Restoring Reproducibility of Jupyter Notebooks

Jiawei Wang
Faculty of Information Technology, Monash University, Australia

Tzu-yang Kuo
Hong Kong University of Science and Technology, Hong Kong

Li Li
Faculty of Information Technology, Monash University, Australia

Andreas Zeller
CISPA Helmholtz Center for Information Security, Germany

ABSTRACT

Jupyter notebooks—documents that contain live code, equations, visualizations, and narrative text—now are among the most popular means to compute, present, discuss and disseminate scientific findings. In principle, Jupyter notebooks should easily allow to reproduce and extend scientific computations and their findings; but in practice, this is not the case. The individual code cells in Jupyter notebooks can be executed *in any order,* with identifier usages preceding their definitions and results preceding their computations. In a sample of 936 published notebooks that would be executable in principle, we found that 73% of them would not be reproducible with straightforward approaches, requiring humans to infer (and often guess) the order in which the authors created the cells.

In this paper, we present an approach to (1) automatically satisfy dependencies between code cells to reconstruct possible execution orders of the cells; and (2) instrument code cells to mitigate the impact of non-reproducible statements (i.e., random functions) in Jupyter notebooks. Our *Osiris* prototype takes a notebook as input and outputs the possible execution schemes that reproduce the exact notebook results. In our sample, Osiris was able to reconstruct such schemes for 82.23% of all executable notebooks, which has more than three times better than the state-of-the-art; the resulting reordered code is valid program code and thus available for further testing and analysis.

CCS CONCEPTS

• **Software and its engineering → Software verification and validation**;

KEYWORDS

Python, Jupyter Notebooks, Reproducibility, Osiris

ACM Reference Format:
Jiawei Wang, Tzu-yang Kuo, Li Li, and Andreas Zeller. 2020. Restoring Reproducibility of Jupyter Notebooks. In *42nd International Conference on Software Engineering Companion (ICSE '20 Companion), October 5–11, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3377812.3390803

1 INTRODUCTION

Jupyter notebooks—documents that contain live code, equations, visualizations, and narrative text—have become the most widely used system for interactive literate programming. They are being used to compute, present, discuss and disseminate scientific findings; and have emerged as the de facto standard for data scientists to easily record and understand data analyses [3, 6]. In September 2018, more than 2.5 million Jupyter repositories were stored on GitHub—10 times more than in 2015 [4].

One of the promises of Jupyter notebooks is that they should make scientific findings *reproducible*—that is, readers should be able to reconstruct and assess the path from raw source data to abstractions and findings, as presented in the notebook [2]. Unfortunately, this is rarely the case. Published notebooks suffer from lack of data, from lack of modules, from lack of metadata indicating tool and library versions, or bad packaging [1]. But even if all of this is given, only a small fraction of notebooks can be faithfully reproduced.

Why is that so? A central feature of Jupyter notebooks is that the individual *cells* they are made of can be executed interactively *in any order.* The language interpreter (typically Python) will execute the code in the cell as soon as a user "runs" it. While Jupyter provides a "run all cells" feature that runs all cells starting from the topmost one, authors do not need to ensure that this results in meaningful execution order. It is not uncommon that notebooks output and present a result at the very beginning, followed by the code that actually produces the result, making the notebook more akin to an article than a conventional program. The interactive nature of notebooks makes all of this possible.

Given how many scientific results now are being produced using Jupyter notebooks, we feel it is the time for the program analysis and testing community to make the best of their approaches available to notebook authors [7]. But with 75% of valid notebooks (revealed by Pimentel et al. [4]) not even running without errors, this means that the majority of notebooks are actually inaccessible for any automated testing and analysis tool.

In this paper, we present an automatic approach to make Jupyter notebooks reproducible, and in consequence, available for analysis and testing. Our approach automatically identifies and satisfies dependencies between Jupyter notebook cells, reconstructing the possible execution orders that reproduce the exact notebook results without errors. The resulting ordered code can thus be subject to testing and analysis; our approach thus forms a necessary prerequisite for further analysis of notebook code. If a given notebook cannot be reproduced, our Osiris prototype provides detailed debugging messages explaining why reproducibility is not achievable.

Experimental results show that our approach is effective: In our sample, Osiris was able to reproduce 82.23% of executable notebooks, which is a large improvement over the 16.7% listed in state of the art [4].

2 ROOT CAUSES OF NON-REPRODUCIBILITY

Through a thorough manual investigation on nearly 1,000 Jupyter notebooks, we have observed the following root causes that may cause Jupyter notebooks non-reproducible.

R1: Randomness. Many scientific computing programs require random functions for sampling from Gaussian distributions or data shuffling. They produce different results after each execution (if no seed is given), making it hard to determine if the results can be reproduced.

R2: Time and Date. Time functions are recurrently used by notebook authors to achieve some specific functions such as evaluating time efficiency, logging for SQL operations, etc. Since time changes continuously, the outputs of the time function also vary every time, making it hard to ascertain the reproducibility of the code.

R3: Plots. We see a considerable number of notebooks that cannot be reproduced because of differences in plotted images. Indeed, in some cases, images are generated based on input data or random numbers that cannot be ensured to remain the same each time when the notebooks are executed.

R4: External Inputs. Jupyter notebooks may rely on external inputs (data fetched from web servers such as web crawler demonstration) to execute. However, the external inputs are subject to change (such as URL decay [5]), causing inconsistencies between the reproduced results and the recorded original results.

R5: Floating Point Numbers. In notebooks, floating point numbers may be printed differently depending on the running machines, or the targeted Python versions. Therefore, the reproduced results (relevant to floating point numbers) might be different from the original ones.

R6: Container Traversal. In Python, the order in which sets and dictionaries are traversed is not fixed. Hence, this order may differ across execution environments, causing differing cell outputs.

R7: Execution Environment. Notebooks may access the execution environment information (e.g., number of CPUs, Python package versions, the memory location of variables, etc.) that is usually specific to each setting and hence is different from one another. Moreover, in different execution environments, the same data might be printed in different formats. All these differences will impact the reproducibility of notebooks.

R8: Inappropriate execution order of cells. Apart from the errors raised by execution environments, we also observe a significant number of notebooks that fail to be executed due to poor code quality, e.g., containing *name errors* (undefined variables), *key error* (key not found in dictionaries), *syntax errors,* etc. These errors are yielded because the execution order of cells is inappropriate (e.g., a variable is used before its definition).

3 APPROACH AND EXPERIMENTS

After identifying the root causes of non-reproducibility, we design and implement a prototype tool called Osiris to restore the reproducibility of Jupyter notebooks. Osiris adopts different strategies (i.e., match and execution strategies) to resolve the aforementioned root causes of non-reproducibility, attempting to maximize the execution and reproducibility of notebooks. Osiris takes as input a Jupyter notebook and outputs the possible execution schemes that reproduce the exact notebook results. If Osiris fails to reproduce the notebook, it will highlight the location of failures (i.e., non-reproducible parts) that could be useful for understanding the root causes of non-reproducibility of Jupyter notebooks.

To evaluate the efficiency of Osiris, we launch Osiris on a set of randomly selected Jupyter notebooks, among which 1,435 notebooks can be successfully executed. Among these executable notebooks, 1,180 of them have been shown reproducible, giving a reproducibility rate of 82.23%, which has more than tripled the rate of the state-of-the-art and significantly increased from the results by simply executed the notebooks following their recorded execution orders. This result experimentally shows that the strategies introduced in Osiris are useful for restoring the reproducibility of Jupyter notebooks.

4 CONCLUSION

Motivated by the low reproducible rate of Jupyter notebooks reported by the state-of-the-art, we present to the SE community the first work to automatically restore reproducibility of Jupyter Notebooks. After conducting an empirical study to observe the root causes that make Jupyter notebooks non-reproducible, we designed and implemented a prototype called Osiris to resolve the observed root causes, aiming to explore all the possible execution schemes that reproduce the exact notebook results. In our evaluation, we show that our prototype tool is effective in restoring the reproducibility of Jupyter notebooks, achieving a significant improvement over the state-of-the-art. Notably that our approach enables further static and dynamic analyses, which can be used for testing, empirical studies, automatic repair techniques, and more.

REFERENCES

[1] Adam Brinckman, Kyle Chard, Niall Gaffney, Mihael Hategan, Matthew B Jones, Kacper Kowalik, Sivakumar Kulasekaran, Bertram Ludäscher, Bryce D Mecum, Jarek Nabrzyski, et al. 2019. Computing environments for reproducibility: Capturing the "Whole Tale". *Future Generation Computer Systems* 94 (2019), 854–867.

[2] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian E Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica B Hamrick, Jason Grout, Sylvain Corlay, et al. 2016. Jupyter Notebooks-a publishing format for reproducible computational workflows.. In *ELPUB*. 87–90.

[3] Jeffrey M. Perkel. 2018. Why Jupyter is data scientists' computational notebook of choice. *Nature news* 563 (2018), 145–146.

[4] João Felipe Pimentel, Leonardo Murta, Vanessa Braganholo, and Juliana Freire. 2019. A large-scale study about quality and reproducibility of jupyter notebooks. In *Proceedings of the 16th International Conference on Mining Software Repositories*. IEEE Press, 507–517.

[5] Diomidis Spinellis. 2003. The decay and failures of web references. *Commun. ACM* 46, 1 (2003), 71–77.

[6] Dan Toomey. 2017. *Jupyter for data science: Exploratory analysis, statistical modeling, machine learning, and data visualization with Jupyter.* Packt Publishing Ltd.

[7] Jiawei Wang, Li Li, and Andreas Zeller. 2020. Better Code, Better Sharing:On the Need of Analyzing Jupyter Notebooks. In *ICSE-NEIR 2020*.