Poster: Towards the Bidirectionalization of Spreadsheet Formulas

Nuno Macedo, Hugo Pacheco, Alcino Cunha, João P. Fernandes, Jácome Cunha, Jorge Mendes, José N. Oliveira
HASLab, INESC TEC & Universidade do Minho, Portugal
fatbit@di.uminho.pt

***Abstract*—Bidirectional transformations have application in a wide number of computer science domains. Spreadsheets, on the other hand, are widely used for developing business applications, but their formulas are unidirectional, in the sense that their result can not be edited and propagated back to their input cells. In this poster, we propose the bidirectionalization of such formulas, by recasting them as *lenses* [1], a successful bidirectional transformation framework with instantiations in various data domains relevant for this endeavor.**

***Keywords*-bidirectional transformations; spreadsheets.**

I. Introduction

Transforming data between two different domains is a typical problem in software engineering. Ideally such transformations should be *bidirectional*, so that changes in either domain can be propagated to the other one. Many of the existing bidirectional transformation frameworks are instantiations of the so called *lenses* [1], proposed as a solution to the well-known view-update problem: if we construct a view that abstracts information in a source domain, how can changes in the view be propagated back to the source?

Spreadsheets are one of the most popular tools for storing and processing information, by providing a simple programming model that makes initiation easier for non-proficient users. However, a spreadsheet formula is only seen as an unidirectional relation that denotes how the value of a target cell can be calculated from other cells in the spreadsheet.

In this poster, we envision the bidirectionalization of spreadsheet formulas by deploying them as lenses. Consider, as an example, the spreadsheet for the forecast of profit and loss depicted in Figure 1. In this spreadsheet each row represents a *product*, where the first column defines its identifier, the second its name and reference, and the next three present a forecast of costs and profits for such product (production cost, taxes cost and sale price). A view of this sheet defines a resumed report (by resorting to `VLOOKUP`) including the reference number of each product (extracted using `RIGHT`) and its total expected profit, calculated in the hidden auxiliary column `Aux` and processed with an `IF` statement: profits are simply presented, and losses are alerted with the string `LOSS`. Thus, a modification to a

This research work is part of the FATBIT project, financed with FEDER funds by COMPETE (*Programa Operacional Factores de Competitividade*) and with national funds by FCT (*Fundação para a Ciência e Tecnologia*) under project number FCOMP-01-0124-FEDER-020532. The first author is also sponsored by FCT under Grant SFRH/BD/69585/2010.

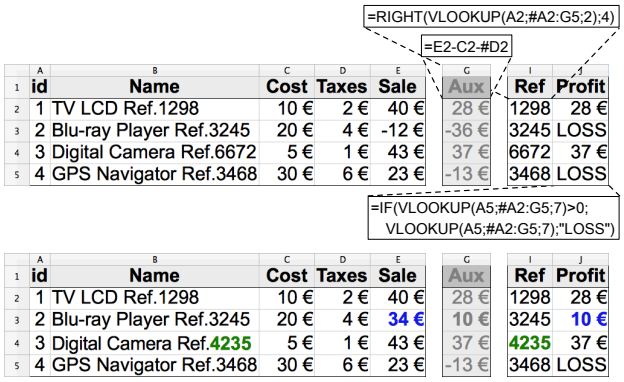

Figure 1. Example spreadsheet and update propagation.

cost will trigger the recalculation of the resulting profit, but the opposite is not possible: one cannot modify the profit margins and trigger a recalculation of the original costs.

To overcome such limitation, one could assign costs on a trial-and-error basis until the desired profit is obtained, what is extremely counter-productive, or invert the formula and redesign the spreadsheet so that the formula works in the opposite direction (for instance, with `Sale = Profit+Cost+Taxes`), although this would lead to the dual problem (we would only be able to update the profit), or even try to use a solver to deduce the input costs. A much better solution would be to make formulas bidirectional. With our proposal, the user would be able to update either the source or result cells, with the underlying lens automatically recomputing related values.

Bidirectionalization of spreadsheet formulas is not always trivial. For instance, a formula may refer to several cells, or multiple times the same cell; it may reference a cell that is itself a formula; it may contain complex primitives, like conditional expressions; and, in most cases, the inverse transformation will not be deterministic: this happens whenever the function defined by the formula is not injective (as is the general case for views). In the rest of this poster we discuss these issues and propose a possible bidirectionalization approach. Furthermore, the multi-disciplinary application of spreadsheets (combining features ranging from databases, arithmetic, functional programming and string processing) makes them an unique scenario, requiring the adoption of bidirectional transformation techniques previously developed for various data domains [2], [3], [1], [4].

II. Bidirectional Spreadsheet Formulas

In our approach, the subjects of bidirectionalization are spreadsheet formulas. Unlike other bidirectional languages, where an expression denotes a global transformation between two structures, our idea is to interpret each formula in a sheet locally as an independent lens. In this sense, likewise a change in a cell may prompt the recalculation of multiple formulas, a change to the result of a formula would be automatically propagated back to the respective source cells by the associated lens, and the updated sources could themselves trigger the recomputation of other formulas both in the forward or backward direction. By working on an *online setting* [5], where the system reacts immediately after each modification, the round-tripping properties of lenses (ensuring that after propagating a view update a forward transformation yields the same view) are instrumental to guarantee that update propagation is *convergent*, i.e., does not loop indefinitely and converges into a consistent state.

Standard spreadsheet systems include database-flavored primitives (like `VLOOKUP` and `HLOOKUP`), functions for the manipulation of strings (like `RIGHT` or `CONCATENATE`) or for arithmetic operations and logical combinators like `IF`. Seeing these combinators as lenses, their backward transformations must also receive the original source values in order to recover missing information. Consider, for example, the formula `RIGHT(A1,3)` that extracts the last 3 characters of cell `A1`. If its result is updated to `ple`, where `A1` contains the string `EXAMPLE`, the backward transformation would recover the prefix of the original string and update the source to `EXAMple`. To handle nested formula application (like `RIGHT(VLOOKUP(...))` in Figure 1) and sequential update propagation via intermediate cells (like column `Aux` in Figure 1), lens composition (see [1]) can be used.

In general, updates on formulas with multiple references (either to the same or to different cells) can be propagated in a variety of ways to different cells. In order to provide an intuitive control, we plan to require users to indicate explicitly which cells can be automatically updated by the bidirectional system, by marking such references with the special symbol `#`. As a consequence, formulas without any `#` marked cell would behave as standard unidirectional ones. In a real implementation, the handling of such marks would require a careful analysis of the formula dependencies to provide adequate update propagation. For example, the system should be able to reject formulas like `A+#A` or even `A+#B` with `B=A`, cases where it is impossible to update only one of the references, and provide informative feedback to users. Even in valid formulas, not all updates can be propagated back. For instance, in an `IF` conditional that either prints the character 'A' or 'B', updating the cell to any other value should raise an error. As such, for each formula, we should be able to dynamically calculate its exact range of values, and restrict the updates by signaling errors to users.

III. Application scenarios

In this section we explore possible application scenarios.

Forecast: The example from the introduction may represent a class of interesting applications for our framework. After analyzing the forecast, the user may wish to explore which parameters would achieve certain different profit margins, by editing the result of the formulas. In Figure 1, we updated a `LOSS` to a positive integer. This single update triggered the update of three combinators: the `IF`, the `VLOOKUP` and the addition on the `Aux` cell. Note that the change was only propagated to the sale price (as marked with `#`), since we assumed that costs are fixed.

Pivot tables: Pivot tables provide summaries for spreadsheets and define typical views. Clicking in any cell of a pivot table presents the set of data contributing to it, in the form of a new sheet; if one finds an error in such sheet, he has to go back to the original data set and edit it, thereupon recalculating the pivot table again. Our framework could enforce this connection between the new sheet and the original data. Moreover, the typical aggregation formulas used in pivot tables could be bidirectionalized by resorting to the previously presented techniques.

IV. Conclusion

In this poster we proposed the lifting of spreadsheet formulas to lenses, so that when results are edited some related input cells (marked by the user) are automatically recomputed to produce the desired results. Concerning deployment, our goal is to devise a technique that is conservative (it does not propagate updates unless explicitly required) and transparent to users. We plan to implement an OpenOffice plug-in, where users can edit both the formula (in the formula bar) and the current value of a cell (in the cell itself) in an intuitive way.

References

[1] J. Foster, M. Greenwald, J. Moore, B. Pierce, and A. Schmitt, "Combinators for bidirectional tree transformations: A linguistic approach to the view-update problem," *ACM T. Progr. Lang. Sys.*, vol. 29, no. 3, p. 17, 2007.

[2] A. Bohannon, B. Pierce, and J. Vaughan, "Relational lenses: a language for updatable views," in *PODS'06*. ACM, 2006, pp. 338–347.

[3] T. Yokoyama, H. Axelsen, and R. Glück, "Principles of a reversible programming language," in *CF'08*. ACM, 2008, pp. 43–54.

[4] A. Bohannon, J. Foster, B. Pierce, A. Pilkiewicz, and A. Schmitt, "Boomerang: resourceful lenses for string data," in *POPL'08*. ACM, 2008, pp. 407–419.

[5] Z. Hu, S.-C. Mu, and M. Takeichi, "A programmable editor for developing structured documents based on bidirectional transformations," *Higher Order and Symbolic Computation*, vol. 21, no. 1–2, pp. 89–118, 2008.