

Causality and the Semantics of Provenance James Cheney
 Xiv : 1004.3241v1 [cs.PL] 19 Apr 2010 LFC S University of
 Edinburgh jcheney@inf.ed.ac.uk Provenance, or information
 about the sources, derivation, custody or history of data, has been
 studied recently in a number of contexts, including databases, scientific
 workflows and the Semantic Web. Many provenance mechanisms
 have been developed, motivated by informal notions such as influence,
 dependence, explanation and causality. However, there has been
 little study of whether these mechanisms formally satisfy appropriate
 policies or even how to formalize relevant motivating concepts such as
 causality. We contend that mathematical models of these concepts are
 needed to justify and compare provenance techniques. In this paper
 we review a theory of causality based on structural models that has
 been developed in artificial intelligence, and describe work in progress
 on a causal semantics for provenance graphs.

1 Introduction

Provenance is a general term referring to the origin, history, chain of custody,
 derivation or process that yielded an object. In analog settings such
 as art and archaeology, such information is essential for understanding
 whether an artifact is authentic, valuable or meaningful. In the digital
 world, provenance is now recognized as an important problem because
 it is very easy to silently alter or forge digital information. We already
 pay a price because of the lack of robust mechanisms for recording and
 managing provenance: serious economic losses have been incurred due
 to the lack of provenance on the Web [3], and lack of transparency
 of scientific processes and results is routinely used to sow confusion
 and doubt about climate change [25]. Much work on provenance
 considers the following basic scenario: we have some input data and
 some complex process that will be run on the input, for example a
 large program, possibly split into many smaller jobs and executed in
 parallel or on a distributed system. In this setting, a proposed solution
 generally records some additional information as the program runs. The
 additional information is often called provenance and it is supposed to
 provide an explanation showing how the results were obtained.
 Of course, this is a loose specification if we do not clarify what we
 mean by an explanation (apart from whatever provenance information
 happens to be recorded by the system). There are at least two obvious
 -seeming choices, neither of which seems satisfactory in practice: First
 , we might record the program that was run along with its input data
 (and any intermediate inputs such as user or network interactions).
 This, at least, allows us to re-run the program later and check that we
 get the same result, and it also allows us to vary the inputs to see how
 changes affect the output. But this is nearly useless as an explanation
 , especially for end-users who are not (and should not be expected to
 be) proficient at debugging black-box systems. Moreover, the inputs
 and outputs may be huge (for example, gigabytes of climate data),
 and it may not be possible for users to manually inspect the data.
 In the longer term, just recording the program is also problematic
 since the computational environment in which the program runs will
 change in some sense, this is also an input that we cannot feasibly
 record.

2 Second

, we might record everything that can be recorded about the computation,
 in the hope that it might someday be useful. This also sounds straightforward but is surprisingly
 difficult to pin down, since everything that can be recorded can be
 interpreted in many different ways. Should we record every function
 call? Every instruction? Every molecular interaction? Do we need to
 record what the programmer had for breakfast on the day the program
 was written? Clearly we have to stop somewhere, and for efficiency
 reasons we should probably stop far short of any reasonable definition
 of everything. Most extant approaches pick some intermediate point
 between these two extremes, committing to some (often not explicitly
 stated) choices about what is important about the computation that
 should be recorded in its provenance. For example, in databases,
 there are models such as where-provenance [2] (tracking the sources
 of copied data), lineage [10], why-provenance [2] or how-provenance
 [14] (tracking tuples used by or that justify a result tuple), or dependency
 provenance [7] (a computable approximation of the information flow
 behavior of the program). Provenance has also been studied extensively
 in other settings, particularly scientific workflow systems (e.g. [21, 20, 18]).
 Scientific workflows are usually high-level, visual programming languages,
 often based on data flow or Petri-net models of concurrent computation,
 and often executed on grid or cloud computing platforms. This architecture
 has the advantage that it puts considerable computational power into the
 hands of scientists without forcing them to learn how to program parallel
 or distributed systems at a low level in C++ or Java. However, it also
 has a serious drawback: distributing a program over a heterogeneous
 network dramatically increases the number of things that can go wrong,
 typically makes the computation nondeterministic and makes it hard
 for the user to trust the results. Scientists are reluctant to publish
 results based on programs that may contain subtle bugs, and whose
 behavior is different every time they are run, or depends on libraries or
 other environmental factors in subtle ways. Provenance is perceived as
 important for helping users understand whether results of such computations
 are repeatable and trustworthy, and in particular for scientists to
 be able to judge the scientific validity of results they may wish to publish.
 The work on database provenance is distinctive in that several different
 formal models have now been defined for database query languages with
 well-understood semantics. This makes it easier to compare, relate
 and generalize these approaches, though such comparisons are only
 starting to appear [8, 14]. For most of these models, there are semantic
 guarantees (or even exact semantic characterizations) relating the
 provenance records to the denotation of the program. On the other
 hand, for workflow provenance, formal definitions of the meaning
 of workflow programs have only started to appear recently (see for
 example [29, 17]), while the provenance semantics of these tools
 is usually specified informally, at best [21]. As a result there is a
 confusing variety of models and styles of provenance for workflows.
 To address this problem, there has been an ongoing community effort,
 centered on a series of Provenance Challenges [23], to understand
 and compare the qualitative behavior of these different systems and
 synthesize a common format for exchanging provenance among them.
 This effort has recently yielded a draft Open Provenance Model, or
 OPM [22]. Instances of this model are graphs whose nodes represent
 agents, processes or artifacts and whose edges represent dependence,
 generation or control relationships. The OPM has semantics in the
 sense of the Semantic Web, in that the nodes and edges are expected
 to have names that are meaningful to reasonably well-informed users.
 The OPM standard draws heavily on informal motivations such as
 provenance is the process that led to a result and edges denote
 causal relationships linking the cause to the effect. But while the OPM
 specifies a graph notation, controlled vocabulary for the edges, and
 inference rules for inferring new edges from existing edges, it
 does not have a semantics in the denotational or operational
 sense by which we might judge whether a graph is consistent or complete
 or whether inferences on the graph are valid. In this paper, we investigate
 the use of structural causal models [24] as a semantics for these graphs,
 and relate the informal motivations invoked in defining OPM graphs
 with the formal definitions of actual cause and explanation due to
 Halpern and Pearl [15, 16]. We do not argue that structural causal
 models provide the only or best causal account of provenance. However,
 structural causal models are quite close to OPM-style provenance graphs
 (modulo cosmetic differences), so the analogy is compelling. Moreover,
 structural models have been studied extensively (e.g. [24, 15, 16, 11, 12, 13]) and
 have proven useful to both philosophical accounts of scientific explanation
 [30] and psychological theories of understanding [27]. Nevertheless,
 it may be enlightening to apply other mathematical theories of causality
 and explanation to provenance, or investigate variations and extensions
 of Halpern and Pearl's approach. The broader aim of this paper is to
 argue by example that semantics (in the mathematical sense) is badly
 needed for research on provenance. One of the major motivations for
 provenance is to improve scientific communication, by allowing scientists
 to generate and exchange computational explanations or justifications
 of their results. In biology, for example, some journals now require
 both data and workflow programs describing how results were obtained,
 and some scientists anticipate that scientific publication will evolve into
 richer documents incorporating text, data, and computation [26].
 However, if the techniques used to do so are poorly specified and
 unverified then we can expect errors and confusion. Programming
 languages and semantics researchers can and should be involved in
 making sure that these techniques are clearly described and robust,
 to help ensure that scientific communications retain long-term value
 as they gain computational structure.

2 Examples

Before delving into technical details, we give a high-level example
 comparing OPM-style provenance graphs with structural causal models.
 The left-hand side of Figure 1 shows a simple OPM graph, based on a
 standard example showing the provenance of a cake [22]. The right-hand
 side shows a structural causal model, depicted as a graph. These two
 graphs are intentionally very similar. In the OPM graph, the ovals
 denote artifacts (i/s)