Yin & Yang: Demonstrating Complementary Provenance from noWorkflow & YesWorkflow

João Felipe Pimentel[⋆] Saumen Dey[†] Timothy McPhillips[‡] Khalid Belhajjame[§]
David Koop[¶] Leonardo Murta[⋆] Vanessa Braganholo[⋆] Bertram Ludäscher[‡]

Abstract. The noWorkflow and YesWorkflow toolkits both enable researchers to capture, store, query, and visualize the provenance of results produced by scripts that process scientific data. noWorkflow captures prospective provenance representing the program structure of Python scripts, and retrospective provenance representing key events observed during script execution. YesWorkflow captures prospective provenance declared through annotations in the comments of scripts, and supports key retrospective provenance queries by observing what files were used or produced by the script. We demonstrate how combining complementary information gathered by noWorkflow and YesWorkflow enables provenance queries and data lineage visualizations neither tool can provide on its own.

1 Introduction

Methods for harvesting provenance information from scripts and runs of scripts have been of great recent interest to the provenance research community, and the resulting tools have received increasing attention from users of scripting languages in the natural sciences. Some of these approaches are language-specific, e.g., noWorkflow[1] [4,5] (Python) and RDataTracker [2] (R scripts), while others are language-independent, e.g., YesWorkflow[2] [3] and LLVM/SPADE [7]. Using such tools often entails annotating the scripts [2,3], monitoring executing scripts as they run [7,4], or both.

Approaches that do not require annotation, such as noWorkflow (NW), rely on the structure of the code itself to build prospective and retrospective provenance graphs. NW includes the actual function and variable names in the prospective provenance records, and it depends on records of run-time function calls to capture the retrospective provenance of script outputs. Consequently, the less meaningful variable and function names are in a script, the less clear the provenance query results and visualizations will be to scientists using the script. noWorkflow thus excels where Python programs are engineered for maintainablity, testability, code reuse, and long-term user support.

YesWorkflow (YW) is an example of a tool that largely ignores the code portions of a script, and instead depends on script authors (or users) adding annotations via comments in scripts. YW annotations declare the scientifically significant steps implemented by code blocks in a script, and the routes of dataflow between these steps. Annotations optionally assign meaningful names to actual (often obscurely named) code-level entities. Consequently, YesWorkflow users need not rename variables, move code

[⋆] Universidade Federal Fluminense, Brazil; [†]UC Davis; [‡]University of Illinois, Urbana-Champaign; [§]Université Paris-Dauphine, France; [¶]University of Massachusetts, Dartmouth.

[1] For "**n**ot **o**nly Workflow", emphasizing that scripts need provenance tracking, too.

[2] Which can be read as "Yes, scripts can be workflows, too!"

into functions, or otherwise refactor scripts that already have been used to produce results (research transparency requires disclosure of the scripts actually used). YW users can capture provenance from a working script without incurring the regression testing costs that refactoring entails. YW thus provides benefits even when scripts are written rapidly in the course of competitive, time-critical research, and when researchers employ scripts that they do not intend to maintain further or to distribute and support.

Given the contrasting aims of noWorkflow and YesWorkflow and the differences in the approaches they take, it is not surprising that each supports queries and visualizations that the other cannot support on its own [1]. Here we show that there are provenance artifacts of great interest to researchers that only a combination of YW and NW provenance can produce. Achieving this combination requires mapping between common entities in both provenance models, and jointly querying the provenance information represented by each system. We refer to the joint provenance model, the system-spanning queries, and the resulting visualizations collectively as YW*NW.

2 Example Queries: noWorkflow, YesWorkflow, and YW*NW

We use the Python script described by McPhillips et al. [3] to demonstrate the kinds of provenance queries NW, YW, and the combination of both support. This script simulates acquisition of diffraction images during macromolecular X-ray crystallography experiments involving multiple samples. The script reads previously measured data quality statistics for each sample from an input spreadsheet; rejects samples that do not meet a minimum quality criterion; and for each accepted sample produces raw and corrected diffraction images according to a data collection strategy that depends on properties of the samples. Although the script only simulates data collection, the order of task execution, the sequence of data production events, and the resulting pattern of dependencies between input, intermediate, and final data items closely mimic those of a real experiment [8]. Queries that probe these dependencies are therefore illustrative of meaningful uses for provenance information. The complete script, marked up with YW annotations, is available on GitHub [6]; a more complete explanation is provided in [3].

noWorkflow. Examples of **prospective** provenance queries of this script that NW supports include: *What functions does the top-level function call? Are any functions defined in the script not called by the top-level function?*

NW can answer **retrospective** provenance queries about runs of this script, such as: *What values did the variable* rejected_sample *take during writes to files referred to by the* rejection_log *variable? What files were written during calls to the* transform_image *function? How many files were written while the* accepted_sample *variable had the value* DRT240*? What variables carry values returned by the* calculate_strategy *function to calls to the* collect_next_image *function? What parameters to the top-level function can effect the results returned by calls to* calculate_strategy*?*

NW also can answer queries about the execution context: *Which user executed the script? What version of Python was used?*

YesWorkflow. YW provenance queries refer to annotated code blocks (workflow steps) rather than to Python functions, and to data names declared via YW annotations instead of to Python variables. Queries of **prospective** provenance supported by YW include:

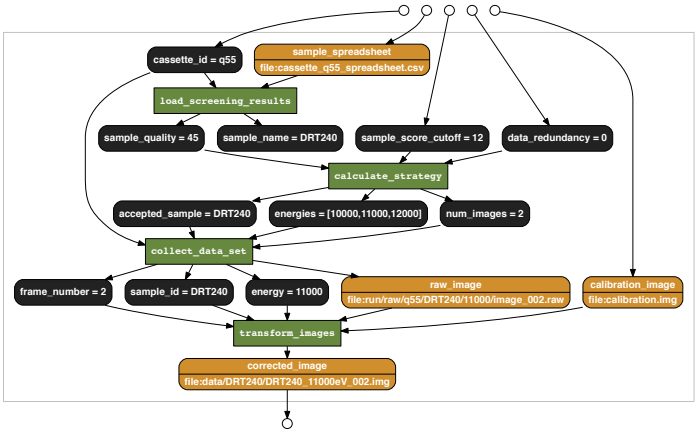

Fig. 1. Hybrid of YW prospective provenance and NW retrospective provenance: nodes and edges comprise the subgraph of the YW model of the script upstream of a single corrected_image; values in nodes are extracted from the NW runtime records of corresponding variable values leading to a particular image.

What are the names of steps that comprise the top-level workflow implemented by the script? What data is output by the collect_data_set *step? What code blocks provide input directly to that step? What data is* corrected_image *(in)directly derived from?*

YW can also answer some **retrospective** provenance queries [3], including: *What samples did the run of the script collect images from? What energies were used during collection of images from sample* DRT240*? Where is the raw image from which corrected image* run/data/DRT322/DRT322_10000eV_001.img *is derived? Are there any raw images for which there are* ***no*** *corresponding corrected images?*

Querying the Combined YW*NW Provenance. Queries that must be answered by combining NW and YW provenance generally involve references both to Python functions or variables *and* to code blocks or data declared via YesWorkflow annotations. Examples include: *Can the* sample_id *output of the* collect_data_set *step ever produce values other than those provided via the* accepted_sample *input to this step? What Python functions may be called as part of the* calculate_strategy *step? What was the set of* energies *produced by the* compute_strategy *step for sample* DRT322?

As these queries demonstrate, the combination of NW and YW provenance enables code-level entities such as Python functions and variables to be queried in terms of data and workflow steps meaningful to the user (and vice versa). Such queries are useful for understanding runs of the script in ways that neither NW nor YW enable on their own. Generalizing these queries yield meaningful visualizations of the full lineage of any product of the script. Consider the hybrid YW*NW provenance graph in Fig. 1, showing the lineage of a specific output image. This lineage graph can be constructed as a subgraph of the original YW model [3] (restricted to predecessors nodes upstream of the corrected_image result node), which is then augmented with NW retrospective provenance; see [6] for details and the YW*NW integration queries.

Because the questions scientists have about runs of scripts often can be answered in terms of lineages of data products, YW*NW queries and visualizations promise to

be of great value to researchers. Moreover, using noWorkflow and YesWorkflow jointly does not entail the major adaptations to code often needed to run existing software in scientific workflow management systems. Indeed, YW*NW provides many benefits of provenance management without requiring working code to be refactored at all.

3 Demonstration

In our demonstration we will highlight the benefits of harvesting, querying, and visualizing provenance with noWorkflow in conjunction with YesWorkflow. Starting with a directory containing just the example script and input files, we will (1) highlight how YW annotations can be visualized as prospective provenance using YesWorkflow; (2) run the script using noWorkflow and relate the resulting data file names and locations to the YW prospective provenance; (3) query the script and its outputs using noWorkflow and YesWorkflow separately to illustrate what each tool can do on its own; and (4) execute joint YW*NW queries that determine the lineage of a single data product and produce visualizations analogous to the one in Figure 1.

A companion GitHub repository for this demonstration is available, along with an expanded version of this short demo description [6]. The repository includes the data collection script discussed above; the files produced by a run of this script; the provenance information produced by noWorkflow and YesWorkflow; and helper scripts for running the queries mentioned above and for producing Figure 1. noWorkflow and YesWorkflow themselves are both available on GitHub and can easily be installed.

References

1. Dey, S., Belhajjame, K., Koop, D., Raul, M., Ludäscher, B.: Linking Prospective and Retrospective Provenance in Scripts. In: Theory and Practice of Provenance (TaPP) (2015)
2. Lerner, B., Boose, E.: RDataTracker: Collecting Provenance in an Interactive Scripting Environment. In: Theory and Practice of Provenance (TaPP). Cologne, Germany (2014)
3. McPhillips, T., Bowers, S., Belhajjame, K., Ludäscher, B.: Retrospective Provenance Without a Runtime Provenance Recorder. In: Theory and Practice of Provenance (TaPP) (2015)
4. Murta, L., Braganholo, V., Chirigati, F., Koop, D., Freire, J.: noWorkflow: Capturing and Analyzing Provenance of Scripts. In: Intl. Provenance and Annotation Workshop (IPAW). pp. 71–83. Cologne, Germany (2014)
5. Pimentel, J., Freire, J., Murta, L., Braganholo, V.: Fine-grained Provenance Collection over Scripts Through Program Slicing. In: Intl. Provenance and Annotation Workshop (IPAW). Washington D.C. (2016)
6. Pimentel, J.F., Dey, S., McPhillips, T., Belhajjame, K., Koop, D., Murta, L., Braganholo, V., Ludäscher, B.: Yin & Yang: Demonstrating Complementary Provenance from noWorkflow & YesWorkflow. Technical Report & Demo github.com/gems-uff/yin-yang-demo (2016)
7. Tariq, D., Ali, M., Gehani, A.: Towards Automated Collection of Application-level Data Provenance. In: Theory and Practice of Provenance (TaPP) (2012)
8. Tsai, Y., McPhillips, S.E., González, A., McPhillips, T.M., Zinn, D., Cohen, A.E., Feese, M.D., Bushnell, D., Tiefenbrunn, T., Stout, C., Ludäscher, B., Hedman, B., Hodgson, K.O., Soltis, S.M.: AutoDrug: fully automated macromolecular crystallography workflows for fragment-based drug discovery. Acta Crystallographica Section D: Biological Crystallography 69(5), 796–803 (2013)