for a broad class of functions, but it also runs in polynomial time with respect to the original output and the program size. This paper received PEPM 2012 Best Paper award, as chosen by the program committee.

Naoki Kobayashi, Kazutaka Matsuda, Ayumi Shinohara and Kazuya Yaguchi present a rare view of programs as data, namely programs as compressed data. Running the compressed data as a functional program recovers the original data. The authors further demonstrate that some program analysis and manipulation techniques can be applied to the program, having the effect of manipulating the compressed data without decompressing it. As in article by Matsuda et al., the authors not only present the compression and manipulation algorithms; they rigorously formulate the correctness property and prove that it holds.

The article on The Interaction of Contracts and Laziness by Markus Degen, Peter Thiemann, and Stefan Wehr is on the topic of contract monitoring: checking dynamically that functions satisfy their contracts as they are run. Given a function and a partial specification of its correctness (a contract), the goal is to transform the function in order to verify, at run-time, that the specification is satisfied. Such a transformation is not at all straightforward in lazy languages. The article defines and justifies the properties of the transformation, and defines what it means for such a transformation to be complete and to preserve meaning. The article rigorously demonstrates that, surprisingly, contract monitoring for lazy functional languages cannot simultaneously have both properties.

Most program analyses and transformations deal with a program as fully written. Rarely do we see rigorous analyses of methods to help write programs – of integrated development environments (IDE) and their user interfaces. Isao Sasano and Takumi Goto's *An Approach to Completing Variable Names for Implicitly Typed Functional Languages* is an exception. The authors investigate variable name completion, a popular feature in IDE. The name completion is guided by the type expected at the current point. Determining the expected type in a not-yet-completed expression is a challenge in implicitly typed functional languages, where the types are inferred only after the program is finished. Isao Sasano and Takumi Goto address this problem, by developing a theory of type inference on incomplete programs. They have prototyped their approach as an Emacs mode.

The final article showcases the practical aspect of PEPM. Tiark Rompf, Nada Amin, Adriaan Moors, Philipp Haller, and Martin Odersky describe the design and implementation of Scala-Virtualized: a set of Scala features that support flexible embeddings of domain-specific languages (DSL). Of particular interest is representing conditionals, pattern matching and other special forms of the embedded language in Scala syntax (but with possibly different semantics). The article should also appeal to the researchers in program generation and partial evaluation: the authors treat an embedded DSL program as a program generator. The paper was presented at PEPM as a tool demonstration. The article in this issue is a greatly expanded version, adding to the new large demonstration example the motivation and a thorough explanation of the virtualization technique.

We hope this issue is informative, and illustrative of PEPM.

Oleg Kiselyov	Julia Lawall	Simon Thompson
Monterey, CA, US	INRIA, FR	University of Kent, UK