A Calculus for Propagating Semantic Annotations through Scientific Workflow Queries*

Shawn Bowers[1] and Bertram Ludäscher[1 2]

[1]UC Davis Genome Center [2]Department of Computer Science
University of California, Davis

Abstract. Scientific workflows facilitate automation, reuse, and reproducibility of scientific data management and analysis tasks. Scientific workflows are often modeled as dataflow networks, chaining together processing components (called *actors*) that query, transform, analyse, and visualize scientific datasets. Semantic annotations relate data and actor schemas with conceptual information from a shared ontology, to support scientific workflow design, discovery, reuse, and validation in the presence of thousands of potentially useful actors and datasets. However, the creation of semantic annotations is complex and time-consuming. We present a calculus and two inference algorithms to *automatically propagate* semantic annotations through workflow actors described by relational queries. Given an input annotation α and a query q, *forward propagation* computes an output annotation α'; conversely, *backward propagation* infers α from q and α'.

1 Introduction

Scientific workflows aim at automating repetitive scientific data management, analysis, and visualization tasks and provide scientists with a mechanism to seamlessly "glue" together different local and/or remote applications and (web) services into complex data analysis pipelines. Fig. 1 shows a simple ecology analysis workflow for computing two biodiversity quantities called Richness and Productivity using the KEPLER scientific workflow system [17]. As can be seen from the figure, scientific workflows are often modeled as networks of computational steps (called *actors*) that query, transform, and analyse input datasets (here, two datasets containing measurement data) via intermediate steps and derived datasets, resulting in a number of data products (here, containing the desired Richness and Productivity information). Scientific workflow systems (*e.g.*, KEPLER, TAVERNA [21], TRIANA [19] and many others [24]) are emerging as flexible and extensible problem-solving environments for designing, documenting, sharing, and executing scientific workflows [18]. In contrast to the use of shell scripts or spreadsheets, scientific workflows offer a versatile and controlled mechanism for automating data analysis pipelines, tracing data provenance [7,23], reproducing results, etc. As more and more workflow components and datasets become available, however, users

* Work supported by NSF/ITR SEEK (DBI-0533368), NSF/ITR GEON (EAR-0225673), and DOE SDM (DE-FC02-01ER25486). We also thank Alin Deutsch and Alan Nash for insightful discussions on the Chase and mapping composition, respectively.

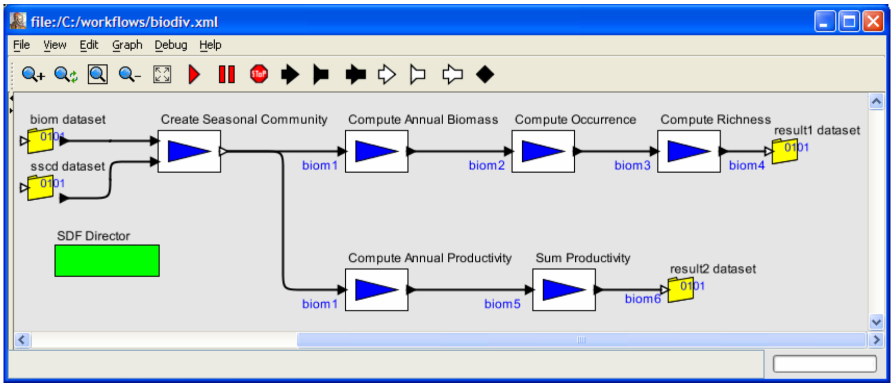


Fig. 1. Simple scientific workflow for computing species richness and productivity [9]

face the problem of selecting from thousands of possibly relevant workflow components (*e.g.*, given as web service operations, command-line tools, functions from statistics packages such as R, or native application components), and an even larger set of possible datasets. Similarly, once candidate actor components and datasets have been identified, there is the problem of whether it is possible to "chain" them together in the desired form. Generic programming language data types (such as `string` or arrays of integers) do *not* provide any guidance as to whether it is meaningful to chain together the output(s) of one actor with the input(s) of another actor. While the use of WSDL or XML Schema types can guide workflow composition, this requires that a single common schema is adopted, which is often impractical. To at least partially capture information about a dataset or analysis component, informal metadata annotations are often used in practice.

Example 1 (Informal Annotation) Consider a dataset D with the relational schema $S = \{\mathtt{R}(\mathtt{Obs}, \mathtt{La}, \mathtt{Lo}, \mathtt{T}, \mathtt{V})\}$. D might be given as a csv (comma-separated values) file, with an accompanying documentation saying that `R.Obs` identifies an *observation* at *time* `R.T`, conducted at a point having *latitude* and *longitude* `R.La` and `R.Lo`, respectively, and having as *value* `V`, which is a *temperature* measurement in degrees *celsius*. □

While such informal annotations are useful for the scientist when manually inspecting and interpreting data, a scientific workflow system cannot make use of this information, *e.g.*, to check whether the annotation of a dataset D is compatible with the annotation of a workflow actor A that consumes or produces D, or whether the output annotation of A_1 is compatible with the input annotation of A_2 in a chain of actors $(\cdots A_1 \xrightarrow{D} A_2 \cdots)$.

To address these problems, formal semantic annotations have been proposed [4,5]. A *semantic annotation* $\alpha\colon S \to \mathcal{O}$ associates elements of a data schema S with concepts and relationships of an ontology $\mathcal{O}$. [1] Thus, α can be seen as a "hybrid type" [5], linking structural information given by S with conceptual level ("semantic") information from a shared community ontology (or controlled vocabulary) $\mathcal{O}$.

[1] We consider ontologies expressed in description logic, *e.g.*, OWL-DL.

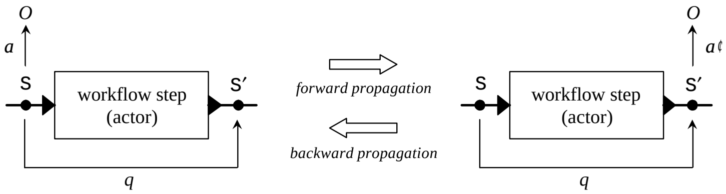


Fig. 2. Forward propagation $\alpha, q \rightsquigarrow \alpha'$ and backward propagation $\alpha', q \rightsquigarrow \alpha$

Example 2 (Semantic Annotation) Let $\mathcal{O}$ be an ontology defining relevant *concepts* of a particular community or domain (*e.g.*, $\mathsf{Observation}$ and Time) and *relationships* (*e.g.*, $\mathsf{hasUnit}$ and $\mathsf{hasLatitude}$) between concepts. The informal annotation above can be formalized by logic rules (constraints) of the form α: $S \to \mathcal{O}$:

$$\underbrace{\mathtt{R}(o, x, y, t, v)}_{\text{query over schema } S} \quad \longrightarrow \underbrace{\mathsf{Observation}(o) \wedge \mathsf{hasUnit}(o, \mathsf{celsius}) \wedge \mathsf{hasVal}(o, v) \wedge \mathsf{Time}(t)}_{\text{assertion over ontology } \mathcal{O}}$$

This rule states that $\mathtt{R}.o$ identifies an $\mathsf{Observation}$, having a unit $\mathsf{celsius}$, and a value $\mathtt{R}.v$, and that $\mathtt{R}.t$ is a Time. Similar rules α are used to map other columns or subsets of $\mathtt{R}$ to concepts and relationships in $\mathcal{O}$. □

By capturing semantic annotations as sets of logic rules α, a scientific workflow or data integration system can exploit these constraints, *e.g.*, for checking semantic type correctness of data-to-actor and actor-to-actor connections, and for suggesting semantically type-correct connections during workflow design [5].

In this paper we study the problem of automatically propagating a set of semantic annotations α "through" workflow actors which are described by relational queries q.[2] We consider the *forward propagation problem* $\alpha, q \rightsquigarrow \alpha'$ of deriving from an input annotation α: $S \to \mathcal{O}$ and a query q: $S \to S'$ an output annotation α': $S' \to \mathcal{O}$. Similarly, the *backward propagation problem* $\alpha', q \rightsquigarrow \alpha$ is to derive from knowledge of an output annotation α' and query q an input annotation α. The forward and backward propagation problems are summarized in Fig. 2.

Example 3 (Forward Propagation) Consider a simple actor A that selects from the above dataset $\mathtt{R}$ (input schema) only those observations with temperature measurements below $0°\mathrm{C}$ and locations that fall into a particular region of interest $\mathtt{R}_{roi}(x, y)$ resulting in a dataset $\mathtt{R}'$ (output schema). We can describe A with a query q as follows:

$$\mathtt{R}'(o, v) \;{:}\!-\; \mathtt{R}(o, x, y, t, v), \mathtt{R}_{roi}(x, y), v < 0. \qquad (q)$$

Given the semantic annotation α of the actor input $\mathtt{R}$ in Example 2, the forward problem is to *automatically* derive an output annotation α' for $\mathtt{R}'$. Here, we obtain

$$\mathtt{R}'(o, v) \longrightarrow \mathsf{Observation}(o) \wedge \mathsf{hasUnit}(o, \mathsf{celsius}) \wedge \mathsf{hasVal}(o, v) \qquad (\alpha')$$

[2] The actor may not be implemented as a relational query q. Instead, q is another form of metadata, a *query annotation*, which describes or approximates an actor's workflow function.

since we "know" from α and q that v is the value of an observation o with unit $\mathsf{celsius}$. Note that to infer this α', we use the "only if" direction $q_{head} \rightarrow q_{body}$ of the (Datalog) rule q_{head} **:** $-$ q_{body} defining the query q above.[3] Since annotations α and α' have a particular form (source-to-ontology constraints in a language $\mathcal{L}^{\alpha}_{S\rightarrow\mathcal{O}}$), α' may not include all deducible information: *e.g.*, here we omit in the consequent of α' the fact that $v < 0$ and possibly other information about $\mathtt{R}$ and $\mathtt{R}_{roi}$ (as these are not $\mathcal{O}$ expressions). □

The manual creation of semantic annotations can be a complex and time-consuming task. Thus providing automated solutions for deriving annotations is desirable for supporting scalable frameworks of "semantics-aware" scientific workflows. In addition, solving the propagation problem also provides new opportunities for *semantic type checking*: if both an input annotation α and an output annotation α' for an actor are given, then employing forward and backward propagation allows us to check the consistency of the given annotations relative to the inferred ones.

Contributions and Previous Work. We first present a formal framework for semantic annotations α and define the associated forward and backward propagation problems. We then present inference rules of our annotation propagation calculus (APC) and two general propagation algorithms f-APC and b-APC for forward and backward propagation, respectively. These algorithms proceed by structural induction on the operator tree corresponding to a relational query q. We use such queries q to represent individual actors of a workflow. An advantage of the APC approach is that it can be "scaled" to different query languages $\mathcal{L}^q$, *i.e.*, for certain query classes we obtain the exact (*i.e.*, most specific) annotation as the propagation result, but we also obtain results (not necessarily most specific) for more expressive classes $\mathcal{L}^q$ for which no exact solution exists.

We introduced semantic annotations in [4,5] to facilitate scientific data integration and workflow design and composition, and proposed to automatically propagate such annotations through workflows [6]. This paper extends our previous work [6] in several ways: (i) by considering both forward and backward propagation, (ii) by introducing the f-APC and b-APC algorithms for annotation propagation, and (iii) by considering propagation challenges in terms of the annotation and query languages used.

The f-APC and b-APC algorithms employ a specific goal-directed resolution strategy for annotation propagation. Similary, goal directed resultion strategies are also used for solving schema mapping composition problems [13,20] as well as query rewriting problems in data integration/exchange settings, where the corresponding rewriting techniques can be understood as specialized versions of resolution [3], but for which certain termination and efficiency guarantees can be given (unlike for general resolution).

Other Related Work. Annotation mechanisms in other related work are generally more restricted than our approach in that they consider only single attribute or value annotations. In [2], annotations are stored in a special attribute, and the user must specify how to propagate such value-based annotations through SQL queries, while we are able to capture more expressive schema-based annotations, and also propagate them automatically. Our approach also does not require a special semantics for interpreting

[3] This is correct, since the symbol '**:** $-$' stands for an equivalence $q_{head} \leftrightarrow q_{body}$, corresponding to a sound and *complete* definition of the query answer (*a.k.a.* "Clark's completion" [10]).

relational algebra queries. [14] present an approach to scientific annotations which allows *value associations* (as opposed to annotations to individual values only [2]). Such associations between different schema columns can be easily expressed in our approach as conjunctive conditions in the body of α.

Propagating annotations is also related to the issue of (*why* and *where*) data provenance [7]. For example, [8] present an approach to propagate annotations through views, but consider again simple text-based instance (*i.e.*, value) annotations. In contrast, our annotations are applied at the schema level, but can also specify subsets of the input data (through the "query part", *i.e.*, the body of α), including individual values just like previous approaches. In [11] methods are presented for lineage tracing of data (within the context of data warehouses), which take advantage of known structure or properties of transformations, similar to our queries q. The lineage problem and propagation methods considered in [11] are related but different from our approach. For example, in their case, for first-order (relational) queries, an exact data lineage can be computed. Conversely, in general, the problem of propagating a semantic annotation through a first-order query may not have a solution in the desired annotation language. This indicates that propagating semantic annotations is in general harder than computing data lineage. The problem of propagation is also related to type inference in programming languages, where types are generally given in less expressive langauges (*e.g.*, compared to dependency constraints) but programs are written in more expressive languages (*e.g.*, compared to relational algebra queries).

2 Formalization of the Annotation Propagation Problem

Here we present our framework for semantic annotations and show how the propagation problem can be formalized and reduced to a constraint implication problem.

Scientific Workflows. These are often modeled as *dataflow process networks* [15,16],[4] consisting of a set of computational components called *actors*, which can run as independent processes or threads, and which exchange data tokens (*e.g.*, scalars, vectors, files, XML fragments, etc.) through unidirectional, buffered FIFO *channels*. Channels connect *output ports* of source actors with *input ports* of target actors (*cf.* Fig. 1).

Mappings. A schema *mapping* is a binary relation[5] on instances D_S, $D_{S'}$ of disjoint schemas S (input) and S' (output). Given S, S', and a set of (logic) constraints Σ, we associate with (S, S', Σ) the mapping $m = \{ \langle D_S, D_{S'} \rangle \mid (D_S \cup D_{S'}) \models \Sigma \}$, *i.e.*, the set of pairs $\langle D_S, D_{S'} \rangle$ of instances of S and S' for which the combined instance $D_S \cup D_{S'}$ satisfies Σ. For example, a *query* q corresponds to a (functional) mapping m_q: (S, S', Σ_q). We write q: $S \to S'$ to emphasize the input/output signature of q.

Actor Schemas and Semantic Annotations. With the input and output ports of an actor A, we associate two disjoint relational schemas, S and S', describing the input and

[4] Many scientific workflow systems (*e.g.*, INFORSENSE, KEPLER, PIPELINEPILOT, TAVERNA, TRIANA), scientific problem-solving environments (*e.g.*, SCIRUN), and commercial LIMS systems (*e.g.*, LABVIEW) are based on this dataflow process network model.

[5] We follow the convention to call such relations "mappings" [1,13,20], although they are *not* (functional) mappings in the traditional sense: *e.g.*, unlike conventional mappings, a (non-functional) "mapping" $\langle D_S, D_{S'} \rangle$ always has an inverse "mapping" $\langle D_{S'}, D_S \rangle$.

output data structures of A, respectively. The input/output behavior of A is described (or approximated) via a query q: $S \to S'$, mapping instances of the input schema S to instances of the output schema S' (see Fig. 2). An instance D_S of a schema S is called a *dataset*. A *semantic annotation* α: $S \to \mathcal{O}$ is a mapping from instances of a (concrete) data schema S to instances of an (abstract, virtual) ontology $\mathcal{O}$. Here, an ontology is a (finite) first-order structure. Given a query q: $S \to S'$ for an actor A, we call α: $S \to \mathcal{O}$ an *input annotation*, and α': $S' \to \mathcal{O}$ an *output annotation* of A (Fig. 2). Using the above notation, a semantic annotation α corresponds to a mapping m_α: $(S, \mathcal{O}, \Sigma_\alpha)$, where Σ_α is a set of logic constraints capturing α.

Annotation Propagation as Composition of Mappings. The forward propagation problem $\alpha, q \rightsquigarrow \alpha'$ can be seen as a mapping composition problem [13,20]. The given signatures α: $S \to \mathcal{O}$, q: $S \to S'$, and α': $S' \to \mathcal{O}$ suggest the definition

$$\alpha' := \alpha(q^-) \qquad \text{(f-prop)}$$

i.e., obtain the propagated annotations as the composition α': $(S', \mathcal{O}, \Sigma_{\alpha'})$ of α and q^-. Here q^-: (S', S, Σ_q) is the *inverse mapping* of q, which is exactly like q but with the roles of inputs and outputs reversed. Similarly, for the backward propagation:

$$\alpha := \alpha'(q) \qquad \text{(b-prop)}$$

one could apply α': $(S', \mathcal{O}, \Sigma_{\alpha'})$ on top of q: (S, S', Σ_q), resulting in α: $(S, \mathcal{O}, \Sigma_\alpha)$.

To view annotation propagation in this way as mapping composition helps to understand some aspects of the subsequent annotation propagation calculus (APC) rules and inference algorithm: *e.g.*, in the foward case we are looking for a constraint α': $S' \to \mathcal{O}$. Given α: $S \to \mathcal{O}$ and q: $S \to S'$, we can "go" from S' to $\mathcal{O}$ by first applying q in the inverse direction q': $S' \to S$, then applying α: $S \to \mathcal{O}$ to the result (cf. Fig. 2), hence we can think of the propagated result as $\alpha' = \alpha(q^-)$. By similar reasoning, we obtain $\alpha = \alpha'(q)$ for the backward propagation.

The Annotation Propagation Problem. We now formally define the annotation propagation problem by relating it to constraint implication as follows:

Definition 1 Consider a semantic annotation α: $(S, \mathcal{O}, \Sigma_\alpha)$ expressed in an annotation language $\mathcal{L}^\alpha$, and a query q: (S, S', Σ_q). Let q^-: (S', S, Σ_{q^-}) be the inverse of q.

We say that α': $(S', S, \Sigma_{\alpha'})$ is a *forward $\mathcal{L}^\alpha$-propagation* of α *through* q, if $\Sigma_{\alpha'}$ is the *most specific* annotation in $\mathcal{L}^\alpha$ that is implied by Σ_α and Σ_{q^-}, denoted $\Sigma_\alpha \cup \Sigma_{q^-} \models \Sigma_{\alpha'}$. We say α_1 is *more specific* than α_2, if $\Sigma_{\alpha_1} \models \Sigma_{\alpha_2}$. The *backward $\mathcal{L}^\alpha$-propagation* is defined analogously: Find the most specific $\Sigma_\alpha \subseteq \mathcal{L}^\alpha$ with $\Sigma_{\alpha'} \cup \Sigma_q \models \Sigma_\alpha$. □

This formalization has several advantages: First, under this propagation semantics, a result annotation may exist even in cases where the mapping composition semantics cannot be expressed in the constraint language of choice. Second, this formalization naturally applies to inference-based (logic) approaches like the APC below.

3 Annotation Propagation Calculus (APC)

We first present the basic annotation propagation calculus (APC), then present two goal-directed inference algorithms f-APC and b-APC for forward and backward propagation.

$$\begin{array}{llll}
(\sigma) & \forall \mathbf{x}\, R(\mathbf{x}) \wedge \psi(\mathbf{x}) \rightarrow S'(\mathbf{x}) & (\sigma^-) & \forall \mathbf{x}\, S'(\mathbf{x}) \rightarrow R(\mathbf{x}) \wedge \psi(\mathbf{x}) \\
(\pi) & \forall \mathbf{x} \forall \mathbf{y}\, R(\mathbf{x},\mathbf{y}) \rightarrow S'(\mathbf{x}) & (\pi^-) & \forall \mathbf{x}\, S'(\mathbf{x}) \rightarrow \exists \mathbf{y}\, R(\mathbf{x},\mathbf{y}) \\
(\times) & \forall \mathbf{x} \forall \mathbf{y}\, R_1(\mathbf{x}) \wedge R_2(\mathbf{y}) \rightarrow S'(\mathbf{x},\mathbf{y}) & (\times^-) & \forall \mathbf{x} \forall \mathbf{y}\, S'(\mathbf{x},\mathbf{y}) \rightarrow R_1(\mathbf{x}) \wedge R_2(\mathbf{y}) \\
(\setminus) & \forall \mathbf{x}\, R_1(\mathbf{x}) \wedge \neg R_2(\mathbf{x}) \rightarrow S'(\mathbf{x}) & (\setminus^-) & \forall \mathbf{x}\, S'(\mathbf{x}) \rightarrow R_1(\mathbf{x}) \wedge \neg R_2(\mathbf{x}) \\
(\cup) & \forall \mathbf{x}\, R_1(\mathbf{x}) \vee R_2(\mathbf{x}) \rightarrow S'(\mathbf{x}) & (\cup^-) & \forall \mathbf{x}\, S'(\mathbf{x}) \rightarrow R_1(\mathbf{x}) \vee R_2(\mathbf{x})
\end{array}$$

a) q: $S \rightarrow S'$ direction for b-APC　　b) q^-: $S' \rightarrow S$ direction for f-APC

Fig. 3. Relational algebra operators (atomic queries) expressed as logic constraints

Query Operators as Logic Constraints. The core idea of APC is the observation that annotation propagation is easy for primitive (atomic) query operators. Therefore, we start by representing a complex (first-order) query q_c in the form of a relational algebra expression, or equivalently, as an operator tree, consisting of atomic query operators $q \in \{\sigma, \pi, \times, \setminus, \cup\}$. Each relational operator q defines a mapping q: (S, S', Σ_q) for the "standard" (*i.e.*, forward) direction $S \rightarrow S'$ of q, and an inverse mapping q^-: (S', S, Σ_{q^-}) for the opposite (*i.e.*, backward) direction $S' \rightarrow S$. Here, Σ_q and Σ_{q^-} are as defined in Fig. 3(a) and Fig. 3(b), respectively.

3.1 Inference Rules of APC

The formalization of annotation propagation using logic constraints (see Definition 1), suggests a natural inference procedure for the forward problem $\alpha, q \rightsquigarrow \alpha'$, *i.e.*, by "applying" the constraints Σ_α to Σ_{q^-}, thus obtaining $\Sigma_{\alpha'}$. Similarly, one can combine Σ_q and $\Sigma_{\alpha'}$ to obtain Σ_α to solve the backward problem. This is the core idea behind the APC inference rules.

Fig. 4 shows the rules for backward propagation, which take an output annotation α' and an atomic query operator q and infer the input annotation α. Similary, Fig. 5 shows how forward propagation is solved by applying the input annotation α on the inverse query q^- to obtain the output annotation α'. The inference rules in both figures are depicted with their *premises* above the horizontal line, and their *consequent*(s) below the line. Each inference rule corresponds to an algebra operator q: $S \rightarrow S'$ or its inverse q^-: $S' \rightarrow S$. Moreover, with every rule for q (in b-APC) and q^- (in f-APC), we associate at least one *goal atom*, marked as $[\![A]\!]$ in the head of q (Figure 4) or the head of q^- (Figure 5). The basic idea of annotation propagation is to "resolve" the goal atom in q (or in q^-) with some literal of the given semantic annotation α' (or α), to obtain the desired propagated annotation.

Applying Substitutions. To simplify the exposition of the rules in Fig. 4 and Fig. 5, the application of unifiers is not shown but implicit. More precisely, let θ be an *mgu* (most-general-unifier) of a goal atom $[\![R(\mathbf{u})]\!]$ in q (or q^-) with a corresponding atom $R(\mathbf{x})$ on the left-hand side of an annotation α' (or α). The unifier θ is a (most general) substitution under which both atoms become identical, *i.e.*, $\theta(R(\mathbf{u})) = \theta(R(\mathbf{x}))$. In the figures, with the b-APC and f-APC rules, we assume that this θ is applied to the consequent rule (below the line) to obtain the propagated annotation.

$$B_\sigma\ \begin{array}{rl} \alpha'\!: & S'(\mathbf{x}),\ \varphi(\mathbf{x}) \rightarrow \exists \mathbf{z}\, \omega(\mathbf{x},\mathbf{z}) \\ q\!: & R(\mathbf{u}),\ \psi(\mathbf{u}) \rightarrow \llbracket S'(\mathbf{u}) \rrbracket \\ \hline \alpha\!: & R(\mathbf{u}),\ \psi(\mathbf{u}),\ \varphi(\mathbf{x}) \rightarrow \exists \mathbf{z}\, \omega(\mathbf{x},\mathbf{z}) \end{array} \qquad B_\pi\ \begin{array}{rl} \alpha'\!: & S'(\mathbf{x}),\ \varphi(\mathbf{x}) \rightarrow \exists \mathbf{z}\, \omega(\mathbf{x},\mathbf{z}) \\ q\!: & R(\mathbf{u},\mathbf{v}) \rightarrow \llbracket S'(\mathbf{u}) \rrbracket \\ \hline \alpha\!: & R(\mathbf{u},\mathbf{v}), \varphi(\mathbf{x}) \rightarrow \exists \mathbf{z}\, \omega(\mathbf{x},\mathbf{z}) \end{array}$$

$$B_\times\ \begin{array}{rl} \alpha'\!: & S'(\mathbf{x},\mathbf{y}),\ \varphi(\mathbf{x},\mathbf{y}) \rightarrow \exists \mathbf{z}\, \omega(\mathbf{x},\mathbf{y},\mathbf{z}) \\ q\!: & R_1(\mathbf{u}),\ R_2(\mathbf{v}) \rightarrow \llbracket S'(\mathbf{u},\mathbf{v}) \rrbracket \\ \hline \alpha\!: & R_1(\mathbf{u}),\ R_2(\mathbf{v}),\ \varphi(\mathbf{x},\mathbf{y}) \rightarrow \exists \mathbf{z}\, \omega(\mathbf{x},\mathbf{y},\mathbf{z}) \end{array} \qquad B_\cup\ \begin{array}{rl} \alpha'\!: & S'(\mathbf{x}),\ \varphi(\mathbf{x}) \rightarrow \exists \mathbf{z}\, \omega(\mathbf{x},\mathbf{z}) \\ q\!: & R_1(\mathbf{u}) \rightarrow \llbracket S'(\mathbf{u}) \rrbracket \\ & R_2(\mathbf{u}) \rightarrow \llbracket S'(\mathbf{u}) \rrbracket \\ \hline \alpha\!: & R_1(\mathbf{u}),\ \varphi(\mathbf{x}) \rightarrow \exists \mathbf{z}\, \omega(\mathbf{x},\mathbf{z}) \end{array}$$

$$B_\setminus\ \begin{array}{rl} \alpha'\!: & S'(\mathbf{x}),\ \varphi(\mathbf{x}) \rightarrow \exists \mathbf{z}\, \omega(\mathbf{x},\mathbf{z}) \\ q\!: & R_1(\mathbf{u}),\ \neg R_2(\mathbf{u}) \rightarrow \llbracket S'(\mathbf{u}) \rrbracket \\ \hline \alpha\!: & R_1(\mathbf{u}),\ \neg R_2(\mathbf{u}),\ \varphi(\mathbf{x}) \rightarrow \exists \mathbf{z}\, \omega(\mathbf{x},\mathbf{z}) \end{array}$$

Fig. 4. Backward propagation (b-APC) rules for $\alpha', q \rightsquigarrow \alpha$

Additional Remarks for f-APC. For f-APC we assume that annotations α and the constraint q_π^-, capturing the inverse of relational projection π, have been "skolemized" (replacing $\exists$-quantified variables by symbolic identifiers while keeping track of the $\forall$-variables they depend on). In the rule F_σ, we denote by $(\varphi(\mathbf{x}) \land \neg\psi(\mathbf{u}))^*$ that φ has ψ "factored out", *i.e.*, we simplify $\varphi \land \neg\psi$.

3.2 Operator-Driven Annotation Propagation in APC

The above APC rules for forward and backward propagation based on atomic query operators induce two natural inference algorithms for complex queries, *i.e.*, consisting of nested expressions of operators. The approach is to drive the application of inference rules by the structure of the operator tree of a given complex query q. To illustrate this structural induction over the operator tree, consider first the backward problem $\alpha', q \rightsquigarrow \alpha$. Let q: $S \rightarrow S'$ be a complex query, expressed as a nested relational algebra expression of unary or binary operators q_i: $q = q_n(q_{n-1}(\cdots q_1 \cdots))$ where q_n corresponds to the top-most (root) node of the operator tree, and leaf nodes (such as q_1) are applied to the input relations of q (cf. Fig. 6). Recall the "composition solution" $\alpha := \alpha'(q)$ to the backward problem (see (b-prop) on page 6). Applying α' to the nested expression for q yields $\alpha = \alpha'(q) = \alpha'(q_n(q_{n-1}(\cdots)))$. As mapping composition is associative, we can first apply α' to q_n (the root node), obtaining an intermediate annotation α_1, which is applied to q_{n-1}, yielding α_2, which is further propagated downward in the tree, etc. This process is repeated until the leaf nodes are reached. Fig. 6 (a) illustrates this top-down process for b-APC.

Similarly, for the forward problem $\alpha, q \rightsquigarrow \alpha'$, we have the "composition solution" (f-prop) of the form $\alpha' := \alpha(q^-)$. First note that the "inverse reading" of the operator tree can be seen as an expression $q^- = q_1^-(q_2^-(\cdots))$ in which the root node q_n becomes an innermost node. Applying α to this expression, and again exploiting associativity, we obtain a bottom-up annotation propagation for f-APC: Fig. 6 (b) depicts this process.

Strictly speaking, the notation (nested expressions) used for q and q^- above were based on unary operators. However, it should be clear how the top-down approach for b-APC and the bottom-up approach for f-APC work in the case of binary operators.

$$
F_\sigma\;\begin{array}{l}
\alpha\!:\; R(\mathbf{x}),\; \varphi(\mathbf{x}) \rightarrow \omega(\mathbf{x},\mathbf{f}(\mathbf{x}))\\
q^-\!:\; S'(\mathbf{u}) \rightarrow \llbracket R(\mathbf{u})\rrbracket,\; \psi(\mathbf{u})\\
\hline
\alpha'\!:\; S'(\mathbf{u}),\; (\varphi(\mathbf{x}) \wedge \neg\psi(\mathbf{u}))^* \rightarrow \omega(\mathbf{x},\mathbf{f}(\mathbf{x}))
\end{array}
\qquad
F_\pi\;\begin{array}{l}
\alpha\!:\; R(\mathbf{x}),\; \varphi(\mathbf{x}) \rightarrow \omega(\mathbf{x},\mathbf{f}(\mathbf{x}))\\
q^-\!:\; S'(\mathbf{u}) \rightarrow \llbracket R(\mathbf{u},\mathbf{g}(\mathbf{u}))\rrbracket\\
\hline
\alpha'\!:\; S'(\mathbf{u}),\; \varphi(\mathbf{x}) \rightarrow \omega(\mathbf{x},\mathbf{f}(\mathbf{x}))
\end{array}
$$

$$
F_\times\;\begin{array}{l}
\alpha\!:\; R_1(\mathbf{x}),\; \varphi(\mathbf{x}) \rightarrow \omega(\mathbf{x},\mathbf{f}(\mathbf{x}))\\
q^-\!:\; S'(\mathbf{u},\mathbf{v}) \rightarrow \llbracket R_1(\mathbf{u})\rrbracket,\; R_2(\mathbf{v})\\
\hline
\alpha'\!:\; S'(\mathbf{u},\mathbf{v}),\; \varphi(\mathbf{x}) \rightarrow \omega(\mathbf{x},\mathbf{f}(\mathbf{x}))
\end{array}
\qquad
F_\setminus\;\begin{array}{l}
\alpha\!:\; R_1(\mathbf{x}),\; \varphi(\mathbf{x}) \rightarrow \omega(\mathbf{x},\mathbf{f}(\mathbf{x}))\\
q^-\!:\; S'(\mathbf{u}) \rightarrow \llbracket R_1(\mathbf{u})\rrbracket,\; \neg R_2(\mathbf{u})\\
\hline
\alpha'\!:\; S'(\mathbf{u}),\; \varphi(\mathbf{x}) \rightarrow \omega(\mathbf{x},\mathbf{f}(\mathbf{x}))
\end{array}
$$

$$
F_\cup\;\begin{array}{l}
\alpha_1\!:\; R_1(\mathbf{x}),\; \varphi_1(\mathbf{x}) \rightarrow \omega_1(\mathbf{x},\mathbf{f}(\mathbf{x}))\\
\alpha_2\!:\; R_2(\mathbf{y}),\; \varphi_2(\mathbf{y}) \rightarrow \omega_2(\mathbf{y},\mathbf{g}(\mathbf{y}))\\
q^-\!:\; S'(\mathbf{u}) \rightarrow \llbracket R_1(\mathbf{u})\rrbracket \vee \llbracket R_2(\mathbf{u})\rrbracket\\
\hline
\alpha'\!:\; S'(\mathbf{u}),\; \varphi_1(\mathbf{x}),\; \varphi_2(\mathbf{y}) \rightarrow \omega_1(\mathbf{x},\mathbf{f}(\mathbf{x})) \vee \omega_2(\mathbf{y},\mathbf{g}(\mathbf{y}))
\end{array}
$$

Fig. 5. Forward propagation (f-APC) rules for $\alpha, q^- \rightsquigarrow \alpha'$

For the case of b-APC we perform a *preorder* traversal of the operator tree. At each operator node we propagate (1) each source annotation given as input to the operator propagation step, and (2) all unique annotations that can be derived (including those that contain intermediate relation symbols) by repeatedly applying the corresponding inference rule of the operator. Once all nodes of the operator tree have been visited, the subset of source-to-target annotations (not mentioning intermediate relations) are propagated through the query. We apply a similar procedure for f-APC, but instead use a postorder traversal (*i.e.*, bottom up), as shown in Fig. 6 (b). The following example illustrates the inference steps of b-APC:

Example 4 Let $\mathtt{R}_1(o,x,y,t,v)$, $\mathtt{R}_2(u,p)$ be input schemas, $\mathtt{S}'(o,x,y,v,u,p)$ the output schema (Fig. 6), where $\mathtt{S}'$ is the given output annotation

α': $\mathtt{S}'(o,x,y,v,u,p) \rightarrow \mathsf{Observation}(o),\ \mathsf{hasVal}(o,v),\ \mathsf{Species}(p)$

Also consider the query shown in the figure, which combines all $\mathtt{R}_1$ observations, made at a particular time $\mathtt{c}$, with species observed at a location $\mathtt{d}$ (*e.g.*, assuming the spatial extent of $\mathtt{R}_1$ is $\mathtt{d}$, the query extends $\mathtt{R}_1$ with its corresponding species). We follow the navigation path given in Fig. 6 (a) to compute the backward propagation. The first step derives $\alpha_1 := \alpha'(q_4)$ by applying $B_\times$ to α' and q_4 (the goal atom is in double brackets):

q_4: $\mathtt{R}''_1(o,x,y,v),\ \mathtt{R}'_2(u,p) \rightarrow \llbracket \mathtt{S}'(o,x,y,v,u,p)\rrbracket$

The resulting annotation $\alpha_1 = \alpha'(q_4)$ is propagated downwards the operator tree:

α_1: $\mathtt{R}''_1(o,x,y,v),\ \mathtt{R}'_2(u,p) \rightarrow \mathsf{Observation}(o),\ \mathsf{hasVal}(o,v),\ \mathsf{Species}(p)$

The next step is $\alpha_2 = \alpha_1(q_3)$ via rule B_π, *i.e.*, applying α_1 to q_3 (= $\pi_{oxyv}(\mathtt{R}'_1)$):

q_3: $\mathtt{R}'_1(o,x,y,t,v) \rightarrow \llbracket \mathtt{R}''_1(o,x,y,v)\rrbracket$

which results in

α_2: $\mathtt{R}'_1(o,x,y,t,v),\ \mathtt{R}'_2(u,p) \rightarrow \mathsf{Observation}(o),\ \mathsf{hasVal}(o,v),\ \mathsf{Species}(p)$

Next we apply α_2 to q_2 (= $\sigma_{t=\mathtt{c}}(\mathtt{R}_1)$)

q_2: $\mathtt{R}_1(o,x,y,t,v), t{=}\mathtt{c} \rightarrow \llbracket \mathtt{R}'_1(o,x,y,t,v)\rrbracket$

and using rule B_σ we obtain $\alpha_3 = \alpha_2(q_2)$:

α_3: $\mathtt{R}_1(o,x,y,t,v), t{=}\mathtt{c},\ \mathtt{R}'_2(u,p) \rightarrow \mathsf{Observation}(o),\ \mathsf{hasVal}(o,v),\ \mathsf{Species}(p)$

Finally, on the second, parallel branch we have a selection q_1 (= $\sigma_{u=\mathtt{d}}(\mathtt{R}_2)$):

q_1: $\mathtt{R}_2(u,p), u{=}\mathtt{d} \rightarrow [\![\mathtt{R}'_2(u,p)]\!]$

We obtain the final result $\alpha_4 := \alpha_1(q_1)$ via B_σ:

α_4: $\mathtt{R}_1(o,x,y,t,v), t{=}\mathtt{c},\ \mathtt{R}_2(u,p), u{=}\mathtt{d} \rightarrow \mathsf{Observation}(o),\ \mathsf{hasVal}(o,v),\ \mathsf{Species}(p)$□

The forward algorithm f-APC proceeds similarly but bottom-up instead of top-down:

Example 5 Consider two relations $\mathtt{R}_1$ and $\mathtt{R}_2$ with semantic annotations α_a and α_b:

α_a: $\mathtt{R}_1(o,x,y,t,v) \rightarrow \mathsf{Observation}(o),\ \mathsf{hasVal}(o,v)$

α_b: $\mathtt{R}_2(u,p) \rightarrow \mathsf{Site}(u),\ \mathsf{Species}(p),\ \mathsf{observedIn}(p,u)$

and the same query q as in the previous example. We follow the navigation path given in Fig. 6 (b) to compute the forward propagation. The first step derives $\alpha_1 := \alpha(q_2^-)$ by applying F_σ to α_a and the inverse q_2^- of the operator $\sigma_{t=\mathtt{c}}(\mathtt{R}_1)$

q_2^-: $\mathtt{R}'_1(o,x,y,t,v) \rightarrow [\![\mathtt{R}_1(o,x,y,t,v)]\!],\ t = \mathtt{c}$

α_1: $\mathtt{R}'_1(o,x,y,t,v) \rightarrow \mathsf{Observation}(o),\ \mathsf{hasVal}(o,v)$

The next step derives $\alpha_2 \leftarrow \alpha_1, q_3^-$ by applying F_π to α_1 and operator $\pi_{o,x,y,v}$ (q_3^-)

q_3^-: $\mathtt{R}''_1(o,x,y,v) \rightarrow [\![\mathtt{R}'_1(o,x,y,g(o,x,y,v),v)]\!]$

α_2: $\mathtt{R}''_1(o,x,y,v) \rightarrow \mathsf{Observation}(o),\ \mathsf{hasVal}(o,v)$

The next step derives $\alpha_3 \leftarrow \alpha_b, q_1^-$ by applying F_σ to α_b and the operator $\sigma_{u=\mathtt{d}}$ (q_1^-)

q_1^-: $\mathtt{R}'_2(u,p) \rightarrow [\![\mathtt{R}_2(u,p)]\!],\ u = \mathtt{d}$

α_3: $\mathtt{R}'_2(u,p) \rightarrow \mathsf{Site}(u),\ \mathsf{Species}(p),\ \mathsf{observedIn}(p,u)$

The last step derives α' (denoted α'_a and α'_b below) by applying $F_\times$ twice, once to α_2 and the operator $\times$ (q_4^-) and then to α_3 and q_4^-.

q_4^-: $\mathtt{S}'(o,x,y,v,u,p) \rightarrow [\![\mathtt{R}''_1(o,x,y,v)]\!],\ \mathtt{R}'_2(u,p)$

α'_a: $\mathtt{S}'(o,x,y,v,u,p) \rightarrow \mathsf{Observation}(o),\ \mathsf{hasVal}(o,v)$

q_4^-: $\mathtt{S}'(o,x,y,v,u,p) \rightarrow \mathtt{R}''_1(o,x,y,v),\ [\![\mathtt{R}'_2(u,p)]\!]$

α'_b: $\mathtt{S}'(o,x,y,v,u,p) \rightarrow \mathsf{Site}(u),\ \mathsf{Species}(p),\ \mathsf{observedIn}(p,u)$

Soundness and Termination of APC Rules and Algorithms. Applications of APC inference rules correspond to one or more first-order resolution steps [22,3]. Thus, from the soundness of resolution, the soundness of f-APC and b-APC is immediate.

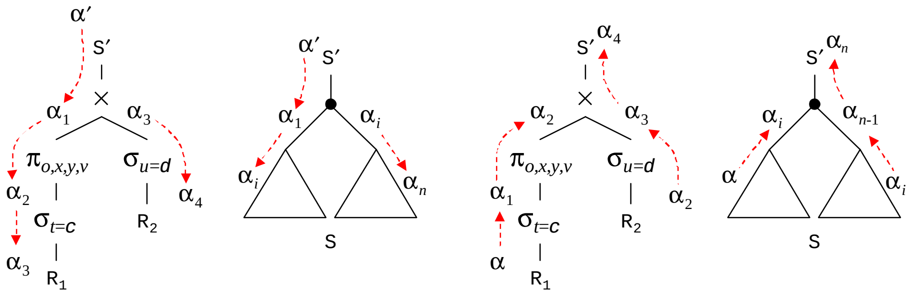
Proposition 1 (Soundness of b-APC and f-APC) *For Σ'_α and Σ_q as in Fig. 4:*

If $\Sigma_{\alpha'} \cup \Sigma_q \vdash_{\text{b-APC}} \Sigma_\alpha$ then $\Sigma_{\alpha'} \cup \Sigma_q \models \Sigma_\alpha$

Similarly, for Σ_α and Σ_{q^-} as in Fig. 5:

If $\Sigma_\alpha \cup \Sigma_{q^-} \vdash_{\text{f-APC}} \Sigma_{\alpha'}$ then $\Sigma_\alpha \cup \Sigma_{q^-} \models \Sigma_{\alpha'}$

Annotation propagation in both the forward and backward versions of APC proceeds by structural induction on the operator tree of q. Since there are only finitely many rule applications per node in the tree, and since each node in the tree is visited once, termination follows. Note however, that we assume here that annotations and query operators are strictly source-to-target (*e.g.*, for recursive (Datalog) queries, termination is not guaranteed).



a) Backward APC operator tree traversal

b) Forward APC operator tree traversal

Fig. 6. Structural inductions on operator trees for queries q, where (a) shows a preorder navigation for b-APC and (b) a postorder navigation for f-APC.

Proposition 2 (Termination of b-APC and f-APC) *For any relational query q and any finite set of annotations, the algorithms b-APC and f-APC terminate.*

4 Discussion

Semantic annotations are a promising approach for ensuring compatibility and reuse of actors in scientific workflows. However, the current manual process of generating semantic annotations limits their utility. We have proposed a method for automatically propagating semantic annotations forward and/or backward through a dataflow process networks of actors, described by relational queries. We have shown how the problem of propagation can be recast as one of constraint implication, and presented a calculus of annotation propagation (APC) and developed two algorithms (b-APC and f-APC), corresponding to a top-down and bottom-up propagation of annotations through a query's operator tree. Both algorithms have been implemented in Prolog. Despite the initial results presented here, several interesting problems remain. The presented approach can be seen as a specialized first-order resolution procedure which is guided by the operator structure of a query. In general, resolution methods, including specialized versions such as the Chase (see *e.g.* [12]), may not terminate, *e.g.*, due to recursive rules and Skolem symbols. In contrast, our approach terminates, because we guide and limit the inference steps using the structure of the operator tree. However, we cannot always guarantee to obtain the most specific annotation via our propagation algorithm. In future work we plan to identify classes of queries and annotations where most specific annotations can be effectively computed. Similarly, we are interested in deriving approximate solutions even in cases where a most specific annotation does not exist. relationship between the formalization of annotation propagation as mapping composition (only sketched in this paper) and the one as constraint implication (used in this paper as the basis for APC).

References

1. P. A. Bernstein. Applying Model Management to Classical Meta-Data Problems. *Conference on Innovative Data Systems Research (CIDR)*, pages 209–220, 2003.

2. D. Bhagwat, L. Chiticariu, W. C. Tan, and G. Vijayvargiya. An Annotation Management System for Relational Databases. *Intl. Conf. on Very Large Data Bases (VLDB)*, 2004.
3. J. Biskup and A. Kluck. A New Approach to Inferences of Semantic Constraints. *Proc. of Advances in Databases and Information Systems*, 1997.
4. S. Bowers, K. Lin, and B. Ludäscher. On Integrating Scientific Resources through Semantic Registration. *Intl. Conf. on Scientific and Statistical Database Management (SSDBM)*, 2004.
5. S. Bowers and B. Ludäscher. Actor-Oriented Design of Scientific Workflows. *Intl. Conference on Conceptual Modeling (ER)*, Klagenfurt, Austria, 2005.
6. S. Bowers and B. Ludäscher. Towards Automatic Generation of Semantic Types in Scientific Workflows. *Intl. Workshop on Scalable Semantic Web Knowledge Base Systems*, 2005.
7. P. Buneman, S. Khanna, and W. C. Tan. Why and Where: A Characterization of Data Provenance. *Intl. Conference on Database Theory (ICDT)*, volume 1973 of *LNCS*, 2001.
8. P. Buneman, S. Khanna, and W. C. Tan. On Propagation of Deletions and Annotations Through Views. *ACM Symposium on Principles of Database Systems (PODS)*, 2002.
9. D. Chalcraft, J. Williams, M. Smith, and M. Willig. Scale Dependence In The Species-Richness-Productivity Relationship: The Role Of Species Turnover. *Ecology*, 85(10), 2004.
10. K. L. Clark. Negation as Failure. *Logic and Databases*. Plemum Press, 1977.
11. Y. Cui and J. Widom. Lineage Tracing for General Data Warehouse Transformations. *Intl. Conference on Very Large Data Bases (VLDB)*, Rome, Italy, 2001.
12. A. Deutsch, L. Popa, V. Tannen. Query Reformulation with Constraints. *SIGMOD Record*, 2006, to appear.
13. R. Fagin, P. G. Kolaitis, L. Popa, and W. C. Tan. Composing Schema Mappings: Second-Order Dependencies to the Rescue. *ACM Symposium on Principles of Database Systems (PODS)*, Paris, France, 2004.
14. F. Geerts, A. Kementsietsidis, and D. Milano. MONDRIAN: Annotating and Querying Databases through Colors and Blocks. *Intl. Conference on Data Engineering (ICDE)*, 2006.
15. G. Kahn and D. B. MacQueen. Coroutines and Networks of Parallel Processes. In B. Gilchrist, editor, *Proc. of the IFIP Congress 77*, pages 993–998, 1977.
16. E. A. Lee and T. Parks. Dataflow Process Networks. *Proceedings of the IEEE*, 83(5):773–799, May 1995.
17. B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao. Scientific Workflow Management and the Kepler System. *Concurrency and Computation: Practice & Experience*, 2006. to appear.
18. B. Ludäscher and C. A. Goble, editors. *Guest Editors' Introduction to the Special Section on Scientific Workflows*, volume 34(3). ACM SIGMOD Record, September 2005.
19. S. Majithia, M. S. Shields, I. J. Taylor, and I. Wang. Triana: A Graphical Web Service Composition and Execution Toolkit. *Proceedings of the IEEE International Conference on Web Services (ICWS'04)*, pages 514–524. IEEE Computer Society, 2004.
20. A. Nash, P. A. Bernstein, and S. Melnik. Composition of Mappings Given by Embedded Dependencies. *ACM Symposium on Principles of Database Systems (PODS)*, 2005.
21. T. M. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, R. M. Greenwood, T. Carver, K. Glover, M. R. Pocock, A. Wipat, and P. Li. Taverna: A Tool for the Composition and Enactment of Bioinformatics Workflows. *Bioinformatics*, 20(17), 2004.
22. J. A. Robinson. A Machine-Oriented Logic Based on the Resolution Principle. *Journal of the ACM*, 12(1):23–41, 1965.
23. Y. Simmhan, B. Plale, D. Gannon. A Survey of Data Provenance in e-Science. In [18].
24. J. Yu, R. Buyya. A Taxonomy of Scientific Workflow Systems for Grid Computing. In [18].