November 2007

Adventures in Bidirectional Programming

Benjamin C. Pierce
University of Pennsylvania, bcpierce@cis.upenn.edu

Postprint version. Published in *Lecture Notes in Computer Science*, Volume 4855, FSTTCS: Foundations of Software Technology and Theoretical Computer Science, November 2007, pages 21-22.
Publisher URL: http://dx.doi.org/10.1007/978-3-540-77050-3_3

Adventures in Bidirectional Programming

Abstract

Most programs get used in just one direction, from input to output. But sometimes, having computed an output, we need to be able to *update* this output and then "calculate backwards" to find a correspondingly updated input. The problem of writing such *bidirectional transformations* — often called *lenses* — arises in applications across a multitude of domains and has been attacked from many perspectives [1–12, etc.]. See [13] for a detailed survey.

Comments

Postprint version. Published in *Lecture Notes in Computer Science*, Volume 4855, FSTTCS: Foundations of Software Technology and Theoretical Computer Science, November 2007, pages 21-22.
Publisher URL: http://dx.doi.org/10.1007/978-3-540-77050-3_3

Adventures in Bidirectional Programming

Benjamin C. Pierce

University of Pennsylvania

Most programs get used in just one direction, from input to output. But sometimes, having computed an output, we need to be able to *update* this output and then "calculate backwards" to find a correspondingly updated input. The problem of writing such *bidirectional transformations*—often called *lenses*—arises in applications across a multitude of domains and has been attacked from many perspectives [1–12, etc.]. See [13] for a detailed survey.

The Harmony project at the University of Pennsylvania is exploring a *linguistic* approach to bidirectional programming, designing domain-specific languages in which every expression simultaneously describes both parts of a lens. When read from left to right, it denotes an ordinary function that maps inputs to outputs. When read from right to left, it denotes an "update translator" that takes an input together with an updated output and produces a new input that reflects the update. These languages share some common elements with modern functional languages—in particular, they come with very expressive type systems. In other respects, they are rather novel and surprising.

We have designed, implemented, and applied bi-directional languages in three quite different domains: a language for bidirectional transformations on trees (such as XML documents), based on a collection of primitive bidirectional tree transformation operations and "bidirectionality-preserving" combining forms [13]; a language for bidirectional views of relational data, using bidirectionalized versions of the operators of relational algebra as primitives [14]; and, most recently, a language for bidirectional string transformations, with primitives based on standard notations for finite-state transduction and a type system based on regular expressions [15]. The string case is especially interesting, both in its own right and because it exposes a number of foundational issues common to all bidirectional programming languages in a simple and familiar setting.

This survey talk discusses several of these issues in depth and describes progress toward solutions.

References

1. Meertens, L.: Designing constraint maintainers for user interaction (1998) Manuscript.
2. Kennedy, A.J.: Functional pearl: Pickler combinators. Journal of Functional Programming **14**(6) (2004) 727–739
3. Benton, N.: Embedded interpreters. Journal of Functional Programming **15**(4) (2005) 503–542
4. Ramsey, N.: Embedding an interpreted language using higher-order functions and types. In: ACM SIGPLAN Workshop on Interpreters, Virtual Machines and Emulators (IVME), San Diego, CA. (2003) 6–14

5. Hu, Z., Mu, S.C., Takeichi, M.: A programmable editor for developing structured documents based on bi-directional transformations. In: Partial Evaluation and Program Manipulation (PEPM). (2004)
6. Brabrand, C., Møller, A., Schwartzbach, M.I.: Dual syntax for XML languages. In: Database Programming Languages (DBPL), Trondheim, Norway. Volume 3774 of Lecture Notes in Computer Science., Springer-Verlag (August 2005) 27–41
7. Kawanaka, S., Hosoya, H.: bixid: a bidirectional transformation language for XML. In: ACM SIGPLAN International Conference on Functional Programming (ICFP), Portland, Oregon. (2006) 201–214
8. Daly, M., Mandelbaum, Y., Walker, D., Fernández, M.F., Fisher, K., Gruber, R., Zheng, X.: PADS: An end-to-end system for processing ad hoc data. In: Proceedings of ACM SIGMOD International Conference on Management of Data, Chicago, IL. (2006) 727–729
9. Alimarine, A., Smetsers, S., van Weelden, A., van Eekelen, M., Plasmeijer, R.: There and back again: Arrows for invertible programming. In: ACM SIGPLAN Workshop on Haskell. (2005) 86–97
10. Stevens, P.: Bidirectional model transformations in QVT: Semantic issues and open questions. In: MODELS. (2007)
11. Bancilhon, F., Spyratos, N.: Update semantics of relational views. ACM Transactions on Database Systems **6**(4) (December 1981) 557–575
12. Gottlob, G., Paolini, P., Zicari, R.: Properties and update semantics of consistent views. ACM Transactions on Database Systems (TODS) **13**(4) (1988) 486–524
13. Foster, J.N., Greenwald, M.B., Moore, J.T., Pierce, B.C., Schmitt, A.: Combinators for bi-directional tree transformations: A linguistic approach to the view update problem. ACM Transactions on Programming Languages and Systems (3) (May 2007) Extended abstract in Principles of Programming Languages (POPL), 2005.
14. Bohannon, A., Vaughan, J.A., Pierce, B.C.: Relational lenses: A language for updateable views. In: Principles of Database Systems (PODS). (2006) Extended version available as University of Pennsylvania technical report MS-CIS-05-27.
15. Bohannon, A., Foster, J.N., Pierce, B.C., Pilkiewicz, A., Schmitt, A.: Boomerang: Resourceful lenses for string data. Technical report, Dept. of CIS University of Pennsylvania (July 2007) Available from `http://www.cis.upenn.edu/~jnfoster/boomerang-tr.pdf`.