Automating State Management in Computational Notebooks

Stephen Macke
smacke@berkeley.edu
University of Illinois (UIUC) and UC Berkeley

Recently, computational notebooks have emerged as essential tools that enable scientists and engineers to perform exploratory data analysis with especially tight feedback. Jupyter [4] in particular has gained widespread popularity. With an estimated 4.7 million notebooks on GitHub as of March 2019, it has been called "data scientists' computational notebook of choice" [9]. Furthermore, its impact has been formally recognized by the ACM Software System award in 2018 [8], and continued shared interest has led to the emergence of the JupyterCon conference. All the evidence suggests that computational notebooks, and especially Jupyter, have cemented themselves as essential data tools that will be with us for years to come.

Despite their popularity, notebooks have a number of drawbacks that are well-documented in academic [5, 9] and industry [2, 3, 10] literature. While the specific complaints about notebook behavior vary, they all stem from the inherent difficulty in manual management of the global state that notebooks keep persistent in memory. This global state can be highly dependent on the order in which cells are run, making it difficult to rectify with code visible on screen. This issue is further exacerbated by the ability to reorder, rerun, edit, and delete notebook cells. However, existing approaches [5, 10] sacrifice flexible any-order execution semantics [6] of notebooks, prompting us to ask the question: can we have our cake (by reducing errors in notebooks) and eat it too (keeping existing notebook semantics)?

Leveraging Wisdom from the DB Community. To reduce errors and reproducibility problems in notebooks, we propose that, just as a relational database pushes responsibility of managing data integrity from application logic to a DBMS, we need ways to push management of notebook state from the brain of the user down into smarter notebook kernels. In doing so, we can enable a number of desirable properties toward making notebooks safer.

1. Atomicity of Cell Execution. We propose that notebooks should obey *transactional semantics* when cells are executed. For example, if a user forgets to define a variable or import a package, we should warn the user, instead of leaving the cell in a partially-executed state — particularly egregious if the cell is not idempotent, and the user goes to re-execute it after fixing any undefined references.

2. Idempotence of Cell Executions. Idempotent cell executions are desirable in their own right, since cells can be re-executed an arbitrary number of times. For example, if a user runs a cell that increments a counter variable, we should detect that this cell is not idempotent. In this way, the user can be warned if they attempt to execute it again, in case they did so by accident.

3. Obedience of Dataflow Constraints. Additionally, notebooks should be able to infer dataflow constraints without explicit guidance from users, based solely on the dependencies between variables and cells. In this way, if a dependency changes, the notebook can automatically determine which cells are unsafe to execute (because they contain references to non-updated variables) using program analysis techniques such as liveness analysis [1, 7].

Furthermore, we propose that even richer desired semantics can be inferred by examining the relationship between cell version and variable version, if only our notebook kernels captured variable version. For example, if some variable has a version more recent than the cell in which it was defined, this suggests that there is some initialization process for this variable; e.g., preprocessing a dataframe after reading a csv from disk. Enforcing this constraint for future uses of the aforementioned dataframe can help prevent errors that occur when the user forgets to execute the preprocessing code.

Our Proposed Approach. We propose that many of these properties can be enforced via a combination of *runtime tracing* (in order to infer desired program semantics without explicit indication from the user) along with *static program analysis* (in order to enforce these semantics). The advantage of inferring desired user semantics is that, unlike in previous systems wherein users must explicitly annotate their data dependencies [5], we can preserve the flexibility of existing notebook semantics. And what more natural way to infer these semantics than by tracing the actual code written by the user?

To enforce these semantics, we propose leveraging *state-aware program analysis* techniques. Based on metadata inferred during execution, we can, for example, be aware of which variables are resident in notebook memory, and we can combine this knowledge with liveness analysis [1] to detect cells that refer to uninitialized variables, and in doing so enforce atomicity.

We are currently building such a system, called NBSAFETY, for which we have already seen that such an approach can capture a subset of the dataflow constraints we indicated earlier [7]. We wager that this basic approach of capturing desired semantics in a *define-by-run* manner can help enable safer computational notebooks while maintaining their existing flexibility, and in doing so seamlessly augment modern data science with structure and rigor.

REFERENCES

[1] Alfred V Aho, Ravi Sethi, and Jeffrey D Ullman. 1986. Compilers, principles, techniques. *Addison wesley* 7, 8 (1986), 9.
[2] Joel Grus. 2018 (accessed June 26, 2020). *I Don't Like Notebooks (JupyterCon 2018 Talk).* https://t.ly/Wt3S.
[3] Or Hiltch. 2019 (accessed August 26, 2020). *Jupyter Notebook is the Cancer of ML Engineering.* https://medium.com/@_orcaman/jupyter-notebook-is-the-cancer-of-ml-engineering-70b98685ee71.
[4] Thomas Kluyver et al. 2016. Jupyter Notebooks-a publishing format for reproducible computational workflows.. In *ELPUB*. 87–90.
[5] David Koop and Jay Patel. 2017. Dataflow notebooks: encoding and tracking dependencies of cells. In *9th {USENIX} Workshop on the Theory and Practice of Provenance (TaPP 2017)*.
[6] Sam Lau, Ian Drosos, Julia M. Markel, and Philip J. Guo. 2020. The Design Space of Computational Notebooks: An Analysis of 60 Systems in Academia and Industry. In *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC) (VL/HCC '20)*.
[7] S Macke, H Gong, D Lee, A Head, D Xin, and A Parameswaran. 2020. *Fine-Grained Lineage for Safer Notebook Interactions.* Technical Report. Available at: https://smacke.net/papers/nbsafety.pdf.
[8] Jim Ormond. 2018 (accessed June 26, 2020). *ACM Recognizes Innovators Who Have Shaped the Digital Revolution.* https://awards.acm.org/binaries/content/assets/press-releases/2018/may/technical-awards-2017.pdf.
[9] Jeffrey M Perkel. 2018. Why Jupyter is data scientists' computational notebook of choice. *Nature* 563, 7732 (2018), 145–147.
[10] Kevin Zielnicki. 2017 (accessed July 5, 2020). *Nodebook.* https://multithreaded.stitchfix.com/blog/2017/07/26/nodebook/.

This article is published under a Creative Commons Attribution License (http://creativecommons.org/licenses/by/3.0/), which permits distribution and reproduction in any medium as well as allowing derivative works, provided that you attribute the original work to the author(s) and CIDR 2021. 11th Annual Conference on Innovative Data Systems Research (CIDR '21), January 10-13, 2020, Chaminade, USA.