Revealing the Detailed Lineage of Script Outputs Using Hybrid Provenance

Qian Zhang
University of Illinois at Urbana-Champaign

Yang Cao
University of Illinois at Urbana-Champaign

Qiwen Wang
University of Illinois at Urbana-Champaign

Duc Vu
University of Illinois at Chicago

Priyaa Thavasimani
Newcastle University

Timothy McPhillips
University of Illinois at Urbana-Champaign

Paolo Missier
Newcastle University

Peter Slaughter
University of California, Santa Barbara

Christopher Jones
University of California, Santa Barbara

Matthew B. Jones
University of California, Santa Barbara

Bertram Ludäscher
University of Illinois at Urbana-Champaign

Abstract

We illustrate how combining retrospective and prospective provenance can yield scientifically meaningful *hybrid provenance* representations of the computational histories of data produced during a script run. We use scripts from multiple disciplines (astrophysics, climate science, biodiversity data curation, and social network analysis), implemented in Python, R, and MATLAB, to highlight the usefulness of diverse forms of *retrospective* provenance when coupled with *prospective* provenance. Users provide prospective provenance, i.e., the conceptual workflows latent in scripts, via simple YesWorkflow annotations, embedded as script comments. Runtime observables can be linked to prospective provenance via relational views and queries. These observables could be found hidden in filenames or folder structures, be recorded in log files, or they can be automatically captured using tools such as noWorkflow or the DataONE RunManagers. The YesWorkflow toolkit, example scripts, and demonstration code are available via an open source repository.

Received 1 February 2017 ~ *Revision received* 24 April 2018 ~ *Accepted* 24 April 2018

Correspondence should be addressed to Qian Zhang, School of Information Sciences (iSchool), UIUC, 112 LIS,501 E. Daniel Street, Champaign IL 61820. Email: zhangqian06@gmail.com

An earlier version of this paper was presented at the 12th International Digital Curation Conference.

The *International Journal of Digital Curation* is an international journal committed to scholarly excellence and dedicated to the advancement of digital curation across a wide range of sectors. The IJDC is published by the University of Edinburgh on behalf of the Digital Curation Centre. ISSN: 1746-8256. URL: http://www.ijdc.net/

Copyright rests with the authors. This work is released under a Creative Commons Attribution Licence, version 4.0. For details please see https://creativecommons.org/licenses/by/4.0/

Introduction

Scripts are widely used to automate scientific workflows. Provenance support for script-based workflows is expected to promote interpretation, openness, and reproducibility of compute- and data-intensive studies (Freire, Fuhr and Rauber, 2016; Freire, Koop, Chirigati and Silva, 2014; Gadrud, 2013; Tilmes, Yesha and Halem, 2010). However, a gap often exists between the provenance that comes with a published dataset and the promised science-level explanations (Freire, Fuhr and Rauber, 2016). Part of the problem is that a retrospective provenance record (e.g., the events observed during the execution of a script on a particular input dataset) is only one of the elements needed to elucidate the process occurring during a computational experiment. Equally important is the general specification of the overall workflow. Following the nomenclature proposed by Zhao, Wilde and Foster (2006) and adopted in McPhillips et al. (2015b), we refer to the latter as *prospective provenance*, to distinguish it from (but also relate it to) what is commonly called *retrospective provenance*. Observables that contribute to retrospective provenance include the location, name, and contents of files read or written by a script, data created or updated during execution, parameter settings, and system-level information such as operating system and software versions, timestamps, etc. Prospective provenance, on the other hand, describes a conceptual high-level overview of the process used in a study. Computational workflow systems such as Kepler, Taverna, Pegasus, Restflow, or Vistrails (Ludäscher et al., 2006; Oinn et al., 2004; Deelman et al., 2005; Tsai et al., 2013; Bavoil et al., 2005) provide the abstraction form of a directed graph, where computational steps are connected via dataflow channels. For script-based workflows, one could consider the script itself a form of prospective provenance, albeit one in which the underlying conceptual view of the dataflow structure may be obscured by details of the implementation. The motivation for developing YesWorkflow (YW) has been to allow users easily to expose high-level workflow models that are latent in scripts, using simple user annotations embedded as script comments (McPhillips et al., 2015b). We show here that the workflow models declared by users and extracted via the YW toolkit can be augmented and enriched by retrospective provenance observables, yielding various forms of hybrid provenance (McPhillips et al., 2015a; Pimentel et al., 2016). By *hybrid provenance* we mean a combination of prospective and retrospective provenance, where fragments of workflow execution traces with structures provided by the user-declared YW models (prospective provenance) are filled in with script execution details from one or more sources of runtime observables (retrospective provenance), as shown in Figure 1.

Building on prior work (McPhillips et al., 2015b; McPhillips et al., 2015a; Pimentel et al., 2016; Cao et al., 2016a; Murta et al., 2014; Cao et al., 2016b; Bowers et al., 2012; Cuevas-Vicenttin et al., 2015), we demonstrate how multiple hybrid provenance representations and products can be derived from a combination of prospective and retrospective provenance sources. We argue that the resulting hybrid views, queries, and visualizations are useful not only for documenting and explaining script-based scientific workflows to others, but also for enabling researchers to query and explore the derivation history of the data products yielded by script execution while the broader computational studies are still underway. The techniques and examples discussed below are available as executable demonstrations via an open source repository (Zhang et al., 2017).

Prospective and Retrospective Provenance Elements

We first describe the tools we use to capture prospective and retrospective provenance, respectively, followed by the approaches to combine them to enable hybrid provenance query capabilities.

Modelling Scripts as Workflows with YW Annotations

YesWorkflow (YW) (McPhillips et al., 2015a; 2015b) aims to provide researchers who develop script-based workflows with many benefits of scientific workflow systems. Script authors can embed YW annotations (`@BEGIN`, `@END`, `@IN`, and `@OUT`) within the comments of their scripts, thereby declaring relevant computational steps and the dataflow between them. The resulting workflow model provides a high-level prospective provenance graph that can be visualized and queried to reveal dataflow dependencies (Figure 1, left). Since prospective provenance is created via user-provided YW annotations, the appropriate level of modelling detail is controlled by the users themselves.

Reconstructing Provenance with YW-Recon

The YesWorkflow model of a script can be augmented with runtime observations to yield hybrid provenance information. Some of the retrospective provenance from script runs can easily be reconstructed using resource URI template declarations (`@URI`): provided that users name and organize their files, folders, and other resources systematically, YW can use declared URI templates to discover the actual files that were read or written at runtime (McPhillips et al., 2015a). For example, a URI template such as the following:

```
@OUT Image @URI file:run/raw/{cassetteID}/{sampleID}/e{energy}/image_{no}.raw
```

denotes that raw image files are found within subfolders of `run/raw/`, relative to the folder in which the script was executed, and are organized first by cassette ID, then by sample ID, beam energy, and finally by image number. After script execution, by finding and matching the file paths to actual resources using the URI templates, YW can link the conceptual workflow entities – the templates that declare the layout of input and output resources on the file systems – with observations made about the script run, in this case the actual folders and files created during the run.

In addition, or as an alternative to the use of URI templates, a script author can employ user-defined *log files* to capture runtime provenance observables at any level of granularity that is required to support the desired provenance queries. Using log files written by a script, YW can then harvest fine-grained retrospective information, e.g., at the level of single record within a file, or specific field(s) within a record. Because the structures of log file entries are declared via `@LOG` template statement, the content of log entries can be chosen to maximize human readability of logs, while at the same time providing YW with machine-readable parsed information. Similar to URI templates, log templates are associated with other (here: `@OUT`) YW annotations, cf. Figure 6a. Moreover, an output resource name can be associated with multiple log entry templates. Each log entry template may include one or more template variables that distinguish multiple files (or other data sources) written by a script in a particular code block. As with URI template variables, log template variables are enclosed in curly braces. Following a script run, YW will use these log templates to discover the actual variable values written during the run and harvest the information from the log entries, cf. Figure 6b. The variables in the templates reveal runtime values of other variables annotated with `@in` or `@out` in the script. In this way, YW is able to support reports of the results of a script execution at various levels of granularity, e.g., at the script-, data-, file-, record-, field-, or function-level, provided prospective provenance declarations have been provided for each of those levels.

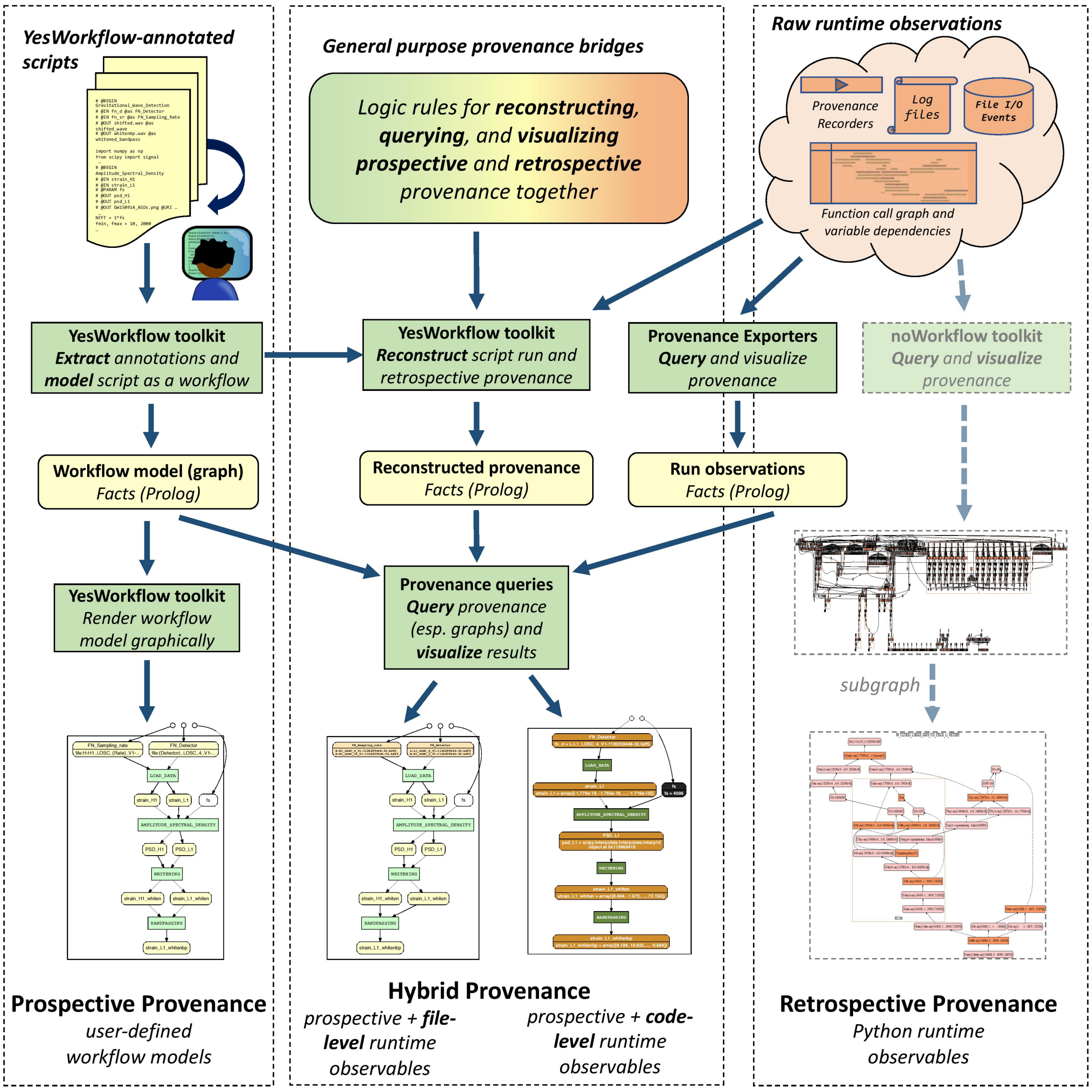

YesWorkflow-annotated scripts
General purpose provenance bridges
Logic rules for reconstructing, querying, and visualizing prospective and retrospective provenance together
Raw runtime observations
Provenance Recorders
Log files
File I/O Events
Function call graph and variable dependencies
YesWorkflow toolkit Extract annotations and model script as a workflow
YesWorkflow toolkit Reconstruct script run and retrospective provenance
Provenance Exporters Query and visualize provenance
noWorkflow toolkit Query and visualize provenance
Workflow model (graph) Facts (Prolog)
Reconstructed provenance Facts (Prolog)
Run observations Facts (Prolog)
Provenance queries Query provenance (esp. graphs) and visualize results
YesWorkflow toolkit Render workflow model graphically
subgraph
Prospective Provenance
user-defined workflow models
Hybrid Provenance
prospective + file-level runtime observables
prospective + code-level runtime observables
Retrospective Provenance
Python runtime observables

Figure 1. Prospective provenance (left) and retrospective provenance (right) are combined to create different hybrid provenance products (center).

Recording File-Level Provenance with DataONE RunManagers (RM)

DataONE[1] provides a RunManager[2] provenance recording and management system for MATLAB and R script executions. The RunManagers overload and thus intercept file I/O operations to automatically capture runtime observables at the file level. The RunManager API provides functions[3] for capturing, searching, archiving, and sharing provenance. DataONE provides two RunManager implementations: the `recordr` R package[4] and the MATLAB toolbox[5]. The provenance captured during a script execution includes information about the script that was run, the files that were read or written, and details about the execution environment at the time of execution. A data package including the script itself, input files, and generated files associated with the run can easily be published to a repository within the DataONE federation.

Code-Level Provenance Capture with noWorkflow

The most detailed retrospective provenance recorder employed in these examples is noWorkflow (Murta et al., 2014). The noWorkflow system (NW) employs a Python profiling library to capture fine-grained runtime provenance at the level of Python function calls, variable assignments, and variable dependencies. The previously reported 'provenance bridge' (Pimentel et al., 2016) between YW and NW provenance information is essential to some of the provenance use cases and demonstrations below (Zhang et al., 2017).

Hybrid Provenance

Figure 1 provides an overview of our approach: A user models a workflow *W* using YW-annotations embedded as script comments. In this way, the user's conceptual workflow model can be bundled in a language-independent way with the executable program code[6]. The resulting YW model of a script constitutes a conceptual-level form of prospective provenance. The YW tool translates these annotations into an internal database format that can be exported as a ProvONE-compliant model (Cuevas-Vicenttín et al., 2015), or in graph form, that can be rendered visually using Graphviz[7]. The graphical rendering provides users with a high-level overview of *W*'s structure, including its underlying dataflow dependencies. Queries against the YW model (implemented, for example, in Prolog or Datalog) can then be used to retrieve the

1 DataONE: https://www.dataone.org/

2 See: https://github.com/DataONEorg/sem-prov-design/blob/master/docs/PROV-capture/Run-manager-API.rst

3 See: https://github.com/DataONEorg/sem-prov-design/blob/master/docs/PROV-capture/Run-manager-API.rst#run-manager-api

4 DataONE recordr R package: https://github.com/NCEAS/recordr

5 DataONE MATLAB toolbox: https://github.com/DataONEorg/matlab-dataone

6 YW models can also be created in separate files, e.g., during workflow design, when an executable version of the script may not yet exist.

7 Graphviz: https://www.graphviz.org/

upstream lineage of any final or intermediate data product *y*, (i.e., the subgraph of data and steps upstream of *y* in *W*), and to compute the downstream influence of *x* (i.e., the subgraph of the data derived from *x* along with the derivation steps, both of which are prospective lineage. While YW can not reveal the retrospective lineage of data on its own, the prospective lineages can be combined with retrospective provenance from other sources to yield partial retrospective lineages in some limited cases as demonstrated later in the paper. The YW toolkit can exploit different sources of retrospective provenance information (Figure 1, right), consisting of runtime-observable events, to produce hybrid provenance (Figure 1, middle). This requires the definition of suitable 'bridge rules' to establish associations between corresponding elements in the two types of provenance. We can distinguish different kinds of runtime observables, e.g., file-level observables from the DataONE RunManagers for MATLAB and R that overload file-IO commands, and code-level observables, e.g., from the noWorkflow system (Murta et al., 2014), which captures fine-grained runtime information at the level of function calls and changes of program variables while executing a Python script. Both the file-level and code-level runtime observables can be exploited for powerful hybrid queries. The former is the focus of our IDCC demonstration, while the latter has been demonstrated elsewhere (Pimentel et al., 2016). Below we discuss some details of different 'provenance bridges' we have developed.

YesWorkflow–RunManager Bridge (YW-RM)

The DataONE RunMangers, i.e., R and MATLAB clients, can record a script run and use the collected file-level provenance to populate a SQLite database, consisting of three main tables: an execution metadata table, a file metadata table, and a tag table. This information can be exported as a YAML file, which in turn can be joined with the YW model to obtain a hybrid provenance graph with file level runtime observables. The DataONE toolbox can also be used to publish data products together with their provenance to a repository that implements the DataONE service API. For example, individual script runs will generate a DataONE metadata package with associated retrospective provenance. After the data package has been indexed by a coordinating node, the logical connection can be viewed and explored via DataONE search[8]. This facilitates data sharing in the community with proper attribution of results transitively across generations of derived data products, i.e., the user can publish a DataONE-compliant data package in OAI-ORE[9] format (including the ProvONE provenance document, the script itself, and its YW-generated workflow view) to a member node within the DataONE federated network of member node repositories (Cao et al., 2016a).

YesWorkflow–noWorkflow Bridge (YW-NW)

Similar to the YW-RM bridge, the YW-NW bridge is used to integrate the prospective provenance declared via YesWorkflow annotations with the retrospective provenance captured from noWorkflow. Since noWorkflow is a provenance recorder that is capable of capturing runtime observables at both file- and code-level, there are different bridge rules for hybrid provenance reconstruction:

8 DataONE Search Site: https://search-sandbox-2.test.dataone.org/#data

9 Open Archives Initiative Object Reuse and Exchange

1. **YW-NW bridge at the file level:** Similar to the YW-RM bridge approach, which can select file-level trace information generated by NW, and export this information from the noWorkflow SQLite database in a YAML format. These file-level observables in YAML can then be joined with YW prospective information to reconstruct hybrid provenance that can be queried by YW model, thus providing a simplified YW-NW bridge with file-level granularity.

2. **YW-NW bridge at the code level:** The "Yin and Yang" approach of (Pimentel, et al., 2016) combines a YW model with the retrospective details captured by noWorkflow at the program variable-level. In Figure 2 we employ YW model to illustrate how the YW-NW bridge has been prototypically implemented. In the YW model, green boxes represent compute steps and yellow nodes represent data flowing between steps. This YW-NW bridge matches variables defined in YW annotations and those in NW, to link Python variable values captured by noWorkflow with data elements declared in YW. With this bridge, we are able to answer hybrid provenance queries of specific data such as *"For a given output value of a variable in an execution of script, what are the values of the inputs and the associated scientific operations and steps?"*. Similarly, we can create data lineage visualizations that combine both prospective and retrospective provenance information, while preserving the simplicity of YW graphs. Below we use a concrete example for illustration, i.e., the LIGO use case.

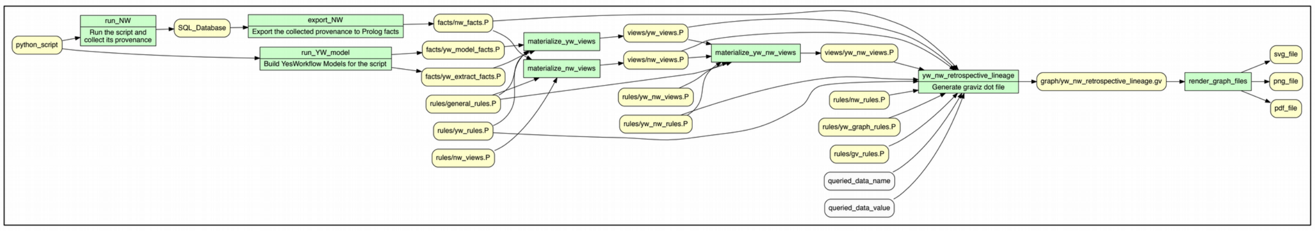


Figure 2. Workflow depicting how the YW-NW bridge from (Pimentel, et al., 2016) works.

Example Use Cases

In this section we present four use cases. The four script-based workflow use cases cover multiple disciplines (astrophysics, climate science, biodiversity data curation, and social networks) and are implemented in different programming languages (Python, R, and MATLAB) and query languages (SQLite and XSB-Prolog), and highlight the usefulness of diverse forms of retrospective provenance when combined with prospective provenance. In a nutshell, we demonstrate three retrospective provenance capturing systems: (a) YW-recon, a lightweight provenance system for reconstructing provenance, (b) the DataONE MATLAB RunManager, and (c) noWorkflow, for capturing Python execution provenance. The four use cases are:

- **Twitter use case:** A Python script in social media analytics, whose hybrid provenance information is captured combining YW and NW at the file level.
- **C3C4 use case:** A MATLAB script in climate science, whose hybrid provenance information is collected from YW and the DataONE MATLAB RunManager.
- **LIGO use case:** A Python script in astrophysics, whose hybrid provenance information is obtained by combining YW and NW at the program variable (i.e., code) level.
- **SPNHC use case:** A Python script in biodiversity data curation, whose hybrid provenance information is obtained by combining YW and log file information, similar to YW-recon.

For each example, by using the YW prospective provenance modelling, a researcher can visualize a YW-annotated script as a ProvONE-compliant workflow graph.

However, data dependencies between script outputs and inputs may be hard to trace visually, especially for complex scripts and fine-grained provenance graphs. In Cao et al. (2016b) and Zhang et al. (2017) we demonstrated how to *query* different types of provenance graphs, i.e., prospective and hybrid graphs. We demonstrated the prospective provenance created by YW and the retrospective provenance collected by various provenance capturing systems, or a combination of several together, to answer queries that cannot be answered solely by prospective provenance or retrospective provenance. More specifically, the following prospective (Q1-Q4) and hybrid queries (Q5-Q6) can be supported for all our four use cases:

- *Prospective queries:*
    - Q1: Render the workflow (sub)graph upstream of a given data product D, where D can be any one (output or intermediate) data element of the YW model of the script;
    - Q2: List the script inputs that are upstream of a given data product D;
    - Q3: Render the workflow (sub)graph downstream of a particular script input;
    - Q4: List the outputs that depend on / are downstream of a particular script input;
- *Hybrid queries:*
    - Q5: Render the hybrid workflow (sub)graph (reconstructed prospective model augmented by retrospective provenance) that is upstream of a given data product D;
    - Q6: Render the hybrid workflow graph (reconstructed prospective model augmented by retrospective provenance) with all runtime observables.

Use Case: Twitter Sentiment Analysis

Sentiment analysis has been very popular in social media analytics over the past few years. In this example, we describe a simple script implemented in Python that uses the

NLTK[10] library to assign sentiment scores to tweets. The script is commented using YW annotations and the conceptual workflow (prospective) graph is visualized by YW in Figure 3a. Figure 3b shows the downstream prospective influence subgraph for the data element ‘PositiveCount’. The green box represents a computation step and the yellow represents a data element. Both graph query results shown in Figure 3 can be obtained by querying the YW model using logic rules (e.g., in XSB-Prolog) without the need to first execute the Python script. The script can be further investigated via more advanced retrospective and hybrid queries; see the GitHub repository[11] for additional details.

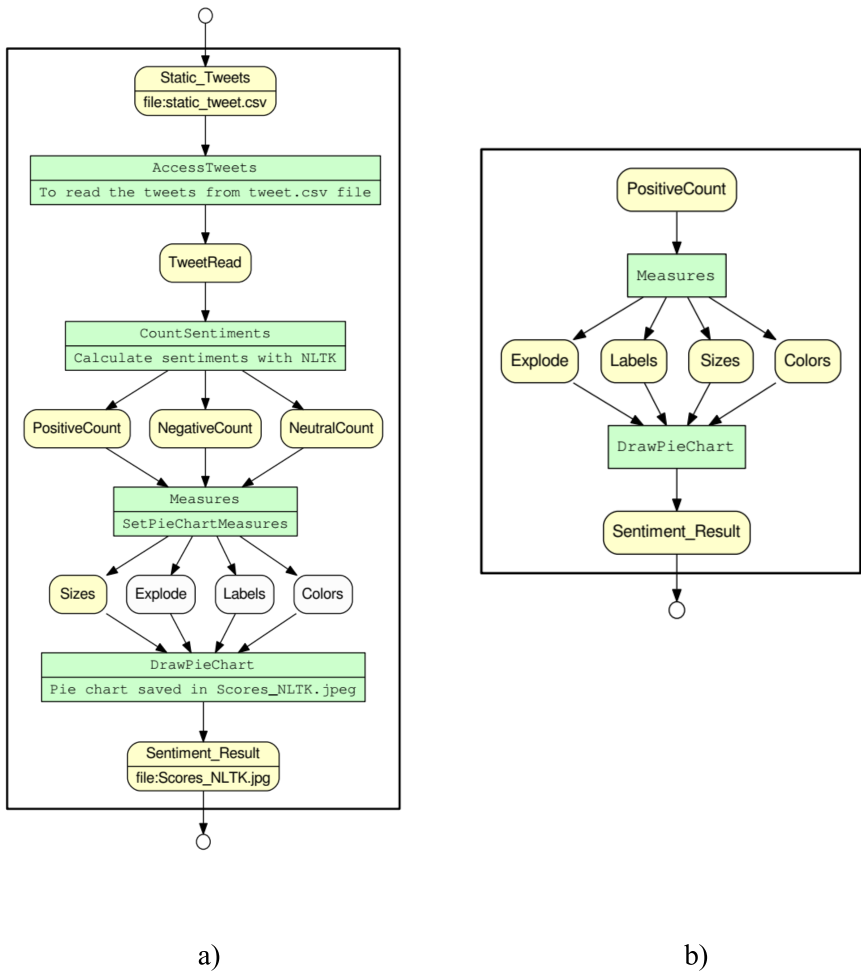


Figure 3. Twitter example (sub)workflows: (a) complete workflow graph; (b) Q3 workflow subgraph (based on YW-perspective provenance) that renders downstream of the data element “PositiveCount”.

10 Sentiment analysis: http://www.nltk.org/howto/sentiment.html

11 Twitter use case GitHub repo: https://github.com/yesworkflow-org/yw-idcc-17/tree/master/examples/Twitter

Use Case: C3C4

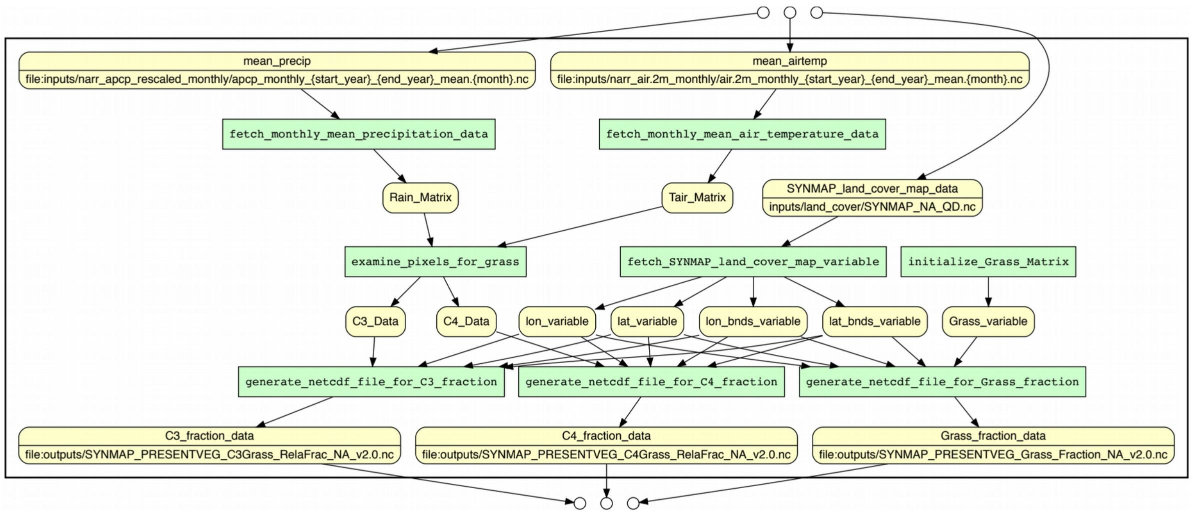
In this section, we use a DataONE example to demonstrate hybrid provenance and hybrid queries. The example MATLAB script produces C3/C4 (Carbon) soil maps for North America using average rain and air temperature monthly data, e.g., from the year 2000 to 2010. An earth scientist uses YW annotations to mark up the script and thereby expose the underlying prospective provenance graph that is inherent in the script (see Figure 4a). Note that a URI template is used for the input and output data files to specify the file name and path metadata. For example, the data "*mean_precip*" includes three template variables ("start_year", "end_year", and "month") in curly braces. It is easy to see from Figure 4a that there are three data inputs ("*mean_precip*", "*mean_airtemp*", and "*SYNMAP_land_cover_map_data*") and three output data elements ("*C3_fraction_data*", "*C4_fraction_data*", and "*Grass_fraction_data*"), respectively. Then, a provenance query can be asked based on the prospective provenance harvested by YW, e.g., *"Show me the workflow graph upstream of the data element Grass_fraction_data"*. The answer to this query is shown in Figure 4b, which indicates that the "*Grass_fraction_data*" does *not* depend on the first two inputs ("*mean_precip*" and "*mean_airtemp*") but may depend on only one input data, i.e., "*SYNMAP_land_cover_map_data*". In this way, a dataflow path from a node *x* to *y* represents a possible data dependency of *y* on *x*. Conversely, the absence of a path from *x* to *y* indicates that *y* does not depend on *x,* according to given YW model.

By using the MATLAB RM tool to record a run of this script, the user can not only obtain the expected execution results, but also capture file-level retrospective provenance information. The hybrid provenance can be derived by joining the YW model and RunManager retrospective provenance via YW-URI templates (i.e., using a YW-URI-RM file level bridge). After obtaining the derived hybrid provenance information, we can ask hybrid queries. For example, (1) *Show the complete hybrid graph at file level where the prospective graph is augmented and enriched by all file-level runtime observables in retrospective provenance*; or (2) *Given a data item "Grass_fraction_data", show the reconstructed upstream augmented provenance graph in context of the YW workflow graph*.

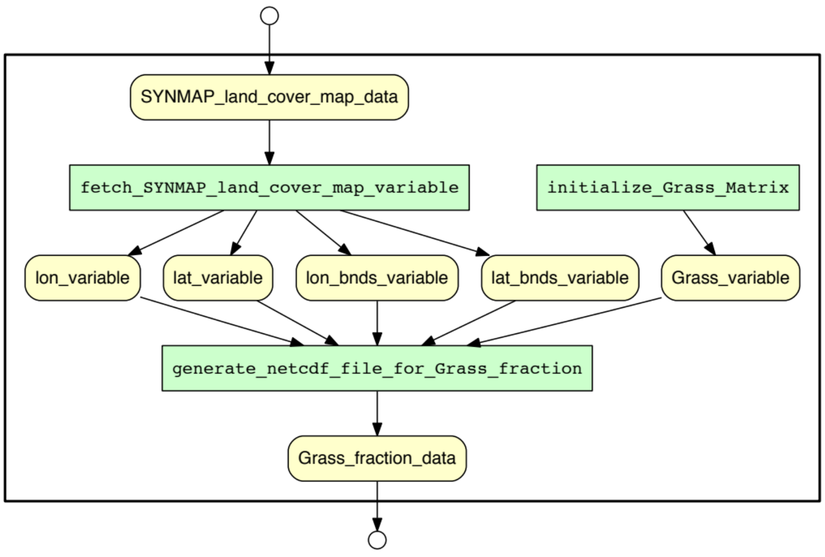
The generated reconstructed (sub)graphs containing derived hybrid provenance information are shown in Figure 4c and Figure 4d, respectively. From Figure 4c, we can see that when the script was running, it has actually read 24 (12*2) data files containing monthly rain and air temperature data files, plus one additional land map file, resulting in a total of 25 input data files, spread across the three data elements in the YW model (Figure 4a). We also see that the script has written three files to the outputs folder.

Use Case: LIGO

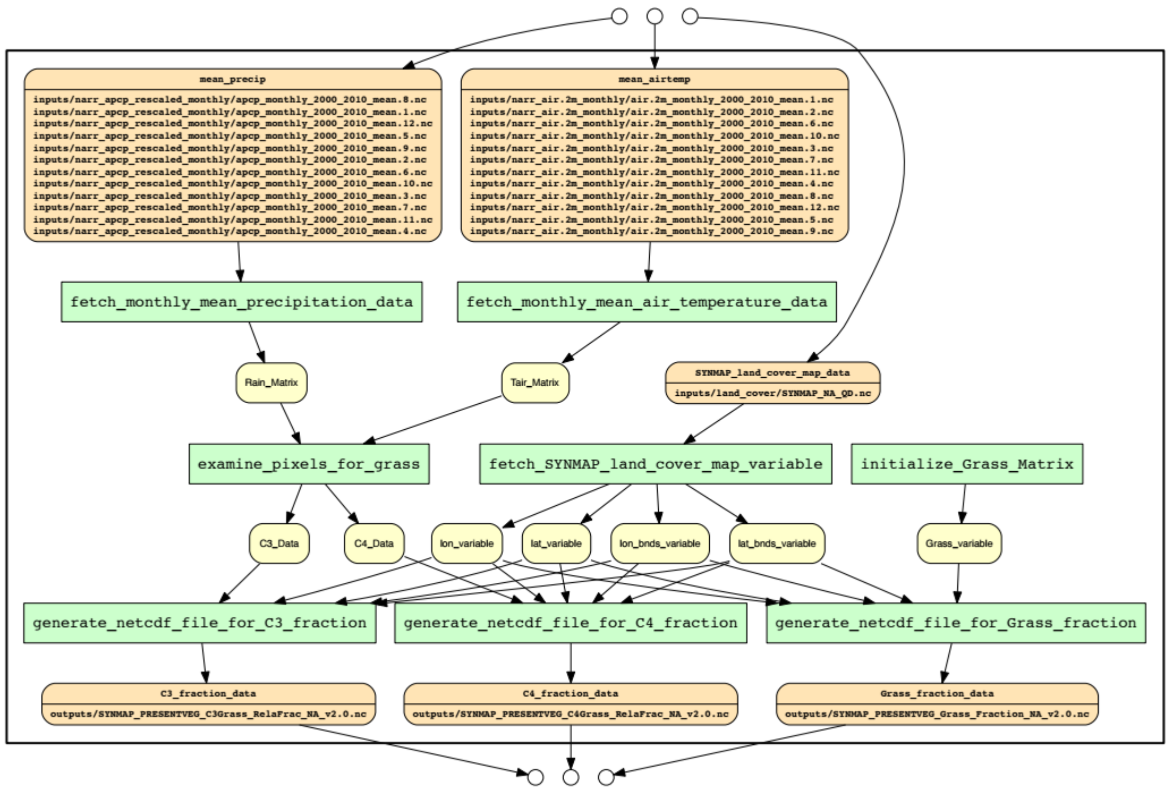
LIGO Open Science Center detected gravitational waves from a binary black hole merger on September 14, 2015, and called the event GW150914 (LSC, 2015). With strain time series data from GW150914 collected by Detector H1 from the LIGO Hanford Observatory and Detector L1 from LIGO Livingston Observatory, electronic engineers write a Python script to perform signal processing tasks that eliminate the disturbances from the local source and visualize the gravitational wave in spectrogram and audio form. Figure 5a depicts the YW subgraph upstream of variable "*strain_L1_whitenbp*" in YW prospective model. Figure 5b and 5c both demonstrate the reconstructed hybrid workflow graph upstream of the same data, whose differences lie in that the former was implemented via YW-recon-URI template at the file-level while the latter YW- NW bridge at both the file and the code level. In the hybrid graph of YW-NW bridge at program variable level (Figure 5c), the black box represents a parameter variable associated with its value, and the orange box for an input/output data associated with its value.


mean_precip
file:inputs/narr_apcp_rescaled_monthly/apcp_monthly_{start_year}_{end_year}_mean.{month}.nc
mean_airtemp
file:inputs/narr_air.2m_monthly/air.2m_monthly_{start_year}_{end_year}_mean.{month}.nc
fetch_monthly_mean_precipitation_data
fetch_monthly_mean_air_temperature_data
Rain_Matrix
Tair_Matrix
SYNMAP_land_cover_map_data
inputs/land_cover/SYNMAP_NA_QD.nc
examine_pixels_for_grass
fetch_SYNMAP_land_cover_map_variable
initialize_Grass_Matrix
C3_Data
C4_Data
lon_variable
lat_variable
lon_bnds_variable
lat_bnds_variable
Grass_variable
generate_netcdf_file_for_C3_fraction
generate_netcdf_file_for_C4_fraction
generate_netcdf_file_for_Grass_fraction
C3_fraction_data
file:outputs/SYNMAP_PRESENTVEG_C3Grass_RelaFrac_NA_v2.0.nc
C4_fraction_data
file:outputs/SYNMAP_PRESENTVEG_C4Grass_RelaFrac_NA_v2.0.nc
Grass_fraction_data
file:outputs/SYNMAP_PRESENTVEG_Grass_Fraction_NA_v2.0.nc

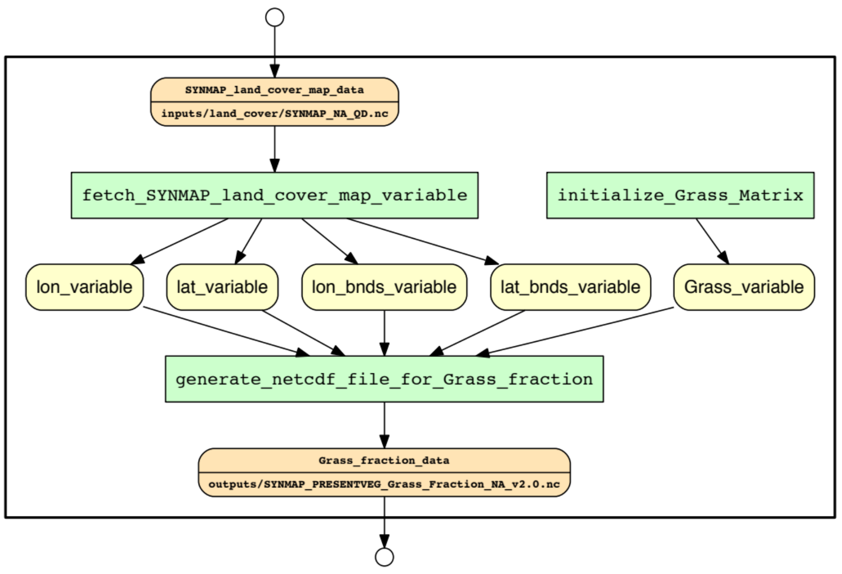
a)



b)


mean_precip
inputs/narr_apcp_rescaled_monthly/apcp_monthly_2000_2010_mean.8.nc
inputs/narr_apcp_rescaled_monthly/apcp_monthly_2000_2010_mean.1.nc
inputs/narr_apcp_rescaled_monthly/apcp_monthly_2000_2010_mean.12.nc
inputs/narr_apcp_rescaled_monthly/apcp_monthly_2000_2010_mean.5.nc
inputs/narr_apcp_rescaled_monthly/apcp_monthly_2000_2010_mean.9.nc
inputs/narr_apcp_rescaled_monthly/apcp_monthly_2000_2010_mean.2.nc
inputs/narr_apcp_rescaled_monthly/apcp_monthly_2000_2010_mean.6.nc
inputs/narr_apcp_rescaled_monthly/apcp_monthly_2000_2010_mean.10.nc
inputs/narr_apcp_rescaled_monthly/apcp_monthly_2000_2010_mean.3.nc
inputs/narr_apcp_rescaled_monthly/apcp_monthly_2000_2010_mean.7.nc
inputs/narr_apcp_rescaled_monthly/apcp_monthly_2000_2010_mean.11.nc
inputs/narr_apcp_rescaled_monthly/apcp_monthly_2000_2010_mean.4.nc
mean_airtemp
inputs/narr_air.2m_monthly/air.2m_monthly_2000_2010_mean.1.nc
inputs/narr_air.2m_monthly/air.2m_monthly_2000_2010_mean.2.nc
inputs/narr_air.2m_monthly/air.2m_monthly_2000_2010_mean.6.nc
inputs/narr_air.2m_monthly/air.2m_monthly_2000_2010_mean.10.nc
inputs/narr_air.2m_monthly/air.2m_monthly_2000_2010_mean.3.nc
inputs/narr_air.2m_monthly/air.2m_monthly_2000_2010_mean.7.nc
inputs/narr_air.2m_monthly/air.2m_monthly_2000_2010_mean.11.nc
inputs/narr_air.2m_monthly/air.2m_monthly_2000_2010_mean.4.nc
inputs/narr_air.2m_monthly/air.2m_monthly_2000_2010_mean.8.nc
inputs/narr_air.2m_monthly/air.2m_monthly_2000_2010_mean.12.nc
inputs/narr_air.2m_monthly/air.2m_monthly_2000_2010_mean.5.nc
inputs/narr_air.2m_monthly/air.2m_monthly_2000_2010_mean.9.nc
fetch_monthly_mean_precipitation_data
fetch_monthly_mean_air_temperature_data
Rain_Matrix
Tair_Matrix
SYNMAP_land_cover_map_data
inputs/land_cover/SYNMAP_NA_QD.nc
examine_pixels_for_grass
fetch_SYNMAP_land_cover_map_variable
initialize_Grass_Matrix
C3_Data
C4_Data
lon_variable
lat_variable
lon_bnds_variable
lat_bnds_variable
Grass_variable
generate_netcdf_file_for_C3_fraction
generate_netcdf_file_for_C4_fraction
generate_netcdf_file_for_Grass_fraction
C3_fraction_data
outputs/SYNMAP_PRESENTVEG_C3Grass_RelaFrac_NA_v2.0.nc
C4_fraction_data
outputs/SYNMAP_PRESENTVEG_C4Grass_RelaFrac_NA_v2.0.nc
Grass_fraction_data
outputs/SYNMAP_PRESENTVEG_Grass_Fraction_NA_v2.0.nc

c)

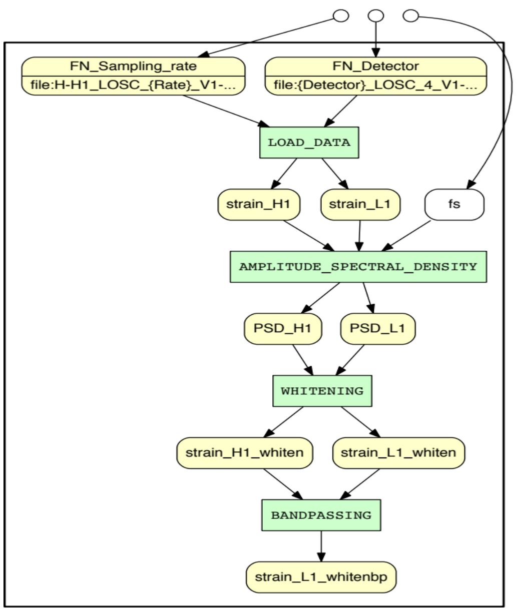

SYNMAP_land_cover_map_data
inputs/land_cover/SYNMAP_NA_QD.nc
fetch_SYNMAP_land_cover_map_variable
initialize_Grass_Matrix
lon_variable
lat_variable
lon_bnds_variable
lat_bnds_variable
Grass_variable
generate_netcdf_file_for_Grass_fraction
Grass_fraction_data
outputs/SYNMAP_PRESENTVEG_Grass_Fraction_NA_v2.0.nc

d)

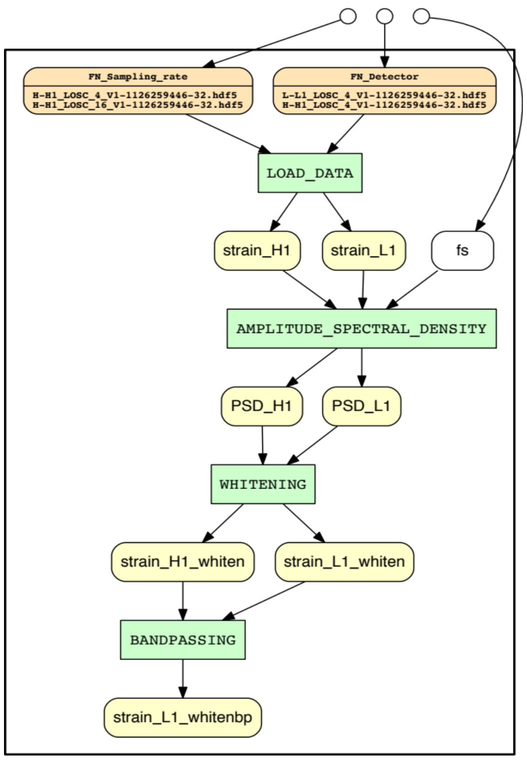
Figure 4. C3C4 example workflows: (a) complete workflow graph (with YW-URI templates); (b) Q1 workflow subgraph (based on YW-perspective provenance), upstream of the data element “*Grass_fraction_data*”; (c) Q6 complete hybrid graph (based on YW-URI-RM at the file level): the reconstructed prospective model augmented by retrospective provenance with all file-level runtime observables; (d) Q5 hybrid subgraph (based on YW-URI-RM at file level), upstream of the data element “*Grass_fraction_data*”.

YW-NW bridge approach can not only enable queries that can be answered by YW, but also answer queries for runtime data information that can’t be answered solely by YW. For example, the hybrid graph can show the variables and their values that are the inputs of the step of *Bandpassing*, and emit the output “*strain_L1_whitenbp*” with value of [8.184, ..., -0.684]. From the prospective provenance graph of YW model (Figure 5a), both data products “*strain_H1_whiten*” and “*strain_L1_whiten*” appear as inputs flowing into the computational step *Bandpassing* from which output “*strain_L1_whitenbp*”. However, from the hybrid graph of YW-NW bridge at program variable level in Figure 5c, only “*strain_L1_whiten*” is being *bandpassed* with value [8.494, ..., 72.156] to produce the desired “*strain_L1_whitenbp*”, which is also self-explanatory from their data names. This means that “*strain_H1_whiten*” is being passed to and used in *bandpassing*, but is irrelevant to the output of “*strain_L1_whitenbp*”. From another point of view, we also noticed from YW-NW bridge at file-level (Figure 5b), that two data files are being detected as input for “*FN_Detector*” since the augmented prospective graph lists all actual inputs of the program that correspond to data variables, whereas only one of the two input files for L1 detector is consumed and thus shown within YW-NW bridge at code level (Figure 5c). The YW-NW bridge at the code level not only records the lineage of the queried data, but also captures the data dependency on the actual input file, i.e. at the file level.

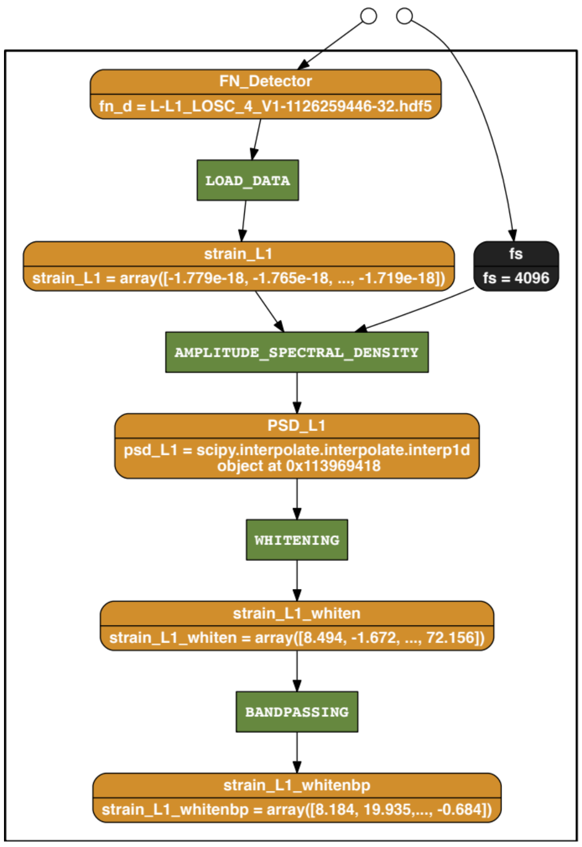
From both the LIGO and C3C4 examples, we can see that the YW-recon approach based on URI templates at the file-level runtime observables alone (1) can determine the definite independency between data products; and (2) may or may not precisely define the data dependencies. In contrast, the YW-NW bridge can generate code-level runtime information, which once be integrated with YesWorkflow model, can capture the correct data lineage and dependency relationships that supplement retrospective data information for YesWorkflow models.



a)


FN_Sampling_rate
H-H1_LOSC_4_V1-1126259446-32.hdf5
H-H1_LOSC_16_V1-1126259446-32.hdf5
FN_Detector
L-L1_LOSC_4_V1-1126259446-32.hdf5
H-H1_LOSC_4_V1-1126259446-32.hdf5
LOAD_DATA
strain_H1
strain_L1
fs
AMPLITUDE_SPECTRAL_DENSITY
PSD_H1
PSD_L1
WHITENING
strain_H1_whiten
strain_L1_whiten
BANDPASSING
strain_L1_whitenbp

b)


FN_Detector
fn_d = L-L1_LOSC_4_V1-1126259446-32.hdf5
LOAD_DATA
strain_L1
strain_L1 = array([-1.779e-18, -1.765e-18, ..., -1.719e-18])
fs
fs = 4096
AMPLITUDE_SPECTRAL_DENSITY
PSD_L1
psd_L1 = scipy.interpolate.interpolate.interp1d object at 0x113969418
WHITENING
strain_L1_whiten
strain_L1_whiten = array([8.494, -1.672, ..., 72.156])
BANDPASSING
strain_L1_whitenbp
strain_L1_whitenbp = array([8.184, 19.935,..., -0.684])

c)

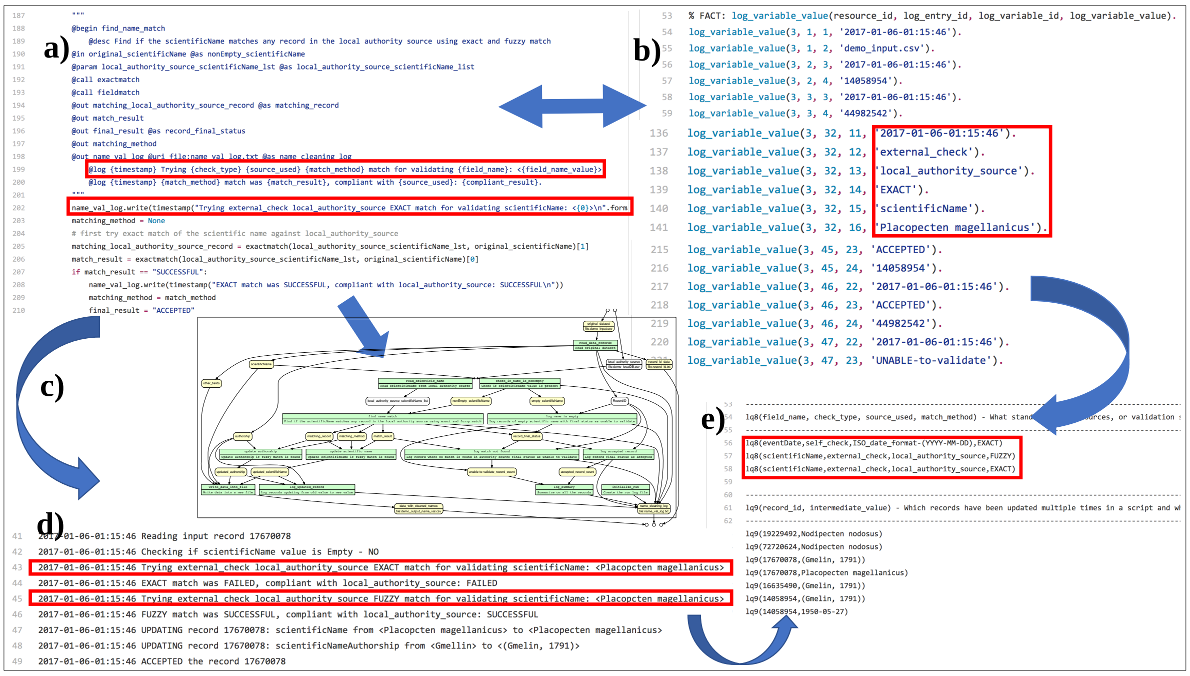
Figure 5. LIGO example (sub)workflows: (a) Q1 workflow subgraph (based on YW-perspective provenance with YW-URI templates) that renders upstream of the data element "*strain_L1_whitenbp*"; (b) Q5 hybrid subgraph (based on **YW-recon-URI at file level)**: the reconstructed prospective model augmented by retrospective provenance with file-level runtime observables that renders upstream of the data element "*strain_L1_whitenbp*"; (c) Q5 hybrid subgraph (based on **YW-NW bridge at both file and code levels)** that renders upstream of the data element "*strain_L1_whitenbp*".

Use Case: SPNHC

In this use case, we demonstrate how the YW toolkit developed by the Kurator[12] team facilitates exploration of the results of running a simple data cleaning script written in Python (Figure 6a) on a biodiversity dataset. This script takes as input a CSV file containing occurrence records represented in Darwin Core terms; in successive steps validates the various fields (scientific name, authorship, and event date) of each input record; replaces nonstandard or incorrect values for particular fields when the meanings or intents of these input values are unambiguous; and optionally discards records missing values for essential fields or containing incorrect or ambiguous values for which enhancements cannot be proposed. The cleaned data set ultimately produced by

12 Kurator: https://opensource.ncsa.illinois.edu/confluence/display/KURATOR/Kurator+Project+Home

the script is written to another CSV file. Each of the data validation steps in the script writes key information about their operations into simple log files (.txt).



a)
b)
c)
d)
e)
original_dataset
file:demo_input.csv
read_data_records
Read original dataset
scientificName
other_fields
local_authority_source
file:demo_localDB.csv
record_id_data
file:record_id.txt
read_scientific_name
Read scientificName from local authority source
check_if_name_is_nonempty
Check if scientificName value is present
local_authority_source_scientificName_list
nonEmpty_scientificName
empty_scientificName
RecordID
find_name_match
Find if the scientificName matches any record in the local authority source using exact and fuzzy match
log_name_is_empty
Log records of empty scientific name with final status as unable to validate
authorship
matching_record
matching_method
match_result
record_final_status
update_authorship
Update authorship if fuzzy match is found
update_scientific_name
Update scientificName if fuzzy match is found
log_match_not_found
Log record where no match is found in authority source final status as unable to validate
log_accepted_record
Log record final status as accepted
updated_authorship
updated_scientificName
unable-to-validate_record_count
accepted_record_count
write_data_into_file
Write data into a new file
log_updated_record
Log records updating from old value to new value
log_summary
Summarize on all the records
initialize_run
Create the run log file
data_with_cleaned_names
file:demo_output_name_val.csv
name_cleaning_log
file:name_val_log.txt

```
        """
        @begin find_name_match
            @desc Find if the scientificName matches any record in the local authority source using exact and fuzzy match
        @in original_scientificName @as nonEmpty_scientificName
        @param local_authority_source_scientificName_lst @as local_authority_source_scientificName_list
        @call exactmatch
        @call fieldmatch
        @out matching_local_authority_source_record @as matching_record
        @out match_result
        @out final_result @as record_final_status
        @out matching_method
        @out name_val_log @uri file:name_val_log.txt @as name_cleaning_log
            @log {timestamp} Trying {check_type} {source_used} {match_method} match for validating {field_name}: <{field_name_value}>
            @log {timestamp} {match_method} match was {match_result}, compliant with {source_used}: {compliant_result}.
        """
        name_val_log.write(timestamp("Trying external_check local_authority_source EXACT match for validating scientificName: <{0}>\n".form
        matching_method = None
        # first try exact match of the scientific name against local_authority_source
        matching_local_authority_source_record = exactmatch(local_authority_source_scientificName_lst, original_scientificName)[1]
        match_result = exactmatch(local_authority_source_scientificName_lst, original_scientificName)[0]
        if match_result == "SUCCESSFUL":
            name_val_log.write(timestamp("EXACT match was SUCCESSFUL, compliant with local_authority_source: SUCCESSFUL\n"))
            matching_method = match_method
            final_result = "ACCEPTED"
```

```
% FACT: log_variable_value(resource_id, log_entry_id, log_variable_id, log_variable_value).
log_variable_value(3, 1, 1, '2017-01-06-01:15:46').
log_variable_value(3, 1, 2, 'demo_input.csv').
log_variable_value(3, 2, 3, '2017-01-06-01:15:46').
log_variable_value(3, 2, 4, '14058954').
log_variable_value(3, 3, 3, '2017-01-06-01:15:46').
log_variable_value(3, 3, 4, '44982542').
log_variable_value(3, 32, 11, '2017-01-06-01:15:46').
log_variable_value(3, 32, 12, 'external_check').
log_variable_value(3, 32, 13, 'local_authority_source').
log_variable_value(3, 32, 14, 'EXACT').
log_variable_value(3, 32, 15, 'scientificName').
log_variable_value(3, 32, 16, 'Placopecten magellanicus').
log_variable_value(3, 45, 23, 'ACCEPTED').
log_variable_value(3, 45, 24, '14058954').
log_variable_value(3, 46, 22, '2017-01-06-01:15:46').
log_variable_value(3, 46, 23, 'ACCEPTED').
log_variable_value(3, 46, 24, '44982542').
log_variable_value(3, 47, 22, '2017-01-06-01:15:46').
log_variable_value(3, 47, 23, 'UNABLE-to-validate').
```

```
2017-01-06-01:15:46 Reading input record 17670078
2017-01-06-01:15:46 Checking if scientificName value is Empty - NO
2017-01-06-01:15:46 Trying external_check local_authority_source EXACT match for validating scientificName: <Placopcten magellanicus>
2017-01-06-01:15:46 EXACT match was FAILED, compliant with local_authority_source: FAILED
2017-01-06-01:15:46 Trying external_check local_authority_source FUZZY match for validating scientificName: <Placopcten magellanicus>
2017-01-06-01:15:46 FUZZY match was SUCCESSFUL, compliant with local_authority_source: SUCCESSFUL
2017-01-06-01:15:46 UPDATING record 17670078: scientificName from <Placopcten magellanicus> to <Placopecten magellanicus>
2017-01-06-01:15:46 UPDATING record 17670078: scientificNameAuthorship from <Gmellin> to <(Gmelin, 1791)>
2017-01-06-01:15:46 ACCEPTED the record 17670078
```

```
--------------------------------------------------------------------------------------------------------
lq8(field_name, check_type, source_used, match_method) - What stand                ources, or validation :
--------------------------------------------------------------------------------------------------------
lq8(eventDate,self_check,ISO_date_format-(YYYY-MM-DD),EXACT)
lq8(scientificName,external_check,local_authority_source,FUZZY)
lq8(scientificName,external_check,local_authority_source,EXACT)

--------------------------------------------------------------------------------------------------------
lq9(record_id, intermediate_value) - Which records have been updated multiple times in a script and wh
--------------------------------------------------------------------------------------------------------
lq9(19229492,Nodipecten nodosus)
lq9(72720624,Nodipecten nodosus)
lq9(17670078,(Gmelin, 1791))
lq9(17670078,Placopecten magellanicus)
lq9(16635490,(Gmelin, 1791))
lq9(14058954,(Gmelin, 1791))
lq9(14058954,1950-05-27)
```

Figure 6. YW-log implementation for the SPNHC biodiversity data cleaning use case: (a) YW-annotated (`@out`, `@uri`, and `@log`) script that declare log file, file path, and log template; (b) *YW-recon* log_variable_value facts; (c) YW-rendered subworkflow graph (d) the log file written by the script (e) log queries.

On the one hand, by analyzing the YW annotations in a script, YesWorkflow can render a prospective provenance graph of a script to be executed (Figure 6c). On the other hand, YW retrospective provenance reconstructions based on URI and log template matching can assist interpreting and querying log files in terms of the structure of the script itself. YW can answer questions about the output data set, the script run, and the data validation, field updates, and record removal events that occurred during the data cleaning process. Using log files easily written by such scripts, YesWorkflow can reveal which fields in particular records were marked as suspect and why, which of these fields were corrected, and what records were removed. YesWorkflow depends on special comments in the script, and thus these annotations enable interpreting and querying log files in terms of the structure of the script itself. Figure 6d and 6e depict the log file content and log query output using URI template and extended log feature, respectively.

Conclusions and Future Work

The challenges we addressed in this paper include: (1) Most computational analyses and workflows are conducted using scripts in different programming languages, such as Python, R and MATLAB; (2) Retrospective provenance observables, from DataONE RunManager (file-level), or noWorkflow (Python code-level) only yield isolated fragments of the overall data lineage and processing history; (3) Prospective provenance could be used to link and contextualize fragments into a meaningful and comprehensible workflow, but script alone do not reveal the underlying workflow graph; and (4) Provenance (like other metadata) appears to be rarely actionable or immediately useful for those who are expected to provide it ('provenance for others'). The approach we propose for hybrid provenance in this paper consists of three steps: (1) Simple YesWorkflow annotations allow users to explicitly reveal the workflow (prospective provenance) that was originally implicit in scripts; (2) Prospective provenance queries expose and test data dependencies at the workflow level; and (3) Hybrid provenance queries situate runtime observables (retrospective provenance) in the overall workflow, yielding meaningful knowledge artifacts. Our proposed approach integrates comprehensible workflow graphs and customizable provenance reports for script runs, along with data and code in scientific studies ('provenance for self'). Using the example scripts as a testbed, we show how the dual views of retrospective and prospective provenance can both reveal the overall story of a script run, and also the detailed history of particular script products. An earlier, hands-on demonstration illustrating our approach is available via an open source repository (Cao et al., 2016b).

However what about the things YesWorkflow cannot do? – what it cannot do by itself, and what it cannot do even in combination with the partial retrospective provenance information from other sources – at least right now. An example of a situation where our hybrid provenance approaches simply will not work is like this: what would happen if the lineage of a particular data product depended on the outputs of more than one execution of a particular upstream YesWorkflow program block when using either of the YW-NW bridges? As to future work, besides the single script run provenance, we are also interested in multi-run and /or multi-script executions. Multi-run means that the same script is executed multiple times in order to compare among different input settings or to seek optimal output result; while multi-script execution occurs in big computation workflow runs that could last days, weeks, or even months, which usually consist of more than one script or a cascade of subroutines of multiple scripts using different input argument(s) and/or parameters configured at each setting. Under such circumstances, some intermediate data files could be generated and consumed implicitly within such a long chain reaction, which usually play a key role in determining the final model output. However, these can easily get ignored if not enough attention were paid. In this complex case, we are interested in what kind of provenance information could be captured, queried or visualized and at which levels of granularity (e.g., at the script level, data level, field level, record level, file level, function, aggregate level, provided that it has been configured at each). In order to reveal data lineage, we might again need to use a combination of YW and other provenance tracking tools (e.g., DataONE RunManager, NW) for exposing prospective, retrospective and hybrid provenance. We will demonstrate our preliminary results in the demo session of the 12th International Digital Curation Conference, showing the new capabilities described above.

Another direction of future work is that we want to generate and query ProvONE-compatible RDF representations of YW annotations, workflow models, and retrospective provenance so that ProvONE compatible vocabulary extensions may be used in YW in the future via the mapping between YW and ProvONE data model.

Acknowledgments

The work presented in this paper is supported, in part, by NSF awards DBI-1356751 (Kurator), ACI-1430508 (DataONE), SMA-1637155 (SKOPE), and ACI-1541450 (Whole Tale).

References

Altintas, I., Barney, O., & Jaeger-Frank, E. (2006). Provenance collection support in the Kepler scientific workflow system. *IPAW, 2006*. doi:10.1007/11890850_14

Bavoil, L., Callahan, S.P., Crossno, P.J., Freire, J., Scheidegger, C.E., Silva, C.T. & Vo, H.T. (2005). VisTrails: Enabling interactive multiple-view visualizations. In *Visualization 2005 (VIS '05)*, pp. 135–142. IEEE. doi:10.1109/VISUAL.2005.1532788

Belhajjame, K., Cheney, J., Corsar, D., Garijo, D., Soiland-Reyes, S., Zednik, S., & Zhao, J. (2013). W3C PROV-O: The PROV ontology. Retrieved from https://www.w3.org/TR/prov-o/

Bowers, S., McPhillips, T., & Ludäscher, B. (2012). Declarative rules for inferring fine-grained data provenance from scientific workflow execution traces. In *Provenance and Annotation of Data and Processes,* 82–96. Springer, Berlin, Heidelberg. doi:10.1007/978-3-642-34222-6_7

Bowers, S., McPhillips, T., Riddle, S., Anand, M.K., & Ludäscher, B. (2008). Kepler/pPOD: Scientific workflow and provenance support for assembling the tree of life. *IPAW, 2008*. doi:10.1007/978-3-540-89965-5_9

Cao, Y., Jones, C., Vicenttín, V.C., Jones, M.B., Ludäscher, B., McPhillips, T.M., Missier, P., Schwalm, C.R., Slaughter, P., Vieglais, D., Walker, L., & Wei, Y. (2016a). DataONE: A data federation with provenance support. *IPAW 2016,* 230-234. doi:10.1007/978-3-319-40593-3_28

Cao, Y., Vu, D., Wang, Q., Zhang, Q., Thavasimani, P., McPhillips, T., Missier, P., & Ludäscher, B. (2016b). YW query demo site on Github. Retrieved from https://github.com/idaks/dataone-ahm-2016-poster

Cuevas-Vicenttín, V., Ludäscher, B., Missier, P., Belhajjame, K., Chirigati, F., Wei, Y., ... & Altintas, I. (2015). ProvONE: A PROV extension data model for scientific workflow provenance. Retrieved from http://jenkins-1.dataone.org/jenkins/view/Documentation%20Projects/job/ProvONE-Documentation-trunk/ws/provenance/ProvONE/v1/provone.html

Deelman, E., Singh, G., Su, M. H., Blythe, J., Gil, Y., Kesselman, C., ... & Laity, A. (2005). Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming, 13*(3), 219-237. doi:10.1155/2005/128026

Freire, J., Fuhr, N., & Rauber, A. (2016). Reproducibility of data-oriented experiments in e-Science (Dagstuhl Seminar 16041). *Dagstuhl Reports, 6*(1). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik

Freire, J., Koop, D., Chirigati, F.S., & Silva, C.T. (2014). Reproducibility using vistrails. In V. Stodden, F. Leisch & R.D. Peng (Eds) *Implementing Reproducible Research*, pp 33. CRC Press.

Gandrud, C. (2013). Reproducible research with R and R Studio. CRC Press.

Goecks, J., Nekrutenko, A., & Taylor, J. (2010). Galaxy: A comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome Biology, 11*(8). doi:10.1186/gb-2010-11-8-r86

Lerner, B.S., & Boose, E.R. (2014). Collecting provenance in an interactive scripting environment. In *Workshop on the Theory and Practice of Provenance (TaPP)*, Cologne, Germany.

LIGO Scientific Collaboration (LSC). (2015). LIGO Open Science Center: Data release for event GW150914. Retrieved from https://losc.ligo.org/events/GW150914/

Ludäscher, B., Altintas, I., Berkley, C., Higgins, D., Jaeger, E., Jones, M., ... Zhao, Y. (2006). Scientific workflow management and the Kepler system. *Concurrency and Computation: Practice and Experience, 18*(10), 1039–1065. doi:10.1002/cpe.994

McPhillips, T., Bowers, S., Belhajjame, K., & Ludäscher, B. (2015a). Retrospective provenance without a runtime provenance recorder. Paper presented at the 7th USENIX Workshop on the Theory and Practice of Provenance (TaPP 2015).

McPhillips, T., Song, T., Kolisnik, T., Aulenbach, S., Belhajjame, K., Bocinsky, R.K., Cao, Y., Cheney, J., Chirigati, F., Dey, S., Freire, J., Jones, C., Hanken, J., Kintigh, K.W., Kohler, T.A., Koop, D., Macklin, J.A., Missier, P., Schildhauer, M., Schwalm, C., Wei, Y., Bieda, M., & Ludäscher, B. (2015b). YesWorkflow: A user-oriented, language-independent tool for recovering workflow information from scripts. *International Journal of Digital Curation 10*(1), 298-313. doi:10.2218/ijdc.v10i1.370

Missier, P., Belhajjame, K., Zhao, J., Roos, M., & Goble, C. (2008). Data lineage model for Taverna workflows with lightweight annotation requirements. *IPAW, 2008.* doi:10.1007/978-3-540-89965-5_4

Murta, L., Braganholo, V., Chirigati, F., Koop, D., & Freire J. (2014). noWorkflow: Capturing and analyzing provenance of scripts. *IPAW 2014,* 71-83. doi:10.1007/978-3-319-16462-5_6

Oinn, T., Addis, M., Ferris, J., Marvin, D., Senger, M., Greenwood, M., Carver, T., Glover, K., Pocock, M.R., Wipat, A., & Li, P. (2004). Taverna: A tool for the composition and enactment of bioinformatics workflows. *Bioinformatics 20*(17), 3045–3054. doi:10.1093/bioinformatics/bth361

Pimentel, J.F., Dey, S.C., McPhillips, T.M., Belhajjame, K., Koop, D., Murta, L., Braganholo, V., & Ludäscher, B. (2016). Yin and yang: Demonstrating complementary provenance from noWorkflow and YesWorkflow. *IPAW 2016,* 161-165. doi:10.1007/978-3-319-40593-3_13

Tilmes, C., Yesha, Y., & Halem, M. (2010). Tracking provenance of earth science data. *Earth Science Informatics, 3*(1-2), 59-65. doi:10.1007/s12145-010-0046-3

Tsai, Y., McPhillips, S.E., González, A., McPhillips, T M., Zinn, D., Cohen, A. E., ... Soltis, S.M. (2013). Autodrug: Fully automated macromolecular crystallography workflows for fragment-based drug discovery. *Acta Crystallographica Section D: Biological Crystallography, 69*(5), 796–803. doi:10.1107/S0907444913001984

Wolstencroft, K., Haines, R., Fellows, D., Williams, A., Withers, D., Owen, S., Soiland-Reyes, S., Dunlop, I., Nenadic, A., Fisher, P., et al. (2013). The Taverna workflow suite: designing and executing workflows of Web Services on the desktop, web or in the cloud. *Nucleic Acids Research, 2013*. doi:10.1093/nar/gkt328

Zhang, Q., Cao, Y., Wang, Q., Vu, D., Thavasimani, P., McPhillips, T., Missier, P., & Ludäscher, B. (2017). yw-idcc-17 hybrid query demo site. GitHub. Retrieved from https://github.com/yesworkflow-org/yw-idcc-17

Zhao, Y., Wilde, M., & Foster, I. (2006). Applying the virtual data provenance model. In *International Provenance and Annotation Workshop* (pp. 148-161). Springer Berlin Heidelberg.