UP & DOWN: Improving Provenance Precision by Combining Workflow- and Trace-Land Information

By Khalid Belhajjame, Paris-Dauphine University, LAMSADE

In collaboration with:

Saumen Dey[1], David Koop[2], Tianhong Song[1], Bertram Ludeascher[1,] Paolo Missier[3]

[1] University of California

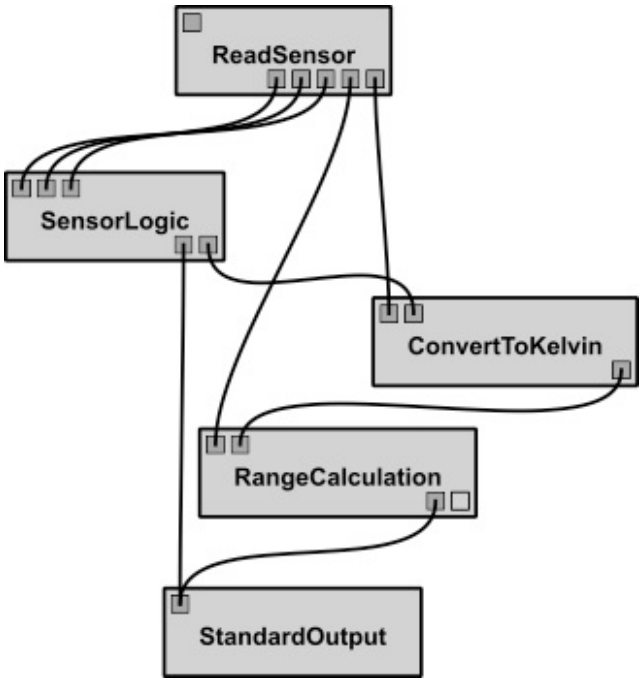
[2] New York University

[3] Newcastle University

usenix Publication
TaPP 2014
PAPERS
6th USENIX Workshop on the Theory and Practice of Provenance
June 12–13, 2014 • Cologne, Germany

Scientific Workflows

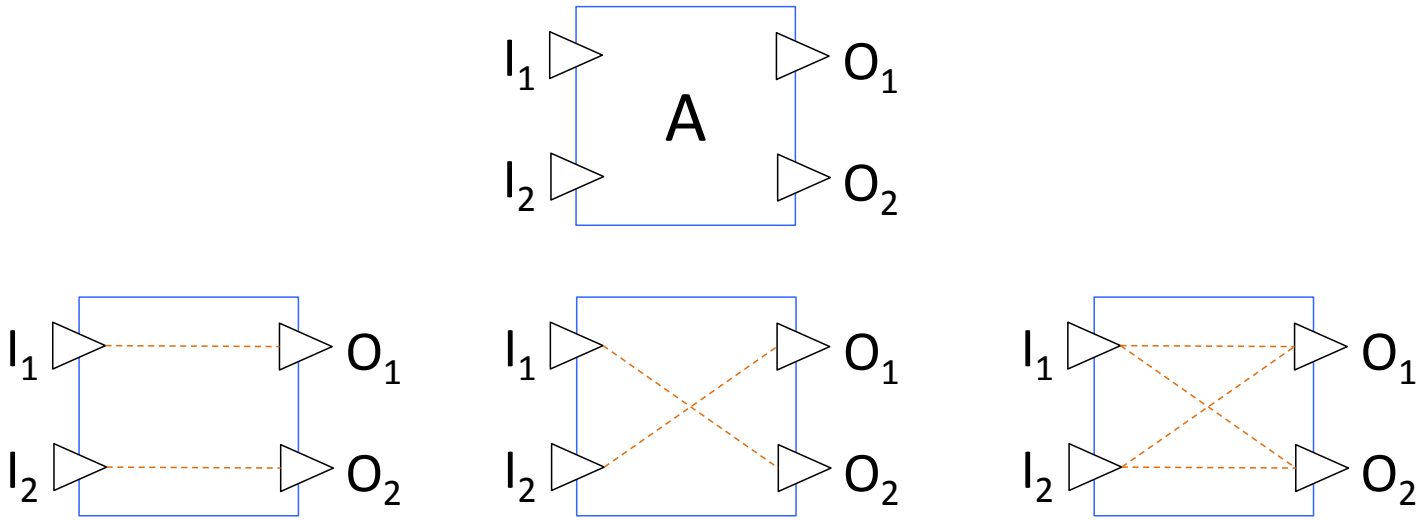
- We have recorded a dramatic increase in the number of scientist who utilize scientific modules as building in the composition of their experiments
  - In 2011, the EBI recorded 21 millions invocation to the scientific modules they host
- Typically, an experiment is designed as a workflow, the steps of which represent invocation to scientific modules, which we will refer to using the term actors.



Scientific Workflow and Provenance

- Many scientific workflows have been instrumented to capture workflow execution events.
  - Typically, such events record the actors that were invoked as part of the workflow execution as well as the data used as input and output by actor invocations.
- Collected executions events can be used, amongst other applications, to understand the lineage of a given workflow results.
- In practice, however, this application is limited by the black box nature of the actors implementing the steps of the workflow

Data Dependencies



- In general, if there are n input ports and m output ports for an actor, then there are 2^{mn} possible internal port dependencies.
- With k such actors in a workflow, there are 2^{kmn} possible port dependency models of the workflow.

Related Work: Looking Inside the Black Box

- Capturing Data Provenance using Dynamic Instrumentation (by Manolis Stamatogiannakis, Paul Groth, and Herbert Bos in IPAW 2014)
- DataTracker, which captures data provenance based on taint tracking, a technique used in the security and reverse engineering fields.

Framework for Generating Possible Data Dependency Models

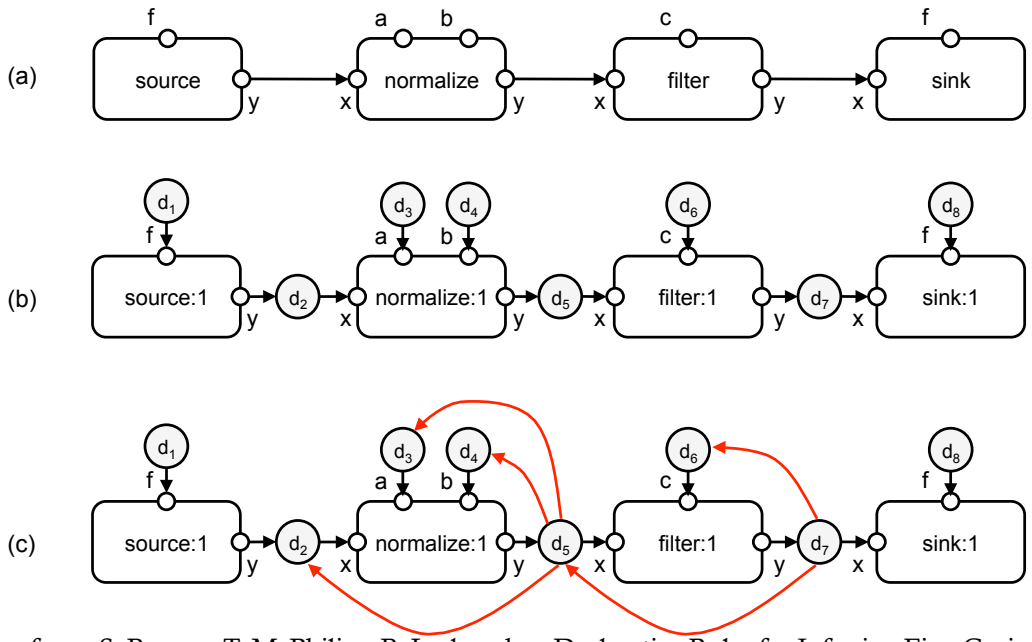
- We set out to build a framework for generating possible data dependency models within a workflow
- By dependency model, we mean a graph in which the nodes represent the input and output ports of the actors within the workflow, and the edges specify derivation dependencies between the ports in question.
- The objective is to reduce the space of possible port dependency models to those for which we have evidence that they hold.

Element of the Solution

1. Dependencies specified during workflow design: the workflow designer would indicates dependencies s/ he knows hold or not.
2. Infer data dependencies by examining the workflow provenance traces
3. Identify data dependencies through actor probing

1. Using Workflow Specifications

- The idea here is to use dependencies that are specified at the level of the workfow to infer derivation relationships at the level of the provenance graph

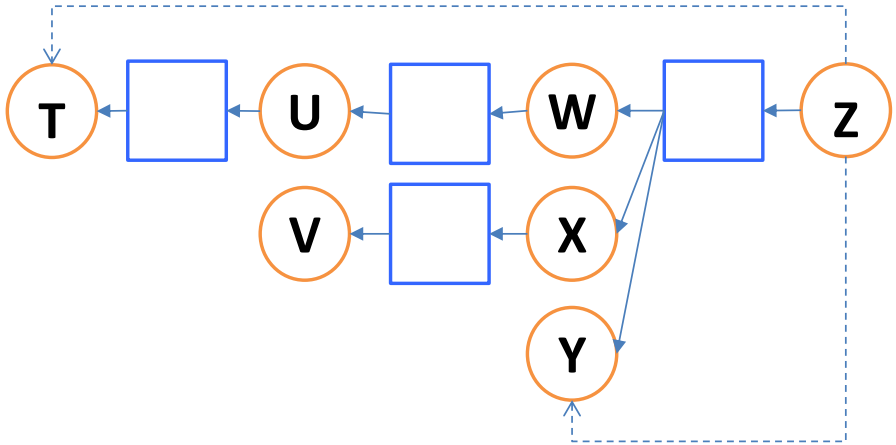


The Figure is taken from: S. Bowers, T. McPhilips, B. Ludeascher. Declarative Rules for Inferring Fine-Grained Data Provenance from Scientific Workfow Execution Traces. IPAW, 2012

2. Examining Provenance Traces

The idea here is to trawl provenance traces generated by the workflow, identify derivation relationships, which may have been manually specified by a user, and infer dependencies between actor ports.

2. Examining Provenance Traces



derBy(Z,Y) and derBy(Z,W)

We conclude that :

The output port associated with Z depends on the input ports associated with W and Y.

2. Examining Provenance Traces

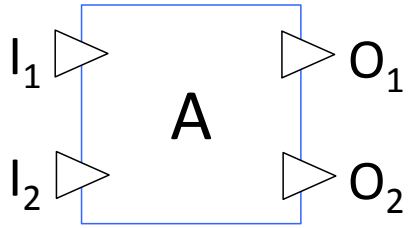
More generally, given provenance traces Gs, the derBy relations in Gs are analyzed

```
pdep(P1,P2) :-
    derBy(Y1,X1),data(X1,P1,A,R,_),
    data(Y1,P2,A,R,_).
```

3. Inferring Port Dependencies Through Actor Probing

- An additional source of information that we can use for inferring port dependencies is through actor probing.
- Given an actor **A** , to identify if the data values produced by a given output port of A depends on the data values used to feed a given input port of A, we probe the actor A by invoking it using different data values, and analyze the values delivered by the actor.

3. Inferring Port Dependencies Through Actor Probing



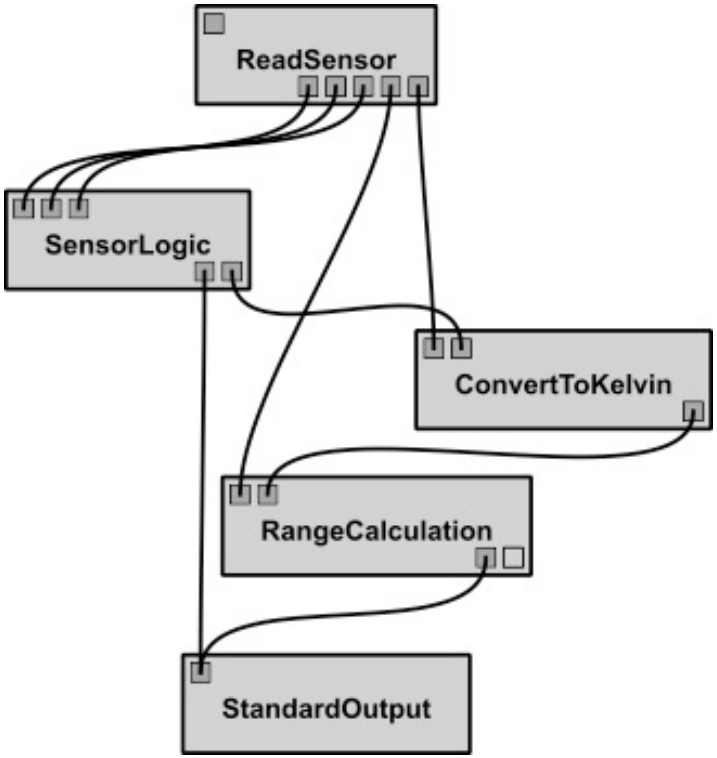
- Consider that the actor A has two input ports I_1 and I_2 and two output ports O_1 and O_2
- To determine if the data values delivered by the output O_1 depends on those used to feed the input I_2, we invoke the actor A using multiple pairs to feed its input ports I_1 and I_2:

$$\langle v_{I_1}, v^i_{I_2} \rangle, 1 \leqslant i \leqslant n$$

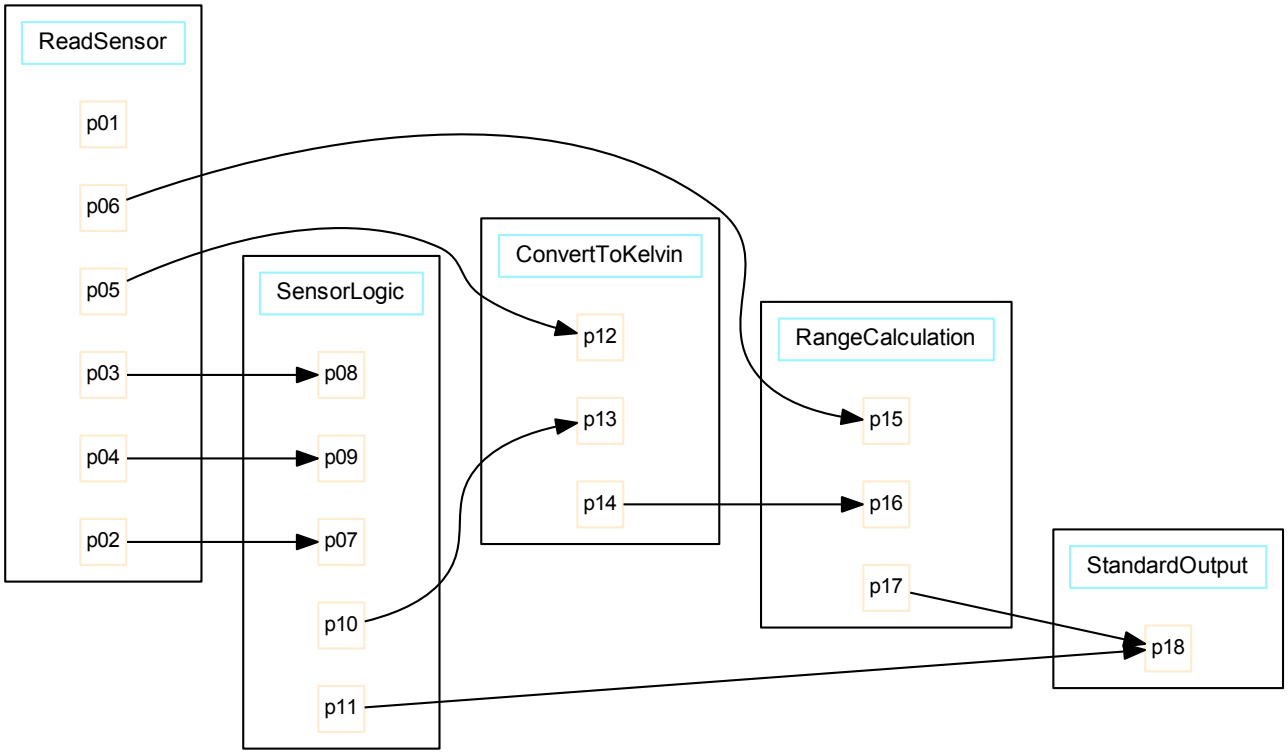
- If the output values of O_1 are the same for all invocations, then we conclude that the output O_1 *may* not depend on the input I_2.

Prototype

- We have an implementation of the framework which given a workflow specification and provenance traces of their execution generate possible port dependency models.
- Example: Climat Data Collection Workflow

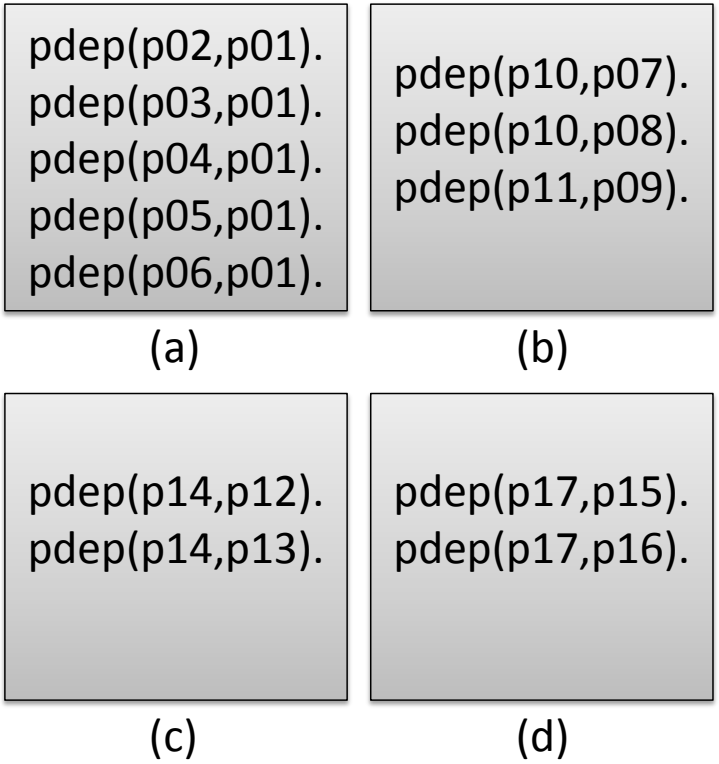


Climate Data Collector Workfow

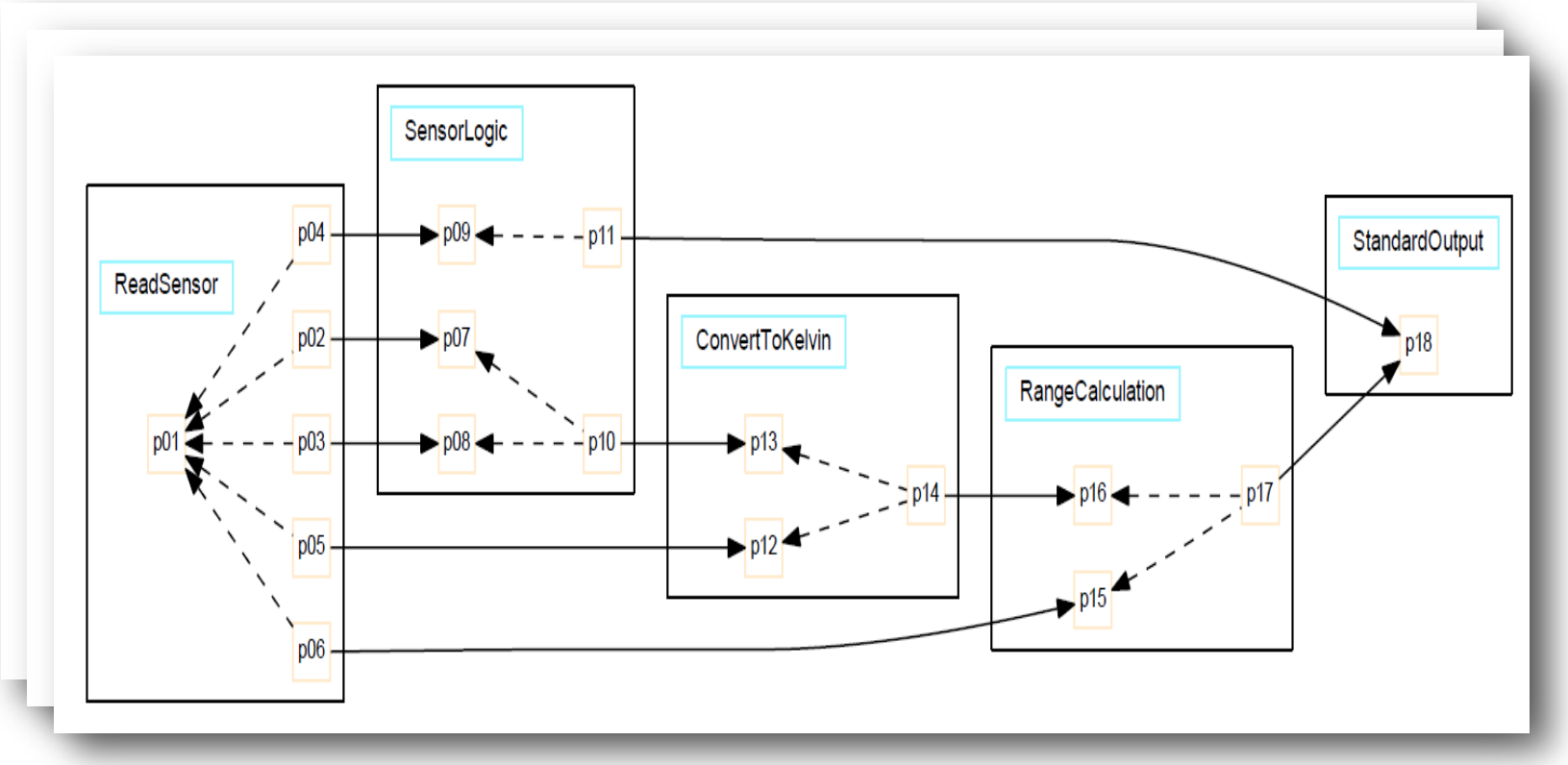


- Climate Data Collector workflow represented by our framework.
- There are **32768** possible dependency models for this workflow !

Climate Data Collector Workfow



Climate Data Collector Workfow



- Using our framwork, we reduced the number of possible dependency models to 6

Limitations and Future Work

- We assumed that the port dependencies are static, by which we mean that they are valid across runs.
  - This is not the case in general unfortunately!
  - The framework that we presented in this paper needs to be refined in order to cater for the identification of dynamic dependencies.
- we treat port dependencies equally.
  - A classification of dependencies is needed to provide the user with better understanding on the kind of relationship between the input and output ports.

Limitations and Future Work

- We did not consider Input port dependencies
  - A substantial number of actors (modules) have this kind of dependencies