L-Vis: Visualizing Language-Level Provenance for Program Comprehension

Joe Wonsil and Francis Nguyen

December 10th, 2019

CPSC 547 Final Project

Driving Scenario: Grad Student Code

- A new graduate student inherits (messy) code for data analysis and must understand how to use and alter the scripts.
- Goal: build mental model of unfamiliar code.
    - Useful information can include **library dependencies and their usage**, understanding how **variables change over time** and the **relationship between inputs and outputs** of the analysis code.
- L-Vis visualizes relationships in the code structure of R scripts to help users understand unfamiliar code.

Requirements & Tasks

We conducted unstructured interviews to understand how program comprehension and visualization interact.

- **Participants:** software engineering professor, peer CS graduate students, and introductory CS undergrads.

Their responses helped define tasks L-Vis should consider in its design.

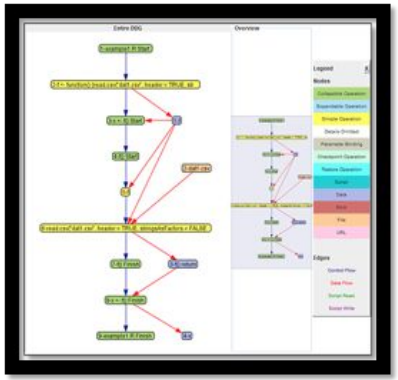
- **Locate:** Identify parts of code an external library affects.
- **Locate:** Display affected code if a user changes a variable's value.
- **Present:** Highlighting the flow of inputs to output through a script.

These tasks can be fulfilled by **provenance.**

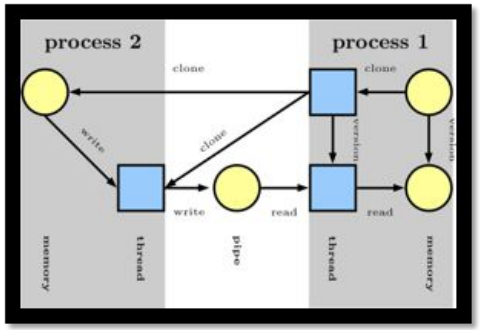
Provenance is an object's history represented as a directed acyclic graph (DAG), which consists of **nodes** and **edges**.

- This conceptual specification is defined by the **W3C PROV Data Model**[1] (a standardized model agnostic of level and implementation).

Application



Language



System

1. Khalid Belhajjame, Reza B'Far, james Cheney, Sam Coppens, Stephen Cresswell, Yolanda Gill, Paul Groth, Graham Klyne, Timothy Lebo, Jim McCusker, and et al. PROV-DM: The PROV Data Model.

Focus: **Language-Level Provenance (LL-prov)**

Scale: Line by line source-code level. For L-Vis, the language is **R**.

LL-Prov contains information about a past execution including:

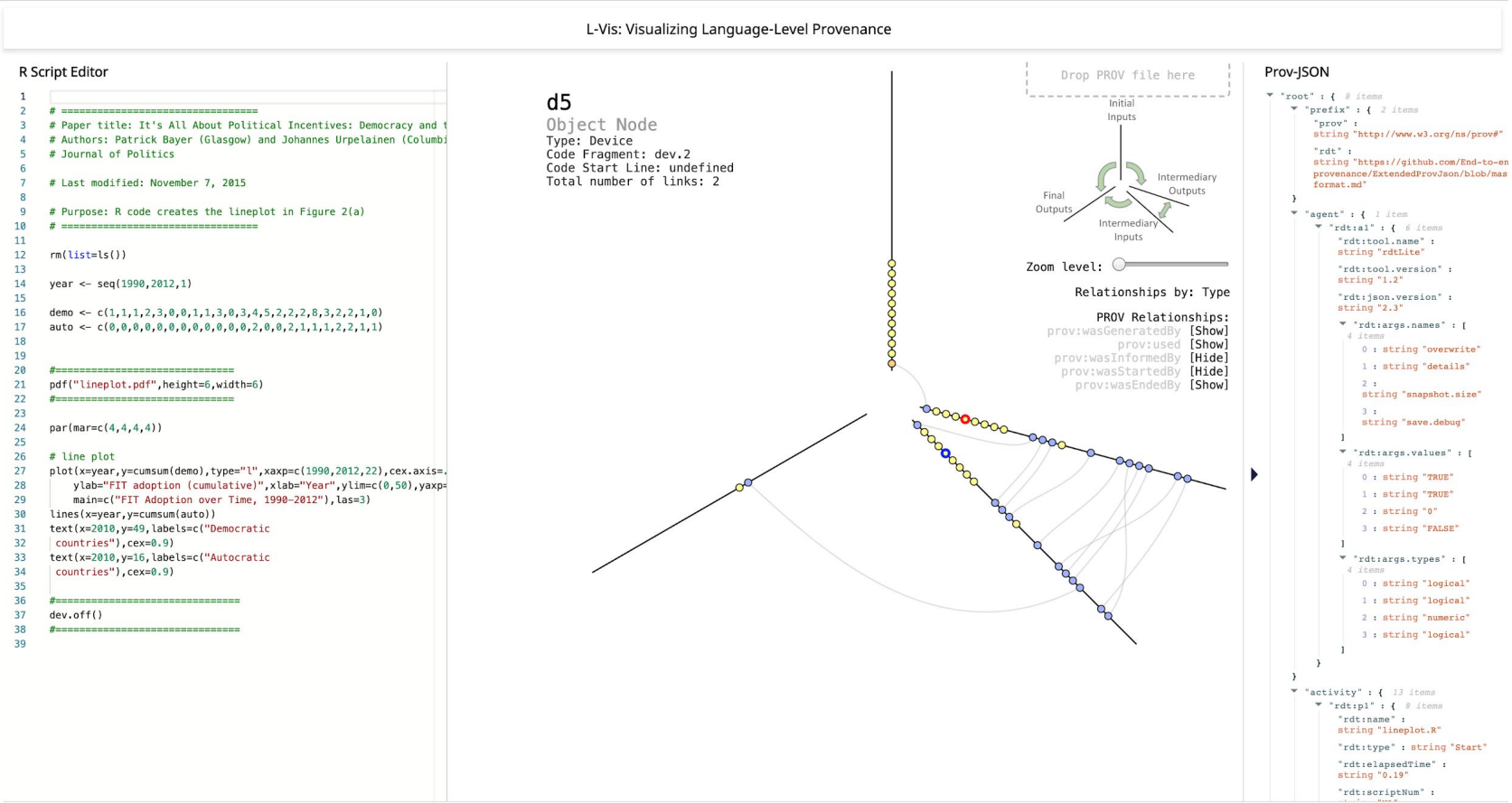
- External library dependencies.
- Function calls.
- How inputs interact to create outputs.

RDataTracker (RDT)[1] is an R package that collects LL-Prov.

- Generates PROV-JSON[2] file of nodes/edges which is a serialization of the PROV Data Model.
- PROV-JSON is the input to L-Vis.

1. B.S. Lerner and E.R. Boose. Rdatatracker and ddg explorer. In Ludäscher B., Plale B. (eds) Provenance and Annotation of Data and Processes (IPAW 2014), volume 8628, pages 288–290, 2015
2. Trung Dong Huynh, Michael O. Jewell, Amir Sezavar Keshavarz, Danius T. Michaelides, Huanjia Yang, and Luc Moreau. The PROV-JSON serialization, 2013.

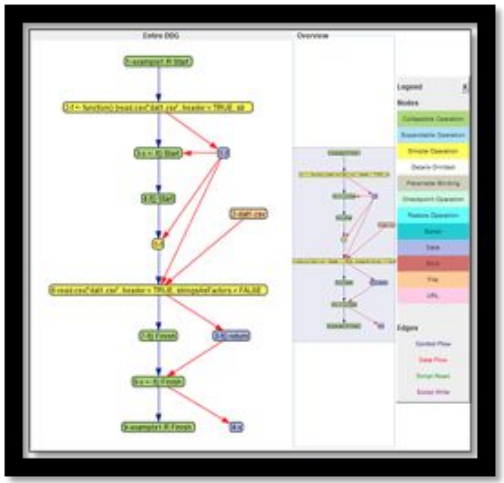
Our Tool: L-Vis



Motivation for Visualization of Provenance

- Not easy for a user to directly read the provenance and parse this information — without visualization, would have to parse lines of JSON.
- Typically visualized as a node-link graph, such as RDT's DDG Explorer.

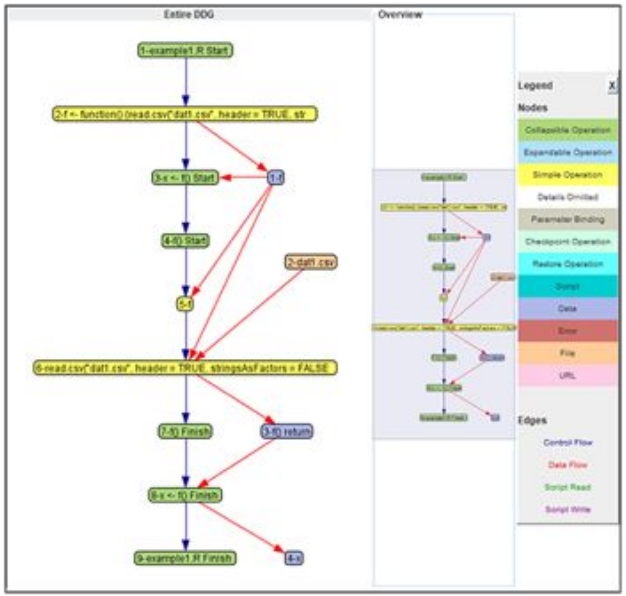

```
{
  "activity" : {
    "rdt:p3": {
      "rdt:name": "year <- seq(1990,2012,1)",
      "rdt:type": "Operation",
      "rdt:elapsedTime": "0.074",
      "rdt:scriptNum": 1,
      "rdt:startLine": 14, "rdt:startCol": 1,
      "rdt:endLine": 14, "rdt:endCol": 24
    }
  },

  "entity" : {
    "rdt:d1": {
      "rdt:name": "year",
      "rdt:value": "1990 1991 1992 1993 1994 1995 1996 1997 1998 1999 2000 2001 2002 2003 2004...",
      "rdt:valType": "{\"container\":\"vector\", \"dimension\":[23], \"type\":[\"numeric\"]}",
      "rdt:type": "Data",
      "rdt:scope": "R_GlobalEnv",
      ...
    },
    "rdt:environment": {
      "rdt:name": "environment",
```

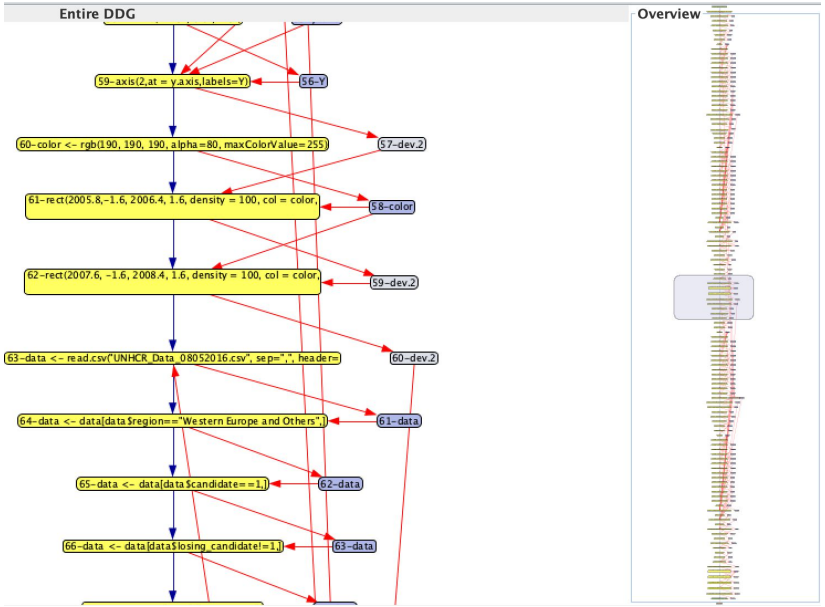
B.S. Lerner and E.R. Boose. Rdatatracker and ddg explorer. In Ludäscher B., Plale B. (eds) Provenance and Annotation of Data and Processes (IPAW 2014), volume 8628, pages 288–290, 2015

Issue: Scale

Images example of existing LL-Prov visualization tool: DDG Explorer.



2 lines of code
12 Nodes, 8 Edges



180 lines of code
267 Nodes, 435 Edges

B.S. Lerner and E.R. Boose. Rdatatracker and ddg explorer. In Ludäscher B., Plale B. (eds) Provenance and Annotation of Data and Processes (IPAW 2014), volume 8628, pages 288–290, 2015

Data Abstraction — Nodes / Edges

NODES

Prov Data Model		**PROV-JSON**		**Fully Abstracted**
"Entities"	→	Data node Library node Function node	→	**Objects:** Variables, external dependencies, files (I/O)
"Activities"	→	Procedure node	→	**Actions:** Executed code segment, typically a single line

EDGES

Indicate action chronology, object creation, and object usage

"Node (a) occurred before Node (b)"

Data Abstraction Cont.

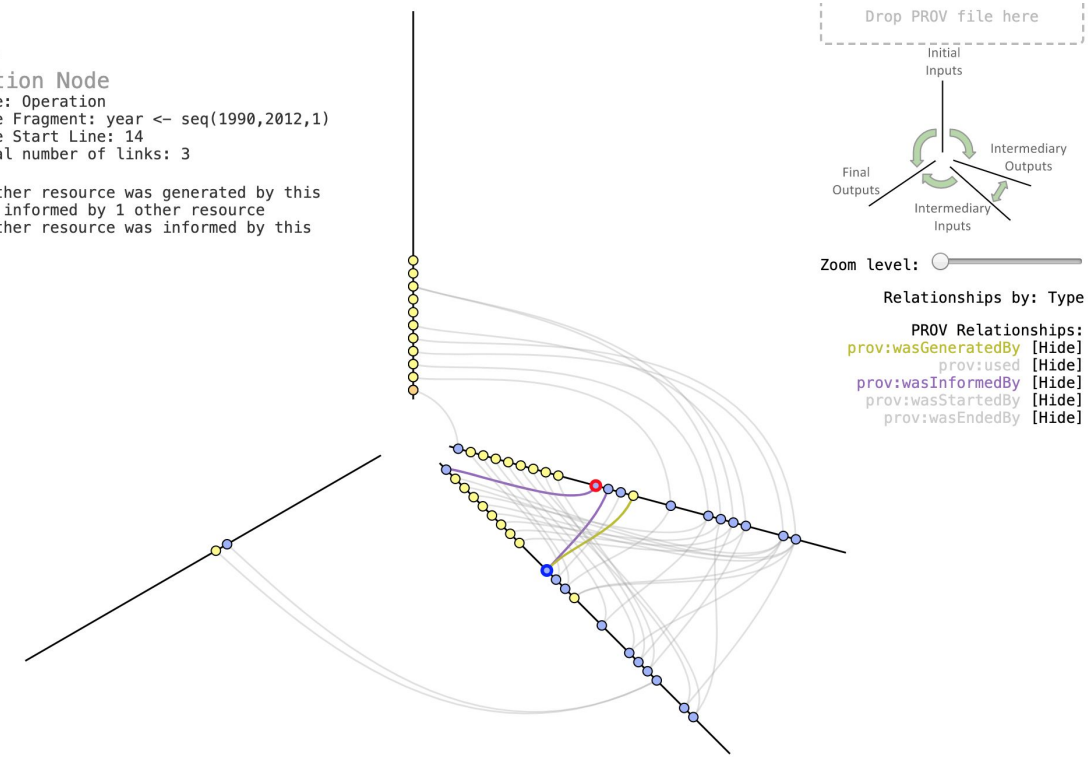
Fully Abstracted

Objects:
Yellow Nodes

Variables, external dependencies, files (I/O)

Actions:
Blue Nodes

Executed Code Segments

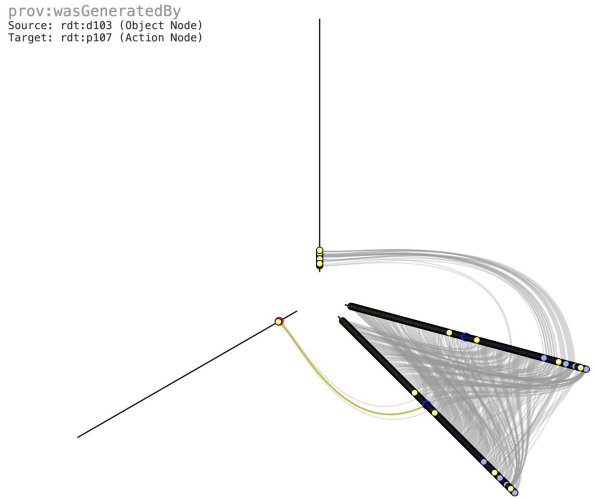


Data Abstraction Cont.

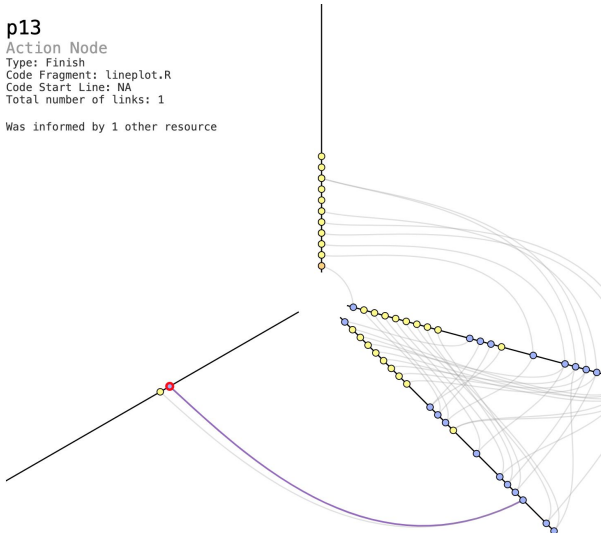
Derived Attribute

Crash Nodes:

An action that produces a new object by using two or more existing objects



Without crash nodes



With crash nodes

L-Vis Demo

Limitations

- Resources are limited. Not enough pixels, but also direction and hierarchy along hive plot axes have limitations.
- Are these the most elegant encodings? User study is necessary to evaluate effectiveness of idioms.
- L-Vis uses PROV-JSON as its input. Trade-Off: More specificity in schema, but code needs to be updated if schema is updated.

Future Work

In the very near future (for CPSC 508):

- Fully integrate L-Vis into existing reproducibility tool containR
- (Less rigorous) Qualitative study with our peers to evaluate L-Vis

On the horizon:

- Additional visualization idioms: Allow users to toggle traditional network layouts or radial layouts
- User interviews to derive tasks from users and get iterative feedback
- Quantitative study to compare L-Vis to other LL-prov tools

Questions?

L-Vis: Visualizing Language-Level Provenance