Towards Automatic Generation of Semantic Types in Scientific Workflows*

Shawn Bowers[†] Bertram Ludäscher[†‡]

[†]UC Davis Genome Center
[‡]Department of Computer Science
University of California, Davis
{sbowers,ludaesch}@ucdavis.edu

Abstract. Scientific workflow systems are problem-solving environments that allow scientists to automate and reproduce data management and analysis tasks. Workflow components include *actors* (*e.g.*, queries, transformations, analyses, simulations, visualizations), and *datasets* which are produced and consumed by actors. The increasing number of such components creates the problem of discovering suitable components and of composing them to form the desired scientific workflow. In previous work we proposed the use of *semantic types* (annotations relative to an ontology) to solve these problems. Since creating semantic types can be complex and time-consuming, scalability of the approach becomes an issue. In this paper we propose a framework to automatically derive semantic types from a (possibly small) number of initial types. Our approach propagates the given semantic types through workflow steps whose input and output data structures are related via query expressions. By propagating semantic types, we can significantly reduce the effort required to annotate datasets and components and even derive new "candidate axioms" for inclusion in annotation ontologies.

1 Introduction

Scientific analyses are often performed as a series of computation steps, grouped together to form the logical stages of an analysis. For example, pre-processing input data, applying one or more statistical or data-mining techniques, and post-processing and visualizing analysis results or discovered patterns may each be constructed from a number of smaller computation steps. Scientific-workflow systems (*e.g.*, Taverna, Triana,[1] and Kepler [13]) have emerged as more versatile, extensible environments, compared with shell scripts and spreadsheets, to model and execute such analytical processes. Scientific workflows are useful to design [6] and execute end-to-end processes, and to enable the composition and sharing of computation steps, allowing scientists to more quickly experiment with and run complex analyses. Scientific workflows systems also provide scientists with a single point of access to heterogeneous data and computation services from multiple scientific disciplines and research groups. Providing such access

* This work supported in part by NSF/ITR SEEK (DBI-0533368), NSF/ITR GEON (EAR-0225673), and DOE SDM (DE-FC02-01ER25486)

[1] See taverna.sourceforge.net and www.trianacode.org, respectively.

enables "cross-cutting" science, *e.g.*, allowing ecological and genomic data to be mixed or complex statistical models to be combined across disciplines. A major challenge in providing this capability are semantic and terminological differences across scientific domains. Even within a discipline such as ecology, these problems exist, making data integration and service composition difficult both automatically and for a scientist. Our work focuses on providing rich metadata and ontologies to help bridge this gap and to enable wide-scale data and workflow interoperability. We have developed a framework for registering the semantics of data and services based on *semantic types*, which are mappings from datasets or services to concept expressions of an ontology. Our framework has been used to (semi-)automate data integration [2,14] and service composition [5], and is currently being developed and used within the Kepler scientific workflow system [2]. Providing semantic types, however, can be time-consuming for data and service providers, thereby limiting the applicability of metadata-intensive approaches to scientific data management.

In this paper, we describe an approach to make the management of semantic types more scalable by automating the generation of intermediate types within workflows. Our approach propagates the given semantic types through actors whose input and output data structures are related by a query expression, possibly approximating an actor's function. In Section 2 we introduce the propagation problem and describe the benefits of our approach. In Section 3 we present our semantic-type framework. In Section 4 we develop our approach for propagating semantic types; related and future work is discussed in Section 5.

2 The Propagation Problem

For our purposes, a *scientific workflow* consists of a number of *actors*, which are connected via directed edges called *channels* (Figure 1). An actor is a component (*e.g.*, a web service, shell command, local function, remote job) that can consume and produce data tokens. An actor has zero or more uniquely named *ports* designated as either input or output. With each port we can associate a *structural type* (or *schema*) $\mathtt{S}$ describing the structure of data (tokens) flowing through that port. Kepler's structural type system, inherited from Ptolemy II [7], includes atomic types, *e.g.*, `string` and `double`, and complex types such as `list` and `record`. In a workflow, actors exchange information using channels that link an output port (token producer) to one or more input ports (token consumers). Workflows are executed according to a *model of computation* (implemented by a so-called *director* [7]), which specifies the overall workflow orchestration and scheduling. Here we assume a model of computation that corresponds to a dataflow process network [11]. Figure 1 shows a simple workflow in Kepler for computing species richness and productivity. The workflow performs a number of distinct computations over two input datasets shown on the left of the workflow, which results in the richness and productivity derived data products shown on the right.

Semantic Type Propagation. Figure 2 depicts the problem of propagating an input semantic type α through an actor, yielding the output semantic type α'. A *semantic type* α associates elements of a schema $\mathtt{S}$ with concepts from an ontology O. The goal of

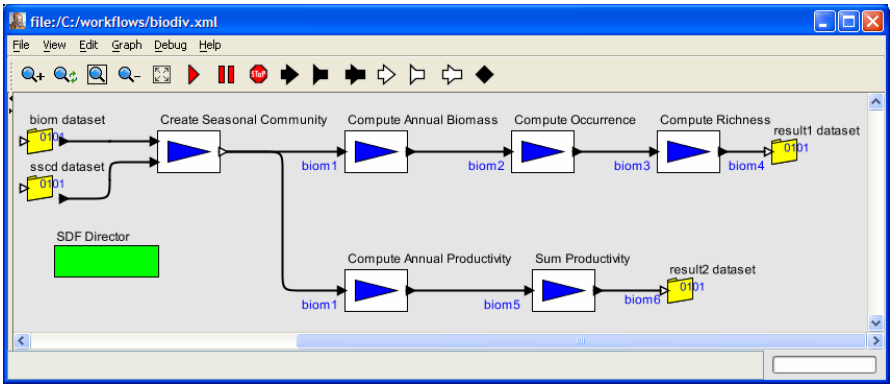


Fig. 1. Simple scientific workflow for computing species richness and productivity [9]

propagation is to automatically generate α', given α. This is only possible if something is known about the relation between elements of S and those in S'. A query expression q provides this relation. The query q can approximate[2] the actual function $f : \mathtt{S} \rightarrow \mathtt{S'}$ computed by the actor, *e.g.*, q might "overestimate" f such that $q(D) \supseteq f(D)$ for any input data D. The propagation problem is to determine $\alpha' : \mathtt{S'} \rightarrow O$, the semantic type of the output, given the input type α and the query q. We denote this problem as computing $\alpha' = \alpha \circ q^{-1}$ in Figure 2, *i.e.*, the composition of α and (the inverse of) q. Based on our semantic-type framework (Section 3), we describe an initial approach towards solving this problem (Section 4). Our approach places few restrictions on where initial semantic types are given. Semantic types may be provided for input data or for inputs of some actor(s) only, significantly reducing the amount of semantic description required to reuse workflows and actors. A user may also provide additional semantic types at specific points within a workflow, *e.g.*, when the result of a computation creates new data values or adds semantic information. These semantic types are also propagated through actors. The advantages of our approach directly benefit scientific workflow engineers at various stages of workflow construction, including:

- Semantic types can be derived at **workflow design-time** (even before all actors or data are available), and thus can be used as a tool to help workflow engineers build new analyses. For example, propagated types can be presented to the user after two actors are connected, showing the resulting semantic types of combining the steps and the impact on the rest of the workflow.
- Scientific workflows can often be executed over different input datasets. The workflow's global inputs are typically quite generic, while a given dataset may have very specific semantic types. Our approach can propagate these specific semantic types of datasets, resulting in more accurate (specialized) semantic types at **data binding-time**.
- When a workflow is executed, derived data products are automatically given the propagated semantic type, minimizing the effort required to semantically type datasets at **workflow runtime**.

[2] Consider a filter function f that removes outliers and returns only "good" tokens. This function can be modeled as a selection σ_θ where θ is the filter condition. Obviously, S = S' in this case, which means that α can be propagated as is (in fact, $\alpha' = \alpha \wedge \theta$ can be derived).

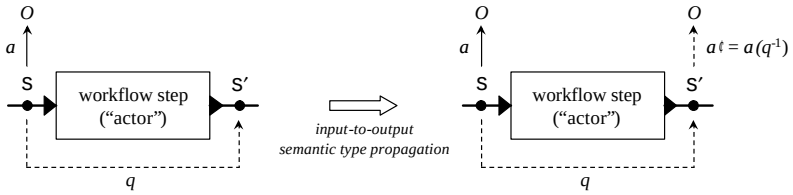
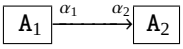


Fig. 2. Actor with semantic type α, propagated via query q, yielding semantic type α'.

Another advantage of propagation is *ontology augmentation*. Consider an actor $\mathtt{A}_2$ having as input species richness data. The developer of $\mathtt{A}_2$ may provide a semantic type α_2 for $\mathtt{A}_2$'s input, stating that it was designed to "consume" RICHNESS data.[3] Assume a workflow designer has connected the output of another actor $\mathtt{A}_1$ to the input of $\mathtt{A}_2$, and for the output of $\mathtt{A}_1$ a semantic type α_1 has been derived via propagation, indicating that $\mathtt{A}_1'$s output is of type $\mathtt{sum}$(OCCURRENCE), *i.e.*, the arithmetic sum (resulting from an aggregation) of OCCURRENCE data. For the link:



to be semantically type correct, we must have that α_1 is "compatible" with α_2, *i.e.*, $\mathtt{sum}$(OCCURRENCE) $\sqsubseteq$ RICHNESS. If we choose to run our propagation system in "automatic mode", it will augment the given ontology O with this additional axiom. Conversely, the system can be run in "interactive mode", asking the user to determine whether the inferred axiom is correct and can be included in O, or whether there is something wrong with the connection. This example also illustrates that semantic type constraints between connected actors are "soft" in the sense that one can still execute the corresponding workflow steps, even though doing so may not be semantically meaningful. In contrast, the structural type constraint when connecting $\mathtt{A}_1$ and $\mathtt{A}_2$ is "hard", *i.e.*, the schema types $\mathtt{S}_1$ and $\mathtt{S}_2$ must satisfy a subtyping constraint $\mathtt{S}_1 \preceq \mathtt{S}_2$ for the connection to be executable.

3 The Semantic-Type Framework

Query Expressions. Actors may have an associated *query expression* q, which may be derived from the component implementation (*e.g.*, from a script or generated data transformation) or explicitly given by a service provider (*e.g.*, for "black-box" actors whose inner workings are unknown). The most general form of a query expression is a logic constraint $\varphi_{\mathtt{S}\cup\mathtt{S}'}$ associating schema elements of the input port(s) $\mathtt{S}$ of an actor with those of its output port(s) $\mathtt{S}'$. In analogy to data integration terminology, we can call q an *Output-as-View* (OAV) mapping if it has the form $q = P_{\mathtt{S}'} \texttt{ :- } \varphi_{\mathtt{S}}$, and an *Input-as-View* (IAV) mapping if it has the form $q = P_{\mathtt{S}} \texttt{ :- } \varphi_{\mathtt{S}'}$. Here, q is a logic atom defining data elements of the output schema $\mathtt{S}'$ (or the input schema $\mathtt{S}$) in terms of the query $\varphi_{\mathtt{S}}$ (or $\varphi_{\mathtt{S}'}$) over $\mathtt{S}$ (or $\mathtt{S}'$, respectively). In this paper we focus on query expressions

[3] We use SMALLCAPS to denote concepts from an ontology O.

given in the OAV form. Query expresssions can contain the standard relational operators select, project, join, union, and group-by with aggregation (*i.e.*, sum, count, avg, min, and max). Note that a query expression q does not need to exactly capture the function f being computed by an actor. It is sufficient if q approximates f such that all structural associations between `S` and `S'` are preserved. These structural associations will then be used to propagate the semantic types from `S` to `S'`.

We use Datalog notation [1], extended with aggregrate functions and grouping, to denote query expressions. Relations are denoted using capitals (`Biom`, `Sscd`, etc.) and variables are in lower-case (x, y, ...). For example, query q_1 approximates the *Create Seasonal Community* component of Figure 1:

$$\texttt{Biom1}(o,y,s,t,p,b) \texttt{ :- } \texttt{Biom}(o,y,s,t,p,b),\ \texttt{Sscd}(p) \qquad (q_1)$$

This query selects `Biom` tuples (returned as `Biom1` tuples) whose p-values are present in the `Sscd` data set. `Biom` represents a relational table consisting of measurements (with measurment-id o) of biomass b for a particular species p, year y, season s, and plot t. The `Sscd` relation contains species found within a particular community.

Query expressions can contain group-by with aggregate operators syntactically written $agg(x|\bar{y})$, where agg is the name of the aggregate operation, x is the aggregation variable, and $\bar{y}$ is a comma-separated list of grouping variables. We introduce a new variable in the head of an aggregate query and assign it (using the "$\leftarrow$" symbol) to the aggregate expression. For example, query q_2 gives the annual biomass for each plot and species (the *Compute Annual Biomass* actor):

$$\texttt{Biom2}(y,t,p,z \leftarrow \texttt{sum}(b|y,t,p)) \texttt{ :- } \texttt{Biom1}(o,y,s,t,p,b) \qquad (q_2)$$

Union operations are expressed in the normal way using multiple rules. For example, query q_3 returns annual occurrence measurements (the *Compute Occurrence* actor):

$$\begin{aligned} &\texttt{Biom3}(y,t,p,1) \texttt{ :- } \texttt{Biom2}(y,t,p,b),\ b > 0 \qquad (q_3)\\ &\texttt{Biom3}(y,t,p,0) \texttt{ :- } \texttt{Biom2}(y,t,p,b),\ b \leq 0 \end{aligned}$$

Ontologies. We use description logic to express ontologies, which are used to formally define the terms (*concepts*) in a given domain and their relationships (*roles*). The OWL-DL standard is also used in Kepler for storing and exchanging ontologies. A simple ontology is shown in Figure 3, representing definitions for ecological measurements (concept Observation) and ecological concepts such as Abundance, Richness, etc. According to the underlying description-logic definitions (not shown in the figure), every observation has exactly one observed property (*e.g.*, abundance) and item (*e.g.*, species), and one or more spatial and temporal contexts. We assume a reasoning system is available to compute subsumption hierarchies from concept and role definitions. We use subsumption in particular to determine whether channels defined between actors are semantically compatible.

Semantic Types. Each structural type of a dataset, input, or output port, may be given a semantic type, which in its most general form is a logic constraint $\alpha_{S \cup O}$ associating

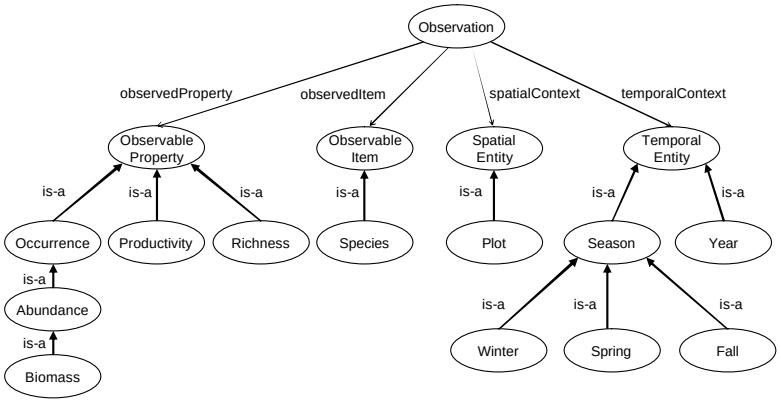


Fig. 3. Simplified ecological measurement ontology shown graphically

$$
\begin{aligned}
&\texttt{Biom}(o, y, s, t, p, b) && \rightarrow \textsc{Observation}(o) && (1)\\
&\texttt{Biom}(o, y, s, t, p, b) && \rightarrow \textsc{temporalContext}(o, y) \wedge \textsc{Year}(y) && (2)\\
&\texttt{Biom}(o, y, s, t, q, b) \wedge s = \texttt{`W'} && \rightarrow \textsc{temporalContext}(o, s) \wedge \textsc{Winter}(s) && (3)\\
&\texttt{Biom}(o, y, s, t, q, b) \wedge s = \texttt{`S'} && \rightarrow \textsc{temporalContext}(o, s) \wedge \textsc{Spring}(s) && (4)\\
&\texttt{Biom}(o, y, s, t, q, b) \wedge s = \texttt{`F'} && \rightarrow \textsc{temporalContext}(o, s) \wedge \textsc{Fall}(s) && (5)\\
&\texttt{Biom}(o, y, s, t, p, b) && \rightarrow \textsc{spatialContext}(o, t) \wedge \textsc{Plot}(t) && (6)\\
&\texttt{Biom}(o, y, s, t, p, b) && \rightarrow \textsc{observedItem}(o, p) \wedge \textsc{Species}(p) && (7)\\
&\texttt{Biom}(o, y, s, t, p, b) && \rightarrow \textsc{observedProperty}(o, b) \wedge \textsc{Biomass}(b) && (8)
\end{aligned}
$$

Fig. 4. An example semantic type linking `biom` elements to ONTOLOGY elements

schema elements S with concept expressions from an ontology $\mathcal{O}$. We consider a syntactic form that we call *Terminology-as-View* (TAV) in which the ontology structure $\mathcal{O}$ is "virtually populated" with elements from the data schema S, thereby establishing the desired semantic type $\alpha = \alpha_S \rightarrow \alpha_\mathcal{O}$. More precisely, we consider semantic types having the form $\alpha = \forall\vec{x}\exists\vec{y}\,\alpha_\mathtt{S}(\vec{x}) \rightarrow \alpha_\mathcal{O}(\vec{x}, \vec{y})$, where $\alpha_\mathtt{S}(\vec{x})$ is a query over a data structure S, linking selected elements (captured via bindings of the variables $\vec{x}$) to concept expressions $\alpha_\mathcal{O}(\vec{x}, \vec{y})$ over the ontology $\mathcal{O}$.

Figure 4 gives example semantic-type constraints linking schema elements of the `Biom` data structure S used above to concepts and roles from the ontology $\mathcal{O}$ given in Figure 3. Line (1) states that every measurement in `Biom` represents an OBSERVATION. Line (2) states (i) that all year values in `biom` are instances of the YEAR concept, and (ii) that the year instance is a temporal context of the corresponding observation instance. Lines (3–5) are similar, but contain an additional condition on the value of the season, and lines (7–8) annotate species and biomass values.

4 Propagating Semantic Types

We divide our approach for propagating semantic types through actors into three classes of increasing expressibility for query expressions: conjunctive queries (*i.e.*, containing only select, project, and join operations); conjunctive queries with aggregation; and conjunctive queries with aggregation and union.

The Conjunctive Case. Let query q and semantic type α be of the form:

- $q = \forall \overline{u} \exists \overline{v}\, P_{\mathtt{S}'}(\overline{u}) \texttt{ :- } \varphi_{\mathtt{S}}(\overline{u}, \overline{v})$
- $\alpha = \forall \overline{x} \exists \overline{y}\, \alpha_{\mathtt{S}}(\overline{x}) \rightarrow \alpha_{O}(\overline{x}, \overline{y})$

Here, $P_{\mathtt{S}'}$ is a logic atom over the output schema `S'` and $\varphi_{\mathtt{S}}$ is a query over the input schema(s) `S`. Similarly, for semantic types α, we relate instances of a schema `S` with those of an ontology O via subformulas $\alpha_{\mathtt{S}}$ and α_O, respectively.

The basic idea of computing $\alpha' = \alpha \circ q^{-1}$ is as follows. We would like to relate instances of the output schema `S'` with instances of O. Assume a substitution that satisfies $\varphi_{\mathtt{S}}$ (in q) implies $\alpha_{\mathtt{S}}$ (in α). For this substitution we can establish the desired relation between `S'` and O, denoted abstractly as $\mathtt{S}' \overset{q^{-1}}{\rightsquigarrow} \mathtt{S} \overset{\alpha}{\rightsquigarrow} O$. More precisely, we consider q as a logical constraint of the form:

$$q(\overline{u}) = P_{\mathtt{S}'} \rightarrow \underbrace{Q_1 \wedge \cdots \wedge Q_n \wedge \psi}_{\varphi_{\mathtt{S}}}$$

and α of the form

$$\alpha(\overline{x}) = \underbrace{A_1 \wedge \cdots \wedge A_k}_{\alpha_{\mathtt{S}}} \rightarrow \alpha_O$$

where $P_{\mathtt{S}'}$ is a logic atom over the output schema `S'`, Q_i and A_j are logic atoms over the input schema(s) `S`, and ψ and α_O are quantifier-free formulas. We assume that all $\exists$-quantified variables ($\overline{v}$ and $\overline{y}$ above) have been eliminated through Skolemization so that q and α can be seen as (implicitly) $\forall$-quantified formulas with variables $\overline{u}$ and $\overline{x}$.[4]

For propagating α "through" q we use the inverse of q, *i.e.*, the left-to-right ('`head` $\rightarrow$ `body`') direction of the query ('`head :- body`'). This direction is the one used in LAV-style query rewritings (*e.g.*, for sound views) and also corresponds to the usually implicit direction in Datalog-style rules (aka Clark's completion [10]). More precisely, q is defined by the equivalence $\forall \overline{u} \exists \overline{v}\, P_{\mathtt{S}'}(\overline{u}) \leftrightarrow \varphi_{\mathtt{S}}(\overline{u}, \overline{v})$ where intuitively, if $\varphi_{\mathtt{S}}(\overline{u}, \overline{v})$ is a result of the query q (in a model M), then $P_{\mathtt{S}'}(\overline{u})$ must also be true (in M) [10,12].

Observe that q can be written as a conjunction $q_1 \wedge \cdots \wedge q_n \wedge q_\psi$ with $q_i = P_{\mathtt{S}'} \rightarrow Q_i$, and $q_\psi = P_{\mathtt{S}'} \rightarrow \psi$. If we assume there is a substitution σ that unifies some atom Q_{i_0} and some A_{j_0}, *i.e.*, $Q^\sigma_{i_0} = A^\sigma_{j_0}$[5], we can infer from $q_{i_0} = P_{\mathtt{S}'} \rightarrow Q_{i_0}$ and α a new semantic type α'_{i_0} of the form:

$$\alpha'_{i_0} = P^\sigma_{\mathtt{S}'} \wedge (\alpha_{\mathtt{S}} \setminus A_{j_0})^\sigma \rightarrow \alpha^\sigma_O$$

[4] We assume that the variables $\overline{u}$ in $q(\overline{u})$ are disjoint from the variables $\overline{x}$ in $\alpha(\overline{x})$.

[5] T^σ denotes the result of applying σ to a term T.

where $(\alpha_S \setminus A_{j_0})$ is the conjunction $A_1 \wedge \cdots \wedge A_k$ with A_{j_0} removed. It is easy to show that the semantic type α'_{i_0} is implied by q_i and α. In this way, by successively "resolving away" atoms A_j from α_S with matching atoms Q_i from φ_S, we can obtain new semantic types α' that relate elements of the output schema `S'` to those in the ontology $\mathcal{O}$.

Example 1 (Propagation for Conjunctive Queries). Consider query q_1 expressed as a first-order formula:

$$\texttt{Biom1}(o, y, s, t, q, p, b) \rightarrow \texttt{Biom}(o, y, s, t, q, p, b) \wedge \texttt{Sscd}(p) \tag{1}$$

This formula can be resolved with semantic-type expression (8) in Figure 4, resulting in the new formula:

$$\texttt{Biom1}(o, y, s, t, q, p, b) \rightarrow \textsc{observedProperty}(o, b) \wedge \textsc{Biomass}(b) \tag{2}$$

Observe that we now have biomass values b for the output schema `Biom1` semantically typed as Biomass instances, linked through the observedProperty role.

Handling Aggregation. The approach for propagating conjunctive queries can also be used for aggregation, due to the particular syntactic representation used to express aggregate operators. As mentioned in Section 2, we perform an additional step for aggregate queries that connects aggregate operators to certain ontology concept defintions, which can be further used to infer new connections between components. The following simple example demonstrates how propagation is used with aggregation.

Example 2 (Propagation for Aggregate Operators). Consider the following Skolemized query q for the *Compute Richness* actor of Figure 1:

$$\texttt{Biom4}(y, t, r \leftarrow \texttt{sum}(c \mid y, t)) \rightarrow \texttt{Biom3}(y, t, f_p(y, t, c), c) \tag{1}$$

and the following (additional) output semantic type of the *Compute Occurrence* actor[6]:

$$\texttt{Biom3}(y, t, p, c) \rightarrow \textsc{Occurrence}(c) \tag{2}$$

We can resolve (1) and (2) above, resulting in the new formula:

$$\texttt{Biom4}(y, t, r \leftarrow \texttt{sum}(c \mid y, t)) \rightarrow \textsc{Occurrence}(c) \tag{3}$$

Observe that in this example we have "preserved" the fact that r is the `sum` of a variable c, and that values for c are Occurrence instances. Thus, we can see that r is exactly the `sum` of Occurrence. For propagated semantic types of this form, we also propagate a semantic-type expression where r is an instance of a new concept formed from the aggregate name and c's assigned concept. Thus, for the previous propagated semantic type we also propagate:

$$\texttt{Biom4}(y, t, r \leftarrow \texttt{sum}(c \mid y, t)) \rightarrow \texttt{sum}(\textsc{Occurrence})(r) \tag{4}$$

With this additional step it becomes possible, *e.g.*, to determine that the *Compute Richness* actor can safely be connected to other actors that input Richness data, leveraging definitions in the ontology such as $\texttt{sum}(\textsc{Occurrence}) \sqsubseteq \textsc{Richness}$. Here, $\texttt{sum}(\textsc{Occurrence})$ represents an ontology concept that is "linked" with a certain functionality in the query expression language.

[6] *e.g.*, given by the actor developer to account for the new data produced by *Compute Occurrence*

Handling Union. Let the union query q be of the form:

$$q = \forall\overline{u}\exists\overline{v}\, P_{\mathrm{S}'}(\overline{u}) \text{ :- } \varphi^1_{\mathrm{S}}(\overline{u},\overline{v})$$
$$\forall\overline{u}\exists\overline{v}\, P_{\mathrm{S}'}(\overline{u}) \text{ :- } \varphi^2_{\mathrm{S}}(\overline{u},\overline{v})$$

which can also be written as the constraint $q(\overline{u}) = P_{\mathrm{S}'} \rightarrow \varphi^1_{\mathrm{S}} \vee \varphi^2_{\mathrm{S}}$ for $\varphi^1_{\mathrm{S}} = Q^1_1 \wedge \cdots \wedge Q^1_n \wedge \psi^1$ and $\varphi^2_{\mathrm{S}} = Q^2_1 \wedge \cdots \wedge Q^2_n \wedge \psi^2$. To resolve q and α, we rewrite q into clausal form, generating the two formulas $q'(\overline{u}) = P_{\mathrm{S}'} \wedge \neg\varphi^2_{\mathrm{S}} \rightarrow \varphi^1_{\mathrm{S}}$ and $q''(\overline{u}) = P_{\mathrm{S}'} \wedge \neg\varphi^1_{\mathrm{S}} \rightarrow \varphi^2_{\mathrm{S}}$.

Observe that q' (and similarly q'') can be rewritten as a conjunction $q'_1 \wedge \cdots \wedge q'_n \wedge q'_\psi$ with $q'_i = P_{\mathrm{S}'} \wedge \neg\varphi^2_{\mathrm{S}} \rightarrow Q'_i$ and $q'_\psi = P_{\mathrm{S}'} \wedge \neg\varphi^2_{\mathrm{S}} \rightarrow \psi^1$. Assume there is a substitution σ that unifies some atom Q_{i_0} and some A_{j_0} for a semantic type $\alpha(\overline{x}) = A_1 \wedge \cdots \wedge A_k \rightarrow \alpha_O$. From q' and α we can infer α' of the form:

$$\alpha' = P^\sigma_{\mathrm{S}'} \wedge \neg\varphi^{2\,\sigma}_{\mathrm{S}} \wedge (\alpha_{\mathrm{S}} \setminus A_{j_0})^\sigma \rightarrow \alpha^\sigma_O$$

where $(\alpha_{\mathrm{S}} \setminus A_{j_0})$ is the conjunction $A_1 \wedge \cdots \wedge A_k$ with A_{j_0} removed, similar to the regular conjunctive case. As before, we successively "resolve away" atoms A_j from α_{S} with matching atoms Q^1_i from φ^1_{S}.

We note that α' may not be in the desired form for semantic types (it may not be in clausal form) because, *e.g.*, $\neg\varphi^2_{\mathrm{S}}$ may result in a disjunctive formula. For such cases, we can apply the following simple conversion. Assuming query expressions $q = P \rightarrow (Q \wedge R) \vee (Q' \wedge R')$ and semantic types $\alpha = Q \rightarrow \alpha_O$, using resolution we infer, *e.g.*, $\alpha' = P \wedge \neg(Q' \wedge R') \rightarrow \alpha_O$, which becomes $\alpha'_1 = P \wedge \neg Q' \rightarrow \alpha_O$ and $\alpha'_2 = P \wedge \neg R' \rightarrow \alpha_O$.

Example 3 (Propagation for Union). Consider query q_3 as the first-order formula:

$$\texttt{Biom3}(y,t,p,c) \rightarrow (\texttt{Biom2}(y,t,p,f_b(y,t,p)) \wedge\ f_b(y,t,p) > 0\ \wedge\ c = 0)\ \vee \quad (1)$$
$$(\texttt{Biom2}(y,t,p,f_b(y,t,p)) \wedge\ f_b(y,t,p) \leq 0\ \wedge\ c = 1)$$

and the Skolemized semantic-type propagated from the *Compute Annual Biomass* actor (note that we only include the result z of the original aggregrate operator):

$$\texttt{Biom2}(y,t,p,z) \rightarrow \textsc{observedItem}(f_o(y,t,p),p) \wedge \textsc{Species}(p) \quad (2)$$

Rewriting (1) into clausal form gives:

$$\texttt{Biom3}(y,t,p,c) \wedge \neg(\texttt{Biom2}(y,t,p,f_b(y,t,p)) \wedge\ f_b(y,t,p) > 0\ \wedge\ c = 0) \rightarrow \quad (3)$$
$$\texttt{Biom2}(y,t,p,f_b(y,t,p)) \wedge\ f_b(y,t,p) \leq 0\ \wedge\ c = 1$$

$$\texttt{Biom3}(y,t,p,c) \wedge \neg(\texttt{Biom2}(y,t,p,f_b(y,t,p)) \wedge\ f_b(y,t,p) \leq 0\ \wedge\ c = 1) \rightarrow \quad (4)$$
$$\texttt{Biom2}(y,t,p,f_b(y,t,p)) \wedge\ f_b(y,t,p) > 0\ \wedge\ c = 0$$

Resolving (2) and (3), *e.g.*, results in the new annotation:

$$\texttt{Biom3}(y,t,p,c) \wedge \neg(\texttt{Biom2}(y,t,p,f_b(y,t,p)) \wedge\ f_b(y,t,p) \leq 0\ \wedge\ c = 1) \rightarrow \quad (4)$$
$$\textsc{observedItem}(f_o(y,t,p),p) \wedge \textsc{Species}(p)$$

We can now rewrite (5) into our desired form for semantic types.

5 Concluding Remarks

The creation of rich semantic-type annotations can be a complex task, making the problem of automatic generation of such types important for scalable "metadata-intensive" and "semantics-intensive" scientific applications. To this end, we have developed and presented a semantic-type propagation approach and sketched how queries involving selection, projection, join, aggregation, and union could be handled. Our approach is based on an inference procedure similar to the chase [1], which itself can be seen as a form of resolution [4]. Propagating semantic types is also related to work on data provenance [3,8] where the focus (in terms of propagation) is on supporting simple text-based annotations of relational table cells (instead of formulas over schemas), and on augmenting SQL to allow users to state specific schemes for propagating these value-based annotations to query results. In contrast, our semantic types are formal logic-based descriptions linking structural types to ontologies. These semantic types can be propagated within the framework of scientific workflows. We are currently investigating the properties of a specialized inference procedure, based on algorithms in [15] for composing mappings given by logic constraints. We plan to implement semantic type propagation within KEPLER as part of future work.

References

1. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison Wesley, 1995.
2. C. Berkley, S. Bowers, M. Jones, B. Ludaescher, M. Schildhauer, and J. Tao. Incorporating semantics in scientific workflow authoring. In *Proc. of SSDBM*, 2005.
3. D. Bhagwat, L. Chiticariu, W. C. Tan, and G. Vijayvargiya. An annotation management system for relational databases. In *Proc. of VLDB*, 2004.
4. J. Biskup and A. Kluck. A new approach to inferences of semantic constraints. In *In Proc. of Advances in Databases and Information Systems*, 1997.
5. S. Bowers and B. Ludäscher. An ontology-driven framework for data transformation in scientific workflows. In *Proc. of DILS*, volume 2994 of *LNCS*, 2004.
6. S. Bowers and B. Ludäscher. Actor-oriented design of scientific workflows. In *24th Intl. Conf. on Conceptual Modeling (ER)*, 2005.
7. C. Brooks, E. A. Lee, X. Liu, S. Neuendorffer, Y. Zhao, and H. Zheng. The Ptolemy II Manual (vol. 1-3). Technical report, UC Berkeley, 2004.
8. P. Buneman, S. Khanna, and W. C. Tan. Why and where: A characterization of data provenance. In *Proc. of ICDT*, volume 1973 of *LNCS*, 2001.
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. In *Logic and Databases*. Plemum Press, 1977.
11. E. A. Lee and T. M. Parks. Dataflow process networks. *Proc. of the IEEE*, 83(5), 1995.
12. M. Lenzerini. Data integration: A theoretical perspective. In *Proc. of PODS*, 2002.
13. 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*, 2005. to appear.
14. B. Ludäscher, A. Gupta, and M. E. Martone. Model-based mediation with domain maps. In *Proc. of ICDE*, 2001.
15. A. Nash, P. A. Bernstein, and S. Melnik. Composition of mappings given by embedded dependencies. In *Proc. of PODS*, 2005.