Annotations are Relative

Peter Buneman
University of Edinburgh
opb@inf.ed.ac.uk

Egor V. Kostylev
University of Edinburgh
ekostyle@inf.ed.ac.uk

Stijn Vansummeren
Université Libre de Bruxelles (ULB)
svsummer@ulb.ac.be

ABSTRACT

Most systems that have been developed for annotation of data assume a two-level structure in which annotation is superimposed on, and separate from, the data. However there are many cases in which an annotation may itself be annotated. For example threads in e-mail and newsgroups allow the imposition of one comment on another; belief annotations can be compounded; and valid time, regarded as an annotation can be freely mixed with belief annotations (at time t_1, B_1 believed that at time t_2, B_2 believed that ...).

In this paper we describe a hierarchical model of annotation in which there is no absolute distinction between annotation and data. First, we introduce a term model for annotations and, in order to express the fact that an annotation may apply to two or more data values with some shared structure, we provide a simple schema for annotation hierarchies. We then look at how queries can be applied to such hierarchies; in particular we ask the usual question of how annotations should propagate through queries. We take the view that the query together with schema describes a level in the hierarchy: everything below this level is treated as data to which the query should be applied; everything above it is annotation which should, according to certain rules, be propagated with the query. We also examine the representation of annotation hierarchies in conventional relational structures and describe a technique for annotating datalog programs.

Categories and Subject Descriptors

H.2.1 [**Database Management**]: Logical Design; H.2.4 [**Database Management**]: Systems—*Relational databases*; F.m [**Theory of Computation**]: Miscellaneous

General Terms

Design, Theory

EDBT/ICDT '13, March 18 - 22 2013, Genoa, Italy

Keywords

Annotation, provenance, terms

1. INTRODUCTION

Annotation of data is regarded as an essential part of the process of maintaining a body of data. Several prototype systems [5, 9, 12, 18] have been developed for annotation of databases and Web data; annotation is an intrinsic process in Wikis and Curated databases; and in RDF, some form of annotation is now regarded as essential for describing validity or provenance [23]. As an acute example of this, a great deal of RDF has been generated by straightforward extraction from existing databases. In this process the time of extraction may not be recorded, with the result that a substantial amount of published RDF data is stale. Without completely restructuring the RDF to record time of extraction, the only viable solution is to annotate the RDF elements with valid time.

In this paper we ask the question: what distinguishes annotation from data? Typically, formal descriptions of annotation, such as the ones provided by the systems cited above, describe a two level structure in which annotation somehow sits on top of data. There are, however, certain kinds of annotation that challenge this assumption. If we take "belief" to be a form of annotation, that is "x believes y" is an annotation on y, then "x believes that y believes z" is surely an annotation on an annotation. Such chains of belief have already been studied, for example, in [10]. Another form of annotation is to be found in email threads. One can either respond to a previously unanswered email, thereby increasing the depth of a thread, or add to the answers to an existing e-mail. In both cases we see a hierarchical structure, leading us to the conclusion that annotation is *relative*: what serves as annotation in one context is data in another.

A hierarchical model of data and annotation. To illustrate the idea that data and annotation are not seperable, suppose that we have some identifier (e.g. a URI) p and that we use $\mathsf{Person}(p)$ to indicate that p identifies a person. We might now write $\mathsf{Believes}(x, \mathsf{Person}(p))$ to indicate the annotation that x believes $\mathsf{Person}(p)$. Suppose also that we want to describe a Height attribute. We could write $\mathsf{Height}(p, 32)$, but this indicates that the height of the URI p is 32, and it might be more accurate to write $\mathsf{Height}(\mathsf{Person}(p), 32)$. What is now the distinction between the $\mathsf{Believes}$ annotation and the Height attribute? And why not assume that $\mathsf{Person}(p)$ is already an annotation on p? By carrying this to the extreme, everything beyond the atomic data values (e.g., p and 32) can be viewed as an annotation.

R_W :

Id	Name	Weight	VT	Comm
123	"Joe"	70	$\{T_1, T_2\}$	$\{C_1\}$
123	"Joe"	90	$\{T_3\}$	$\{C_2\}$

R_H :

Id	Name	Height	VT	Comm
123	"Joe"	180	$\{T_2\}$	$\{C_1\}$
321	"Ann"	160	$\{T_3\}$	$\{C_3\}$

R_1 :

Id	Name	Weight	Height	VT	Comm
123	"Joe"	70	180	$\{T_2\}$	$\{C_1\}$
123	"Joe"	90	180	$\{\}$	$\{C_1, C_2\}$

R_2 :

Id	Name	VT	Comm
123	"Joe"	$\{T_2\}$	$\{C_1, C_2\}$

Figure 1: Tables and queries with valid time and comment annotations

Some may find this extreme view of data as annotation to be unattractive, and it is actually not required for the development of our model. We give it here to illustrate that the boundaries between data and annotation are not fixed. It is interesting to note that the idea that data on the Web is annotation, is not new. It was already described in the early development of hypertext and the Web [3].

Let us carry the idea further and assume that we want to use annotations to describe membership in a subclass hierarchy. For example, we could write Student(Person(p)) and Employee(Person(p)) to describe that person p is also both a student and an employee. The fact that p is a teaching assistant (TA) is now described by the annotation TA(Student(Person(p)), Employee(Person(p))). But is TA annotating two things or one? It is annotating two other annotations but there is only one underlying Person. In order to resolve this we need to introduce a formalism of annotation schema that allows us to express sharing of subterms.

Our first step is therefore to develop a hierarchical model of annotation in which annotations can be considered as data, and conversely data can be considered as annotations. Essentially, databases in our model will simply be sets of terms drawn from a term algebra. The associated schema formalism allows recursion as well as sharing of subterms.

Annotation propagation. Following both the theory and the practice of annotation, one of the most important questions to answer in annotation models is how annotations propagate through queries. Since, in our hierarchical annotation model, we no longer have an absolute notion of annotation, we take the view that a query implicitly identifies some level in an annotation hierarchy. The parts of the hierarchy at that level and below are treated as data while everything above that level is treated as annotation to be propagated. We develop this view formally both for unions of conjunctive queries and for datalog.

Before developing a propagation model for a hierarchical model, let us briefly review how we expect annotations to propagate in a relational model. Following existing proposals [4,7], consider the situation where annotations are *sets* of atomic annotation values that are attached to tuples. There exists two semantics of propagation for such annotations: the *lineage* propagation semantics and the *Boolean* propagation semantics. The first applies to annotations such as comments, the second to annotations such as belief or valid time. To illustrate these two semantics, consider the relations R_W and R_H in Fig. 1. In both relations we treat the first three columns as data while the last two columns contain annotations represented as sets of annotation values. Here, VT stands for valid time and Comm for comments, and these annotations are completely independent of each other. Consider the query Φ_1 that computes the natural join of R_W and R_H. The resulting relation R_1 is given in Fig. 1. Observe that to propagate valid time, it is natural to take the intersection of the sets of valid times of the contributing tuples (which is how the Boolean semantics propagates), but for comments it is more natural to take the union (which is how the lineage semantics propagates).

In our hierarchical model, annotations can be placed on top of one another. In particular, suppose, for example, that we want to allow comments to be placed on top of the valid time annotations, how would the annotations then propagate through queries? And suppose further that comment annotations were themselves annotated with valid time (which is not unreasonable), how would we represent and propagate such a hierarchy? An attempt to deal with such situations was made in [20], where *combined* annotations were considered. However, this model suffers from the same problems as other existing models of annotations—the distinction between data and annotations is given beforehand and all the pieces of data should have an annotation from the same fixed combination of domains. In particularly, it implies that only hierarchies of fixed height can be represented in this model. Our development overcomes these limitations and generalizes both the lineage and Boolean annotation propagation semantics of the flat relational model.

We should mention here that, for the relational case, the lineage and Boolean semantics are special cases of the *semiring* model of Green et al. [16]. They are in fact the semantics that one obtains when applying the semiring model to two natural semirings that interpret their domain as sets of atomic annotation values. The question whether there is a generalization of the general semiring model that works for hierarchies is intriguing. As we will show, however, the interaction between lineage and Boolean propagation in hierarchical structures is already rather subtle. We therefore focus on this interaction, and leave an investigation of the general semiring model on hierarchies for future work.

Relational representation. While the hierarchical annotation model allows us to naturally treat annotations as data and vice versa, it has the disadvantage that many practical applications use ordinary relational databases for storing annotations. We therefore provide a possible representation of hierarchical annotations within the relational data model, in which the annotation relationship is represented as an inclusion dependency between two tables.

Contributions and paper organization. To summarize, our contributions in this paper are the following.

(1) We develop a hierarchical model of annotation in which both data and annotations are described as sets of terms that conform to a schema of constraints for meaningful annotations and sharing sub-terms (Sec. 2).

(2) We then develop the idea of querying hierarchical annotations (Sec. 3). The crucial idea is that the query, together with the data, describes what is to be treated as data and what as annotation. Roughly speaking, anything in the hierarchy that is mentioned by the query is treated as data and anything above it is treated as annotation. The rules for propagating annotations through the query, i.e. annotating the result of the query, are based on lineage and

Boolean semantics. We do this both for unions of conjunctive queries and for datalog, and show that the complexity of query answering does not differ from the complexity for the usual relational databases.

(3) Finally, we provide a possible representation of annotations within the relational data model in which the annotation relationship is represented as an inclusion dependency between two tables (Sec. 4).

We discuss related work in Sec. 5 and conclude in Sec. 6.

2. HIERARCHICAL ANNOTATION MODEL

Terms. We assume an infinite set $\mathcal{V}$ of *domain variables* (which will range over data values); an infinite set $\mathcal{X}$ of *term variables* (which will range over terms); and a set $\mathcal{F}$ of *symbols*, all pairwise disjoint and disjoint from an infinite domain of *data values* $\mathbf{D}$. Each symbol $F \in \mathcal{F}$ has an associated non-negative number $Arity(F)$ called the *arity* of F. We refer to elements of $\mathcal{V} \cup \mathcal{X}$ simply as *variables*.

A *term* is an expression t generated by the grammar

$$t ::= d \mid v \mid x \mid F(t_1, \ldots, t_n)$$

where d ranges over data values in $\mathbf{D}$, v ranges over domain variables, x ranges over term variables, F ranges over symbols in $\mathcal{F}$, and $n = Arity(F)$. We write $\mathcal{T}$ for the set of all terms and $Vars(t)$ for the set of all variables occurring in a term t. A *data term* is a term that does not contain any variables.

The *height* of a term t (denoted $Height(t)$) is the maximum depth of nesting of symbols from $\mathcal{F}$ in t. For example, $Height(F(H(x), d)) = 2$ while $Height(d) = 0$, where $d \in \mathbf{D}$.

A *substitution* over a set of variables $X \subseteq \mathcal{V} \cup \mathcal{X}$ is a mapping $\pi\colon X \to \mathcal{T}$ that assigns a data value $\pi(v) \in \mathbf{D}$ to each domain variable $v \in X \cap \mathcal{V}$, and a term $\pi(x) \in \mathcal{T}$ to each term variable $x \in X \cap \mathcal{X}$. If t is a term, we write $\pi(t)$ for the term obtained from t by simultaneous replacing each variable $x \in X$ that occurs in t by $\pi(x)$. For example, if $\pi = \{v_1 \mapsto 5, x_1 \mapsto F(H(x_2)), x_2 \mapsto 10\}$, then $\pi(G(v_1, x_1, x_3)) = G(5, F(H(x_2)), x_3)$. We extend substitutions pointwise to sets of terms, and hence given such a set T, write $\pi(T)$ for the set $\{\pi(t) \mid t \in T\}$.

Instances and schemas. As usual, a term t' is said to be a subterm of a term t if t' occurs somewhere in t. For example, a, b, $G(b)$, and $F(G(b), a)$ are all subterms of $F(G(b), a)$.

Note the intuition that when we annotate something the annotation is "about" the entire term, including its subterms. As such, we require instances to be closed under taking subterms.

Definition 2.1. An *(annotated) instance* is a finite set of data terms that is closed under the subterm relation.

As we will show in the end of this section, an (unannotated) relational instance can be seen as an instance with terms of height 1.

In order to rule out meaningless annotations and constrain the manner in which annotations can be stacked upon one another in an instance, we next define a simple notion of annotation schema.

Definition 2.2. A *constraint* is a term in which there are no multiple occurrences of the same term variable. An *(annotation) schema* $\mathcal{A}$ is a finite set of constraints.

For example, $F(x, v, v)$ and $F(5, x, v)$ with $x \in \mathcal{X}$ and $v \in \mathcal{V}$ are constraints, but $F(x, x, v)$ is not.

A data term t *conforms* to a schema $\mathcal{A}$ if for every subterm t' of t (including t itself), there is a substitution π such that $t' \in \pi(\mathcal{A})$. An instance Δ conforms to $\mathcal{A}$ if every term in Δ conforms to $\mathcal{A}$. We also say that Δ is an instance *over* $\mathcal{A}$ in that case. In what follows we write $\mathcal{T}_{\mathbf{D}}(\mathcal{A})$ for the set of all data terms conforming to the schema $\mathcal{A}$.

Again, we will see that an (unannotated) relational schema can be easily representable as an annotation schema with constraints of height 0 and 1.

The following examples illustrate the modelling power of schemas and instances.

Example 2.3. Consider the simple annotation schema

$$\mathcal{A} = \{ \quad v, \quad \mathsf{Person}(v_1, v_2), \\ \mathsf{Weight}(\mathsf{Person}(v_1, v_2), v_3), \\ \mathsf{Height}(\mathsf{Person}(v_1, v_2), v_3) \quad \},$$

where all the variables are domain variables from $\mathcal{V}$. The information in the attributes Id, Name, Weight, and Height in the relations R_W and R_H from Fig. 1 can be represented by the annotated instance Δ over $\mathcal{A}$ consisting of the data terms

$joe = \mathsf{Person}(123, \text{"Joe"}), \quad \mathsf{Weight}(joe, 70), \ \mathsf{Height}(joe, 180),$
$ann = \mathsf{Person}(321, \text{"Ann"}), \ \mathsf{Weight}(joe, 90), \ \mathsf{Height}(ann, 160),$

together with all the domain values occurring in these terms (123, "Joe", 180, etc.) Note that, for readability, we have given names to some terms, e.g. $\mathsf{Height}(joe, 180)$ should be read as $\mathsf{Height}(\mathsf{Person}(123, \text{"Joe"}), 180)$.

We now want to annotate some pairs of Height and Weight annotations with a body-mass index (or BMI, for short). For example, we want to add to Δ the term

$$\mathsf{BMI}(\mathsf{Weight}(\mathsf{joe}, 180), \mathsf{Height}(\mathsf{joe}, 90), \text{"High"}).$$

Since we only want to annotate such pairs when they apply to the *same* person, we augment the schema $\mathcal{A}$ with the following constraint:

$$\mathsf{BMI}(\mathsf{Weight}(\mathsf{Person}(v_1,v_2),v_3), \mathsf{Height}(\mathsf{Person}(v_1,v_2),v_4),v_5).$$

Note in particular that Weight and Height are required to have the same person because of the repeated domain variables v_1 and v_2. ■

We say that a schema is *non-recursive* if it does not contain any term variables; otherwise, it is *recursive*. The schema from Ex. 2.3 is non-recursive. Any instance of a non-recursive schema $\mathcal{A}$ contains only data terms of height at most $\max\{Height(c) \mid c \in \mathcal{A}\}$. Annotations cannot be "stacked" to arbitrary depths in non-recursive schemas.

Example 2.4. To give an example of a recursive schema in which annotations *can* be arbitrarily deeply stacked, we add the following constraints to the schema $\mathcal{A}$ of Ex. 2.3:

$$\mathsf{Comm}(x_1, v), \qquad \mathsf{VT}(\mathsf{Comm}(x_1, x_2), v), \\ \mathsf{VT}(\mathsf{Weight}(x_1, x_2), v), \quad \mathsf{VT}(\mathsf{Height}(x_1, x_2), v).$$

Here, x_1, x_2 are term variables and v is a domain variable. Intuitively, $\mathsf{Comm}(t, C)$ indicates that there is a comment C on term t, where we assume for simplicity that comment values belong to the general domain $\mathbf{D}$. Note that Comm annotations are completely generic, they are applicable to any

term, including those that contain comments themselves. In turn, $\mathsf{VT}(t, T)$ indicates that the fact t was valid at time T. By the schema, this annotation is applicable to height, weight and comments, but not to valid time itself or persons. By adding the following data terms to the instance of Ex. 2.3 we complete a representation of the information from the relations R_W and R_H from Fig. 1:

$$\begin{array}{ll}
\mathsf{VT}(\mathsf{Weight}(joe, 70), T_1), & \mathsf{VT}(\mathsf{Weight}(joe, 70), T_2), \\
\mathsf{VT}(\mathsf{Weight}(joe, 90), T_3), & \\
\mathsf{VT}(\mathsf{Height}(joe, 180), T_2), & \mathsf{VT}(\mathsf{Height}(ann, 160), T_3), \\
\mathsf{Comm}(\mathsf{Weight}(joe, 70), C_1), & \mathsf{Comm}(\mathsf{Weight}(joe, 90), C_2), \\
\mathsf{Comm}(\mathsf{Height}(joe, 180), C_1), & \mathsf{Comm}(\mathsf{Height}(ann, 160), C_3),
\end{array}$$

(and the data values T_1, C_1, etc.) Also, we may put some valid times "on top" of comments, in the spirit of dependent annotations of [20]:

$$\begin{array}{l}
\mathsf{VT}(\mathsf{Comm}(\mathsf{Weight}(joe, 70), C_1), T_3), \\
\mathsf{VT}(\mathsf{Comm}(\mathsf{Height}(joe, 180), C_1), T_4).
\end{array}$$ ■

Reasoning with schemas. In should be noted that, from a practical point of view, it can be rather costly to store annotated instances as subterm-closed sets of terms. Indeed, one can always reconstruct an instance Δ from the set of its *maximal terms* (i.e. terms that are only subterms of themselves in Δ) by taking the subterm closure. We call the straightforward representation of an annotated instance Δ *complete* and a representation as a non-closed set of terms, whose closure yield Δ, *incomplete*. Thus the first important problem is to check whether a term is in the subterm closure of a set of terms, which means checking whether a term is a subterm of another one. This can easily be done in P (polynomial time).

Two other reasoning problems concerning schemas, instances, and representations of instances also naturally arise. The first is to check that an instance conforms to an annotation schema. The second is to check that a schema $\mathcal{A}$ is *consistent*, i.e. that there exists an instance Δ conforming to $\mathcal{A}$, such that for every constraint α from $\mathcal{A}$ there exists a data term $t \in \Delta$ and a substitution π such that $\pi(\alpha) = t$. The following proposition observes that both problems have efficient decision procedures.

PROPOSITION 2.5. *The following two problems can be decided in* P*:*

1. *checking that a complete or incomplete representation of an instance conforms to an annotation schema $\mathcal{A}$;*
2. *checking that an annotation schema $\mathcal{A}$ is consistent.*

Note on the expressiveness of schemas. A limitation of our notion of schema is that we make a rather awkward distinction between domain and term variables: sharing can only be specified by repeating domain variables from $\mathcal{V}$, but not term variables from $\mathcal{X}$. This restriction may seem ad hoc. It is well-known, however, that more expressive formalisms for describing term instances that do support sharing of term variables, makes certain decision problems for schemas, such as checking consistency, undecidable [6, Chapter 4]. In the present paper therefore, we have opted for the above restricted schema formalism.

Annotations are relative. As already mentioned in the introduction, we can look at a term like $F(G(d), a)$ from two viewpoints:

- as a normal, standalone piece of data;
- as annotated data, which can be read either as "F annotates $G(d)$ with a" or as "F annotates a with $G(d)$".

Both viewpoints are reasonable in different circumstances: in Sec. 3 we will treat the term as ordinary data when we want to query nested terms; and we will treat it as annotation when want to propagate annotations through queries.

To formalize this approach, fix $\square \in \mathcal{X}$ to be a distinguished term variable. A *(data) context* is a term from $\mathcal{T}$ in which $\square$ occurs exactly once, and no other variable occurs. For example $F(G(\square), d)$ is a context, but $F(G(\square), \square)$ and $F(G(\square), x)$ with $x \in \mathcal{V} \cup \mathcal{X}$ are not.

DEFINITION 2.6. Given an instance Δ and a term t, we say that t is *annotated* in Δ by the set of data contexts

$$A_\Delta(t) = \{c \mid \exists t' \in \Delta : \ \{\square \mapsto t\}(c) = t'\}.$$

Note, that $A_\Delta(t) = \emptyset$ iff $t \notin \Delta$. Moreover, if $t \in \Delta$, then $A_\Delta(t)$ always contains the trivial context $\square$. Also, since Δ is closed under taking subterms, $A_\Delta(t)$ is closed under taking subcontexts (i.e. closed under taking subterms that are themselves data contexts). In what follows, we write $\mathcal{H}$ for the set of all finite subcontext-closed sets of data contexts.

EXAMPLE 2.7. In the instance Δ from Ex. 2.4 the annotation $A_\Delta(\mathsf{Weight}(joe, 70))$ consists of the contexts $\mathsf{VT}(\square, T_1)$, $\mathsf{VT}(\square, T_2)$, $\mathsf{Comm}(\square, C_1)$, $\mathsf{VT}(\mathsf{Comm}(\square, C_1), T_3)$, and $\square$. ■

Correspondence to the relational model. With the definition of schemas and instances given above, schemas that contain only constraints of height 1 without data values, term variables, and repeating data variables (and a technical constraint v), are simply relational database schemas. The instances conforming to such schemas are isomorphic to normal relational instances (except that they also contain the data values mentioned in the tuples). For example, the annotation schema

$$\{ \quad R_W(v_1, v_2, v_3), \quad R_H(v_4, v_5, v_6), \quad v \quad \}$$

corresponds to the relational schema from Fig. 1 (disregarding all the annotations), and the conforming instance contains, e.g. the term $R_W(123, \text{"}\mathsf{Joe}\text{"}, 70)$.

In existing annotation models for relational databases, annotations are often assumed to be simply sets of atomic annotation values that are attached to tuples [4, 7]. We call such sets of atomic annotation values *simple annotations*, to contrast them with richer models in which annotations are modeled as *bags* rather than sets, or in which annotations are assumed to have specific operations on their domain (cf. the semiring approach to provenance) other than the set-theoretic ones. The annotations in tables R_W and R_H in Fig. 1 are examples of simple annotations with sets of valid times and sets of comments.

To represent databases with simple annotations we just need to augment the schema with an auxiliary unary constraint $\mathsf{Atomic}(v)$ to store atomic annotation values in the instance and a generic constraint, e.g. $\mathsf{Annot}(x, \mathsf{Atomic}(v))$ to store correspondence between tuples and atomic annotation values. For simplicity, in the rest of the paper we consider the case where the set of all atomic annotation values coincides with $\mathbf{D}$. In this case we can omit the unary

constraint and simplify $\mathsf{Annot}(x, \mathsf{Atomic}(v))$ to $\mathsf{Annot}(x, v)$. We call this representation of annotated relational databases the *term representation*.

The hierarchical annotation model is a thus a generalization of the standard relational model as well as a model of relational databases with simple annotations.

3. QUERYING AND PROPAGATING HIERARCHICAL ANNOTATIONS

In this section we study the propagation of hierarchical annotations through unions of conjunctive queries and datalog programs. We begin by defining (unions of) conjunctive queries and their normal semantics in Sec. 3.1. We describe existing semantics of propagation of simple annotations in the relational case in Sec. 3.2 and generalize it to hierarchical annotations in Sec. 3.3. We consider datalog in Sec. 3.4.

3.1 Term Conjunctive Queries

Conjunctive queries on hierarchical instances are defined as for conjunctive queries in the relational model, except that now they operate with terms in the body.

DEFINITION 3.1. A *term conjunctive query* (or *TCQ*, for short) is a rule ψ of the form

$$F(\mathbf{x}) \leftarrow \tau_1 \wedge \ldots \wedge \tau_k,$$

where $\tau_1, \ldots, \tau_k$ are terms; $\mathbf{x} = x_1, \ldots, x_n$ is a tuple of distinct variables such that $\{x_1, \ldots, x_n\} \subseteq \mathit{Vars}(\tau_1) \cup \ldots \cup \mathit{Vars}(\tau_k)$, and F is a symbol from $\mathcal{F}$ of arity n.

We call the term $F(\mathbf{x})$ on the left-hand side of ψ the *head* of ψ. We call the set of all terms $\{\tau_1, \ldots, \tau_k\}$ on the right-hand side of ψ the *body* of ψ, and denote this by $\mathit{Body}(\psi)$. Finally, we write $\mathit{Vars}(\psi)$ for the set of all variables that occur in ψ.

DEFINITION 3.2. Let Δ be an instance over an annotation shema $\mathcal{A}$. An *embedding* of a TCQ ψ of the form $F(\mathbf{x}) \leftarrow \tau_1 \wedge \ldots \wedge \tau_k$ into Δ is a substitution π over $\mathit{Vars}(\psi)$ such that that $\pi(\mathit{Body}(\psi)) \subseteq \Delta$. The *result* of evaluating ψ over Δ is the set of terms

$$\psi(\Delta) := \{\pi(F(\mathbf{x})) \mid \pi \text{ is an embedding of } \psi \text{ into } \Delta\}.$$

As the following example shows, the result of a TCQ need not to be closed under subterms, and is therefore not always an instance.

EXAMPLE 3.3. Let ψ be the TCQ

$$\mathsf{WeightAndHeight}(x) \leftarrow \mathsf{Weight}(x, y_1) \wedge \mathsf{Height}(x, y_2)$$

that computes the set of persons that have both a Weight and a Height. The result of ψ over the instance Δ from Ex. 2.3 contains a single term

$$\mathsf{WeightAndHeight}(\mathit{joe}). \qquad \blacksquare$$

In what follows it will be convenient to have the answer to a TCQ to also be an instance. We therefore define Δ_ψ to be the union of Δ and $\psi(\Delta)$. This set is always an instance.

DEFINITION 3.4. A *union of TCQs* (or *TUCQ*, for short) is a finite set Ψ of TCQs that all have the same head. We write $\mathit{Vars}(\Psi)$ for the set of all variables occurring in TCQs in Ψ. The *result* $\Psi(\Delta)$ of a TUCQ $\Psi = \{\psi_1, \ldots, \psi_m\}$ on an instance Δ over an annotation schema $\mathcal{A}$ is defined as

$$\Psi(\Delta) := \psi_1(\Delta) \cup \ldots \cup \psi_m(\Delta).$$

As for TCQs, we write Δ_Ψ for $\Delta \cup \Psi(\Delta)$.

Every TCQ can be seen as a TUCQ of just one disjunct. Hence in what follows we usually state properties only for TUCQs and, if it is not explicitly stated otherwise, they hold for TCQs as well.

Note, that if an annotation schema represents a relational database schema, then TCQs and TUCQs with only terms of height 1 in the body are just standard relational conjunctive queries and unions of conjunctive queries (or, shortly, RCQs and RUCQs, correspondingly).

Though TUCQs generalize RUCQs, next we will see that the complexity of query evaluation does not increase when moving from querying relational databases to querying in the hierarchical annotation model. It should be seen as a sanity check for querying hierarchical annotations.

An algorithm for query evaluation depends on the representation of the input. It is straightforward to show that both combined and data complexity of checking whether a data term t is in the answer to a TUCQ Ψ over an annotated instance Δ is the same as for usual relational databases: NP-complete and in LOGSPACE (in AC^0), respectively, if we are given a complete representation of Δ. It is just a little more elaborate to show, using, e.g. the results of [19] on composition-free queries on trees, that these complexities do not change if the representation is incomplete.

PROPOSITION 3.5. *Given a TUCQ Ψ, a complete or incomplete representation of an annotated instance Δ, and a term t, checking whether $t \in \Delta_\Psi$ is* NP*-complete. It is in* LOGSPACE *if Ψ is fixed.*

3.2 Lineage and Boolean Semantics of Simple Annotation Propagation

Following both the theory and the practice of annotation, one of the most important questions to answer in annotation models is how annotations propagate through queries. A general theory of such propagation is given by the *semiring model* of Green et al. [16]. In this model annotations should form a *(commutative) semiring* which is a structure $\langle \mathcal{K}, +, \times, 0 \rangle$ with a domain of annotations $\mathcal{K}$, binary commutative and associative operations $+$ and $\times$, such that $\times$ distributes over $+$, and a neutral element $0 \in \mathcal{K}$ for $+$, i.e. such an element that $k + 0 = k$ holds for every $k \in \mathcal{K}$.[1] Every database tuple t is associated to an element $\alpha(t) \in \mathcal{K}$ (its annotation). The neutral annotation 0 corresponds to the fact that the tuple "is not" in the database, i.e. in this model all possible tuples are annotated. Queries cannot inspect the annotations, and as such there is a the distinction between data and annotations. During querying, annotations propagate automatically according to the semiring operations: $+$ corresponds to union and projection on tuples and $\times$ corresponds to join. Using our notation, the semiring semantics is formally defined as follows. Given an RCQ ϕ of the form $R(\mathbf{x}) \leftarrow R_1(\mathbf{y}_1) \wedge \ldots \wedge R_k(\mathbf{y}_k)$ and a term instance Δ that represents a relational database, each output tuple $R(\pi(\mathbf{x}))$ with π an embedding of ϕ to Δ is associated to the

[1] Usual definitions of an annotation commutative semiring also include a neutral element for $\times$, though this is not necessary for the exposition that follows.

annotation $a = \sum_{\pi'} \prod_{1 \le i \le k} \alpha(R(\pi'(\mathbf{y}_i)))$, where π' ranges over all embeddings of ϕ to Δ such that $\pi'(\mathbf{x}) = \pi(\mathbf{x})$ and $\alpha(R(\pi'(\mathbf{y}_i)))$ is the annotation of the tuple $R(\pi'(\mathbf{y}_i))$ in Δ.

Our aim is to generalize such propagation of annotations to the hierarchical model in a way that the boundary between data and annotations is not fixed, but is defined dynamically by the query. Note, however, that in our model atomic annotation values come from the general domain $\mathbf{D}$ and we do not assume any specific (semiring) operations on $\mathbf{D}$. Moreover, recall from Def. 2.6 that we consider a hierarchical term t to be annotated by the *set* of data contexts $A_\Delta(t)$ each of which occurs "above" t in the instance Δ.

Hence, such a generalization makes sense only for the cases where the semiring model is applied to semirings whose domain values are also sets (of atomic annotation values) such that $\alpha(t)$ is a set, for every database tuple t. In other words, such a generalization makes sense only for the setting of relational databases with simple annotations introduced in Sec. 2. Essentially, there are two such cases, and we devote this subsection to their brief description. We describe the generalization of these semantics to the hierarchical model in Sec. 3.3.

Let $\mathcal{A}$ be an annotation schema that represents the relational schema with simple annotations as described in the end of Sec. 2. It contains a set of constraints of height 1 and one generic constraint $\mathsf{Annot}(x, v)$ for annotations. Let ϕ be a RCQ of the form $R(\mathbf{x}) \leftarrow R_1(\mathbf{y}_1) \wedge \ldots \wedge R_k(\mathbf{y}_k)$ and let Δ be a term instance that represents a relational database with simple annotations.

The first semantics of propagation of simple annotations is called *lineage* [7]. A good example of annotations with this propagation semantics are comment annotations. Doubt annotations (the converse of belief annotations) behave in the same way. In this case the *annotated result* of ϕ on Δ contains not only the term $R(\pi(\mathbf{x}))$ for every embedding π of ϕ to Δ, but also the set of terms

$$\{\mathsf{Annot}(R(\pi(\mathbf{x})), a) \mid \exists i, 1 \le i \le k,\ \mathsf{Annot}(R_i(\pi(\mathbf{y}_i)), a) \in \Delta\}.$$

Hence, a tuple in the annotated result has an atomic annotation iff *at least one* of the source tuples has this annotation for some embedding which gives this tuple on free variables. The annotated result of a RUCQ over lineage semantics is a union of the results for composing RCQs.

In the semiring model, the lineage semantics corresponds to the setting where each database tuple t is associated with the annotation set $\alpha(t) = \{a \mid \mathsf{Annot}(R(\pi(\mathbf{x})), a) \in \Delta\}$ and where propagation is done using the *lineage semiring* $\langle \mathcal{P}_{\text{fin}}(\mathbf{D}) \cup \{\bot\}, \cup^+, \cup^\times, \bot \rangle$. Here, $\mathcal{P}_{\text{fin}}(\mathbf{D})$ is the set of all finite subsets of $\mathbf{D}$, $\bot$ is an extra element not in $\mathcal{P}_{\text{fin}}(\mathbf{D})$, and the operations $\cup^+$ and $\cup^\times$ coincide with each other and the usual set union $\cup$ on elements of $\mathcal{P}_{\text{fin}}(\mathbf{D})$, while for any $A \in \mathcal{P}_{\text{fin}}(\mathbf{D}) \cup \{\bot\}$ we have $\bot \cup^+ A = A \cup^+ \bot = A$ and $\bot \cup^\times A = A \cup^\times \bot = \bot$ [4].

Note that $\emptyset$ here represents the fact that a tuple t exists in the database, but is not annotated with any atomic annotation value (i.e. $A_\Delta(t) = \{\Box\}$ in the term representation Δ of the database), and $\bot$ represents the fact that the tuple does not exist in the database (i.e. has $A_\Delta(t) = \emptyset$).

The second semantics of propagation of simple annotations we consider is *Boolean*, which models the behaviour of valid time stamps, beliefs, etc. In this case the *annotated result* of ϕ on Δ besides the term $R(\pi(\mathbf{x}))$ for every embedding π of ϕ to Δ, contains the set of terms

$$\{\mathsf{Annot}(R(\pi(\mathbf{x})), a) \mid \forall i, 1 \le i \le k,\ \mathsf{Annot}(R_i(\pi(\mathbf{y}_i)), a) \in \Delta\}.$$

Essentially, it means that a tuple in the result has an atomic annotation iff *all* source tuples have this annotation for some embedding which gives this tuple on free variables. The Boolean semantics of RUCQs is the same as for lineage.

In the semiring model, the Boolean semantics corresponds again to the setting where each database tuple t is associated to the annotation set $\alpha(t) = \{a \mid \mathsf{Annot}(R(\pi(\mathbf{x})), a) \in \Delta\}$ and where propagation is done using *Boolean algebra*[2] semiring $\langle \mathcal{P}_{\text{fin}}(\mathbf{D}) \cup \{\bot\}, \cup, \cap, \bot \rangle$. The operations $\cup$ and $\cap$ are usual set-theoretic union and intersection on $\mathcal{P}_{\text{fin}}(\mathbf{D})$ and are extended to $\bot$ and any $A \in \mathcal{P}_{\text{fin}}(\mathbf{D})$ as in the lineage semiring: $\bot \cup A = A \cup \bot = A$, $\bot \cap A = A \cap \bot = \bot$. Note, that this definition differs from the standard algebraic definition of Boolean algebra, since the latter does not have $\bot$ and uses $\emptyset$ as a neutral element. We introduce this deviation for the same reason as it is used in the lineage semiring: $\bot$ represents the fact that the tuple in not present, while $\emptyset$ annotation represents the fact that the tuple is present, but has no annotation. However, we still call this semiring *Boolean algebra*, even if it is not in the algebraic sense.

The following example illustrates both semantics.

Example 3.6. Consider the term representations of two annotated relational instances obtained from the relations R_W and R_H in Fig. 1 by disregarding one of annotation attributes. The first one has only VT simple annotations, which has Boolean semantics, and the second – only Comm, which has lineage semantics. Then it is readily verified that the annotated results of the RCQ Φ_1 of the form

$$R_1(x, y, z, u) \leftarrow R_W(x, y, z) \wedge R_H(x, y, u)$$

on these two instances are (the term representation of) the relation R_1 in Fig. 1 (in the VT and Comm columns, respectively).

Now consider the RCQ Φ_2 of the form

$$R_2(x, y) \leftarrow R_W(x, y, z) \wedge R_H(x, y, u)$$

that, like Φ_1, computes the natural join of R_W and R_H, but also projects on Id and Name. Note, that this query is just a reformulation of the TCQ ψ from Ex. 3.3. The annotated results are given in the relation R_2 in Fig. 1. ■

3.3 Propagating of Annotations through Term Conjunctive Queries

We now turn to the automatic propagation of hierarchical annotations through TUCQs. The crucial idea is that annotations are distinguished from data not *statically*, as is the case in the relational semiring models [16, 20], but *dynamically*, depending on the query under consideration and, moreover, the particular embedding from the query into the instance. In other words, the image of the body of the query under the embedding defines the data that is queried, and everything "above" the image is considered as annotations to be propagated automatically.

[2]Not to be confused with Boolean provenance polynomials semiring from [14], which models not simple annotations.

Since, as the previous subsection illustrates, we may want to have different propagation semantics for different types of annotation, we need the concept of a *propagation schema*.

DEFINITION 3.7. A *propagation schema* $\mathcal{P}$ is a partition of the set $\{(F,i) \mid F \in \mathcal{F}, 1 \leq i \leq Arity(F)\}$ into two disjoint sets $\mathcal{L}$ and $\mathcal{B}$.

Intuitively, a pair (F,i) is in $\mathcal{L}$ if F behaves like lineage semantics in the case when the i'th argument of F is considered as data, but F itself with its other arguments, as well as everything "above", is considered as annotation. Similarly, (F,i) is in $\mathcal{B}$ if F has Boolean semantics in such a case.

EXAMPLE 3.8. To model the propagation of comments and valid time as illustrated in Fig. 1 in the introduction, we fix the propagation schema $\mathcal{P}$ where $(\mathsf{Comm},1)$ belongs to $\mathcal{L}$ and the $(\mathsf{VT},1)$ – to $\mathcal{B}$. For completeness, all other pairs belong to $\mathcal{B}$. ■

To ease notation in what follows, we assume $\mathcal{P}$ to be arbitrarily fixed for the rest of this paper.

Recall from Def. 2.6 that we take a term t in an instance Δ to be annotated by the set of data contexts $A_\Delta(t)$, and that we write $\mathcal{H}$ for the set of all finite subcontext-closed set of data contexts. Since we are going to propagate such annotations through queries we need to define two operations on $\mathcal{H}$. One should correspond to union and projection, and another to join. Similarly to the relational case, we would like this propagation to be invariant to standard rearrangement of queries, e.g. swapping two conjuncts, so we also need to show that $\mathcal{H}$ with these operations forms a semiring ([16]).

The first operation, corresponding to union and projection, is just a usual union of sets of contexts from $\mathcal{H}$. By a naive definition of the second operation, which corresponds to join, one would intersect sets of contexts on levels of annotations with Boolean semantics (i.e. with symbols from $\mathcal{B}$ group) and union such sets on levels with lineage semantics (i.e. with symbols from $\mathcal{L}$). However, we opt to a more elaborate definition, which is justified by the following example.

EXAMPLE 3.9. Let us reconsider the TCQ ψ of the form

$$\mathsf{WeightAndHeight}(x) \leftarrow \mathsf{Weight}(x,y_1) \wedge \mathsf{Height}(x,y_2)$$

from Ex. 3.3 as well as the instance Δ from Ex. 2.4 and the propagation schema $\mathcal{P}$ of Ex. 3.8. The (usual) answer to this query is $\mathsf{WeightAndHeight}(joe)$. According to the lineage and Boolean semantics above, the annotated answer to this query should also contain the following data terms on the first level of annotations:

$$\mathsf{VT}(\mathsf{WeightAndHeight}(joe),T_2),\\ \mathsf{Comm}(\mathsf{WeightAndHeight}(joe),C_1),\\ \mathsf{Comm}(\mathsf{WeightAndHeight}(joe),C_2).$$

Note that, since $(\mathsf{VT},1) \in \mathcal{B}$, the set of valid time annotations of *joe*'s height (namely $\{T_2\}$) and weight (namely $\{T_1,T_2,T_3\}$), are intersected when generating the valid time annotation for $\mathsf{WeightAndHeight}(joe)$. Similarly, the set of comment annotations of *joe*'s height (namely $\{C_1\}$) and weight (namely $\{C_1,C_2\}$) are united since $(\mathsf{Comm},1) \in \mathcal{L}$.

By the naive definition of the second operation described above, the annotated answer should not contain any terms of second level of annotations. In particular, the valid time annotation sets $\{T_3\}$ and $\{T_4\}$ would intersect for the term $\mathsf{Comm}(\mathsf{WeightAndHeight}(joe),C_1)$ and give $\emptyset$, since $(\mathsf{VT},1) \in \mathcal{B}$. However, this result is unsatisfactory, since there is no reason why we should lose valid times of the comment C_1. Hence, the following definition of the second operation also gives the terms in the annotated answer to ψ:

$$\mathsf{VT}(\mathsf{Comm}(\mathsf{WeightAndHeight}(joe),C_1),T_3),\\ \mathsf{VT}(\mathsf{Comm}(\mathsf{WeightAndHeight}(joe),C_1),T_4). \quad ■$$

The formalization of the intuition from this example requires the following auxiliary definitions.

DEFINITION 3.10. Given a context c, consider the directed path from its root to the $\Box$ leaf. If all the pairs (F,i) along this path, where F is the symbol in the node and i is the number of the sub-tree containing $\Box$, are in $\mathcal{B}$, then we call this context *Boolean*.

Note in particular that the context $\Box$ is always Boolean.

EXAMPLE 3.11. By the propagation schema from Ex. 3.8, the context $\mathsf{VT}(\Box,T_1)$ is Boolean, but $\mathsf{Comm}(\Box,C_1)$ and $\mathsf{VT}(\mathsf{Comm}(\Box,C_1),T_3)$ are not. ■

Note that for every context c there exist unique contexts c' and c'' such that $c = \{\Box \mapsto c''\}(c')$ and

1. c' is either the trivial context $\Box$ or has a pair from $\mathcal{L}$ just above $\Box$;
2. c'' is Boolean.

We write this fact as $c = c' \langle c'' \rangle$.

EXAMPLE 3.12. For the contexts from Ex. 3.11 we have

$$\mathsf{VT}(\Box,T_1) = \Box\langle \mathsf{VT}(\Box,T_1)\rangle,\\ \mathsf{VT}(\mathsf{Comm}(\Box,C_1),T_3) = \mathsf{VT}(\mathsf{Comm}(\Box,C_1),T_3)\langle\Box\rangle,\\ \mathsf{VT}(\mathsf{Comm}(\mathsf{VT}(\Box,T_1),C_1),T_3) =\\ \mathsf{VT}(\mathsf{Comm}(\Box,C_1),T_3)\langle \mathsf{VT}(\Box,T_1)\rangle. \quad ■$$

DEFINITION 3.13. Given $A_1, A_2 \in \mathcal{H}$ we define

$$A_1 \sqcap A_2 = \{c \mid c \in A_1 \cup A_2, c = c'\langle c''\rangle, \text{ and } c'' \in A_1 \cap A_2\}.$$

The idea behind this definition comes from the intuition given in Ex. 3.9. Indeed, the valid time annotations of the data form Boolean contexts, so we should take their intersection for join. The comment annotations of the data form contexts $\mathsf{Comm}(\Box,C)\langle\Box\rangle$, so we should take their union. The valid time annotations of the comments (as for any annotations "above" comments, no matter which group their symbol belongs to) form contexts $\mathsf{VT}(\mathsf{Comm}(\Box,C),T)\langle\Box\rangle$, so we should take their union.

It is readily verified that, since A_1 and A_2 are closed under subcontexts, so is $A_1 \sqcap A_2$. In other words, $\sqcap$ is a binary operation on $\mathcal{H}$.

PROPOSITION 3.14. *The structure $\langle \mathcal{H}, \cup, \sqcap, \emptyset\rangle$ is a commutative semiring with idempotent $\cup$ and $\sqcap$.*

We call $\langle \mathcal{H}, \cup, \sqcap, \emptyset\rangle$ the *semiring of hierarchical annotations (relative to $\mathcal{P}$)*, or simply the *hierarchical semiring*.

Finally, we are ready to give an annotation semantics for TCQs and TUCQs over annotated instances.

DEFINITION 3.15. The *annotated result* $\psi^+(\Delta)$ of a TCQ ψ of the form $F(\mathbf{x}) \leftarrow \tau_1 \wedge \ldots \wedge \tau_k$ on an instance Δ is defined as the set of terms

$$\{\{\square \mapsto \pi(F(\mathbf{x}))(c) \mid \pi \text{ is an embedding of } \psi \text{ into } \Delta, \\ c \in A_\Delta(\pi(\tau_1)) \sqcap \ldots \sqcap A_\Delta(\pi(\tau_k))\}.$$

The *annotated result* $\Psi^+(\Delta)$ of a TUCQ $\Psi = \{\psi_1, \ldots, \psi_m\}$ on Δ is defined as $\psi_1^+(\Delta) \cup \cdots \cup \psi_m^+(\Delta)$.

We write Δ_ψ^+ for the set $\Delta \cup \psi^+(\Delta)$ and Δ_Ψ^+ for the set $\Delta \cup \Psi^+(\Delta)$. As for the non-annotated case, these sets are subterm-closed, i.e. instances.

It is important to note that, since $\square \in A_\Delta(t)$ for every t in Δ, we always have $\square \in A_\Delta(\pi(\tau_1)) \sqcap \ldots \sqcap A_\Delta(\pi(\tau_k))$. It readily follows that $\Delta_\Psi \subseteq \Delta_\Psi^+$, i.e. the annotated result of any TUCQ contains all the data terms from the usual result. However, it contains also higher level annotating terms which have been automatically propagated through the TUCQ according to the propagation schema.

The fact that the hierarchical semiring is an idempotent semiring justifies the correctness of the definition of annotated result of a query, in the sense that such a result does not depend on the order of terms in TCQs, order of TCQs in TUCQs, elimination of duplicated terms in TCQs, and other equivalent query transformations (see the details in [16]).

The definition above generalizes the lineage and Boolean semantics of relational annotated databases. Also, it conforms to the subtlety of Ex. 3.9, where comments were considered as *annotations* and their valid times unioned for the join in the query. In the following example comments are considered as *data* and their valid time annotations are intersected for a join. As the title of this paper suggests, we are treating valid time as annotations relative to comment annotations.

EXAMPLE 3.16. Consider the TCQ ψ' of the form

$$\mathsf{WHComm}(x) \leftarrow \mathsf{Comm}(\mathsf{Weight}(y_1, y_2), x) \wedge \\ \mathsf{Comm}(\mathsf{Height}(y_3, y_4), x),$$

which asks for all comments used for both weight and height. The query ψ' under annotation semantics augments the instance Δ from Ex. 2.3 and 2.4 with the term $\mathsf{WHComm}(C_1)$. Note, that the augmented instance $\Delta_{\psi'}^+$ contains none of the terms $\mathsf{VT}(\mathsf{WHComm}(C_1), T_3)$ and $\mathsf{VT}(\mathsf{WHComm}(C_1), T_4)$, since in this case these time stamps are in such a position in the hierarchical annotation of the weight comment C_1, that their monadic sets should be intersected. ■

At the end of this section we address the complexity of annotated query answering. It turns out that it is the same as for usual semantics.

PROPOSITION 3.17. *Given a complete representation of an annotated instance Δ, a TUCQ Ψ and a data term t, it is NP-complete to check whether $t \in \Delta_\Psi^+$. It is in* LOGSPACE *if Ψ is fixed. The complexity does not change if Δ is given in incomplete representation.*

3.4 Annotating Datalog Programs

We next move to annotation semantics for datalog programs over hierarchical annotations, which we motivate by means of the following example.

EXAMPLE 3.18. Consider the annotation schema $\mathcal{A}$ that contains, among other constraints, the constraints

$$\mathsf{Believe}(\mathsf{Person}(\mathbf{v}), x) \text{ and } \mathsf{Trust}(\mathsf{Person}(\mathbf{v}_1), \mathsf{Person}(\mathbf{v}_2)),$$

where $\mathbf{v}, \mathbf{v}_1$, and $\mathbf{v}_2$ are tuples of distinct domain variables of the size of $\mathit{Arity}(\mathsf{Person})$. The following rule looks as a TCQ except that we want to evaluate it recursively, as a relational datalog rule:

$$\mathsf{Believe}(x_1, x_2) \leftarrow \mathsf{Trust}(x_1, y) \wedge \mathsf{Believe}(y, x_2). \quad ■$$

We would like to adapt our approach of propagating hierarchical annotations to the setting of recursive positive datalog programs to deal with rules like in this example. We take a simplified approach and do not distinguish between intensional and extensional databases. Recall that $\mathcal{T}_\mathbf{D}(\mathcal{A})$ is the set of all data terms conforming to the schema $\mathcal{A}$.

DEFINITION 3.19. Let $\mathcal{A}$ be an annotation schema. A *(positive) term datalog program* (*TDP* for short) Π is a finite set of TCQs. The *immediate consequence* of Π on an instance Δ conforming to $\mathcal{A}$ is the instance

$$\Delta_{\Pi,1} := \Delta \cup \bigcup_{\psi \in \Pi} (\psi(\Delta) \cap \mathcal{T}_\mathbf{D}(\mathcal{A})).$$

The *result* Δ_Π of Π on Δ is the least fixpoint of the sequence

$$\Delta_{\Pi,0} \subseteq \Delta_{\Pi,1} \subseteq \Delta_{\Pi,2} \subseteq \ldots$$

where $\Delta_{\Pi,0} = \Delta$ and $\Delta_{\Pi,N+1} = (\Delta_{\Pi,N})_{\Pi,1}$ for $N > 1$.

Sometimes TCQs in TDPs are called *rules*.

Note that given a TCQ ψ and the TDP $\Pi = \{\psi\}$, the sets Δ_ψ and $\Delta_{\Pi,1}$ may be different. Indeed, the immediate consequence $\Delta_{\Pi,1}$ depends on the annotation schema $\mathcal{A}$ and contains only those terms from Δ_ψ that conform to $\mathcal{A}$. Hence, both the immediate consequence and the result of Π on Δ are instances, which conform to $\mathcal{A}$.

The least fixpoint of the above sequence is guaranteed to be finite, since we only allow heads of the form $F(\mathbf{x})$ in rules. Had we allowed arbitrary terms in the heads, a finite fixpoint is not guaranteed to exist, and moreover, deciding whether a TDP is *safe*, i.e. has a finite fixpoint on every input, is undecidable [21]. For that reason, we have restricted ourselves to TCQ rules with heads of the form $F(\mathbf{x})$.

The semantics of term datalog programs which propagate annotations can now be defined as follows.

DEFINITION 3.20. Let $\mathcal{A}$ be an annotation schema, Π be a TDP, and Δ be an instance over $\mathcal{A}$. The *annotated immediate consequence* of Π on Δ is the instance

$$\Delta_{\Pi,1}^+ := \Delta \cup \bigcup_{\psi \in \Pi} (\psi^+(\Delta) \cap \mathcal{T}_\mathbf{D}(\mathcal{A})).$$

The *annotated result* Δ_Π^+ of Π on Δ relative to $\mathcal{A}$ is the least fixpoint of the sequence

$$\Delta_{\Pi,0}^+ \subseteq \Delta_{\Pi,1}^+ \subseteq \Delta_{\Pi,2}^+ \subseteq \ldots$$

where $\Delta_{\Pi,0}^+ = \Delta$ and $\Delta_{\Pi,N+1}^+ = (\Delta_{\Pi,N}^+)_{\Pi,1}^+$ for $N > 1$.

As with the usual case, it holds that $\Delta_{\Pi,1}^+ \subseteq \Delta_\psi^+$ for a TCQ ψ and TDP $\Pi = \{\psi\}$, but the inclusion may be strict.

We already mentioned that, since the heads of TCQ rules are restricted, a TDP always has a finite least fixpoint under the normal semantics. The next proposition says that this fact is also true for the annotation semantics.

PROPOSITION 3.21. *Given an annotation schema $\mathcal{A}$, for any TDP Π and any annotated instance Δ over $\mathcal{A}$ there exists a number $N_0 \geq 0$ such that $\Delta^+_{\Pi,N_0+1} = \Delta^+_{\Pi,N_0}$.*

This proposition and the notes above guarantees that the annotated result Δ^+_Π of a TDP Π on an instance Δ conforming to the schema $\mathcal{A}$, also conforms to $\mathcal{A}$.

Since the hierarchical semiring is idempotent, the definition of annotated result of a TDP is correct (in the sense that the result is invariant under the usual query transformations), and, by the results of [16], this semiring can be used as an annotation domain for relational positive datalog programs.

Due to the restriction on heads of TCQ rules, both the combined and data complexity of the evaluation problem is the same as for usual relational positive datalog ([17, 22]).

PROPOSITION 3.22. *Checking whether $t \in \Delta^+_\Pi$ for a TDP Π, a complete or incomplete instance Δ, and a data term t is* EXP*-complete. It is* P*-complete if Π is fixed.*

Apart from computing recursive answers, the main difference between term datalog programs and (unions of) term conjunctive queries is that datalog programs are always guaranteed to return instances conforming to $\mathcal{A}$. This is useful in scenarios such as the one in the following example.

EXAMPLE 3.23. Consider the annotation schema $\mathcal{A}$ from Ex. 2.4 augmented with the constraints

$$\mathsf{Bike}(v_1, v_2) \quad \text{and} \quad \mathsf{Weight}(\mathsf{Bike}(v_1, v_2), v_3),$$

where the first parameter of Bike is the ID of this bike and the second one is the ID of the owner of this bike. If the annotated instance Δ from Ex. 2.4 is augmented with the data terms

$$\mathsf{Bike}(222, 123) \quad \text{and} \quad \mathsf{Weight}(\mathsf{Bike}(222, 123), 70),$$

then the annotated result of the TCQ ψ_4 of the form

$$\mathsf{BOwner}(x_1, x_2) \leftarrow \mathsf{Bike}(x_1, y) \wedge \mathsf{Person}(y, x_2),$$

which asks for names of bike owners, contains the terms

$$\mathsf{BOwner}(222, \text{"Joe"}) \text{ and } \mathsf{Weight}(\mathsf{BOwner}(222, \text{"Joe"}), 70),$$

i.e. it propagates weight annotation, no matter to which group the pair (Weight, 1) belongs. Such a propagation is unsatisfactory in this case because these are really completely different weights. To deal with this we can disallow, in $\mathcal{A}$, that a Weight annotation occurs above BOwner, and consider the TCQ ψ_4 as a TDP of just one TCQ. Then as desired, the annotated result of this TDP does not contain the Weight(BOwner(222, "Joe"), 70), since this term does not conform to the schema. ■

4. RELATIONAL REPRESENTATION FOR HIERARCHICAL ANNOTATIONS

In this section we consider a translation from our term model into the relational model. Since we do not bound the height of terms beforehand, the potential number of relations in the resulting representation can be infinite. However, since we consider only finite term instances, we can always find a finite set of relations which we need to store and manipulate a given instance. That is why w.l.o.g. we consider infinite relational schemas and queries here.

4.1 Representation of the Model

We start with some auxiliary definitions. A constraint α is *simple* if it does not contain variables from $\mathcal{X}$ (i.e. variables which range over terms). Note that every subterm α' of a simple constraint α is again a simple constraint. We call α' a *sub-constraint* of α.

The set *Simple*($\mathcal{A}$) of *simple constraints* induced by an annotation schema $\mathcal{A}$ is the smallest set satisfying:

- all simple constraints in $\mathcal{A}$ are in *Simple*($\mathcal{A}$);
- if α is a constraint in $\mathcal{A}$ and π is a substitution from *Vars*(α)$\cap\mathcal{X}$ to *Simple*($\mathcal{A}$) then $\pi(\alpha)$ is also in *Simple*($\mathcal{A}$).

Intuitively, *Simple*($\mathcal{A}$) is the recursive unwinding of $\mathcal{A}$ into a (possibly infinite) set of simple constraints. Clearly, an instance Δ conforms to $\mathcal{A}$ iff it conforms to *Simple*($\mathcal{A}$).

A simple constraint α_1 is *unifiable* to a simple constraint α_2 if there exists a mapping on the domain variables $\pi\colon \mathcal{V} \to \mathcal{V}$ such that $\pi(\alpha_1) = \alpha_2$. For example, $F(v_1, v_2)$ is unifiable to $F(v_3, v_3)$, but $F(v_3, v_3)$ is not unifiable to $F(v_1, v_2)$. We write $\alpha_1 \succeq \alpha_2$ to indicate that α_1 is unifiable to α_2, and $\alpha_1 \simeq \alpha_2$ to indicate that both $\alpha_1 \succeq \alpha_2$ and $\alpha_2 \succeq \alpha_1$. A simple constraint α from *Simple*($\mathcal{A}$) is *maximal* in *Simple*($\mathcal{A}$) if for every $\alpha' \in$ *Simple*($\mathcal{A}$) with $\alpha' \succeq \alpha$ it holds that $\alpha' \simeq \alpha$.

We will consider maximal simple constraints up to renaming of variables $\mathcal{V}$. To this end we fix for every annotation schema $\mathcal{A}$ an arbitrary set of simple constraints $\mathcal{S}_\mathcal{A}$ such that for every maximal constraint α in *Simple*($\mathcal{A}$) there exists exactly one $\alpha' \in \mathcal{S}_\mathcal{A}$ such that $\alpha \simeq \alpha'$. As an exception, it will be convenient not to consider trivial constraints $v \in \mathcal{V}$ to be elements of $\mathcal{S}_\mathcal{A}$ (since these constraints corresponds to the domain). In addition, we fix some order on variables (strictly speaking, on positions of variables) in every maximal simple constraint α and denote the resulting tuple of variables by $\mathbf{v}_\alpha$.

The set $\mathcal{S}_\mathcal{A}$ can still be seen as a representation of the annotation schema $\mathcal{A}$, since an instance Δ conforms to $\mathcal{A}$ iff it conforms to $\mathcal{S}_\mathcal{A}$. Note that $\mathcal{S}_\mathcal{A}$ may be infinite if $\mathcal{A}$ is recursive. If, however, we fix the maximal height of a maximal simple constraint, it becomes finite. The relational representation of an annotation schema is based on these observations.

Recall that a *relational schema* is a schema that contains only a technical constraint v and constraints of height 1 (called *relations*) without data values, term variables, and repeating data variables. A *relational instance* is an instance which conforms a relational schema. In what follows we do not mention the constraint v in relational schemas and domain values in relational instances, but always assume their presence.

DEFINITION 4.1. A (possibly infinite) relational schema $\mathbf{R}[\mathcal{A}]$ is a *relational representation* of an annotation schema $\mathcal{A}$ if it contains a relation name $R[\alpha]$ for every maximal simple constraint from $\mathcal{S}_\mathcal{A}$. The arity of the relation $R[\alpha]$ is the size of the tuple of variables $\mathbf{v}_\alpha$.

The sub-schema $\mathbf{R}^h[\mathcal{A}]$ of $\mathbf{R}[\mathcal{A}]$ is an *h-height bounded relational representation* of $\mathcal{A}$ for some $h > 0$, if it consists of all relations $R[\alpha]$ for which *Height*(α) $\leq h$.

Clearly, h-height bounded relational representations are always finite, but may contain exponential number of relations in the number of constraints in the annotation schema $\mathcal{A}$ (and in the number h).

EXAMPLE 4.2. In our running example, the constraints for Person, Weight, Height, and BMI are maximal simple, so have corresponding relations in $\mathbf{R}[\mathcal{A}]$. The last of them has arity 5, since the constraint has 7 occurrences of 5 different variables. Each of the constraints for VT has a single maximal simple constraint and a corresponding relation. The constraint for Comm is recursive, so it generates infinite number of maximal simple constraints and, hence, also relations. ■

Every maximal simple constraint α from $\mathcal{S}_\mathcal{A}$ identifies a set of data terms that conform to α in an annotated instance Δ over $\mathcal{A}$. Also, every data term in the instance conforms to a maximal simple constraint from $\mathcal{S}_\mathcal{A}$. It is possible that a data term conforms to two different maximal simple constraints: e.g. the term $F(d,d,d)$ conforms to both of the constraints (which are also maximal simple ones) of the annotation schema $\{F(v_1,v_1,v_2), F(v_1,v_2,v_2)\}$. However, this situation is hardly frequent in any real settings and also does not harm the following exposition.

DEFINITION 4.3. Given an annotated instance Δ over an annotation schema $\mathcal{A}$, a relational instance $\mathbf{I}[\Delta]$ over the relational schema $\mathbf{R}[\mathcal{A}]$ is a *relational representation* of Δ if for every data term t from Δ, which is not an element of the domain $\mathbf{D}$, and every maximal simple constraint α from $\mathcal{S}_\mathcal{A}$, which is conformed by t by the mapping π (i.e. $t = \pi(\alpha)$), the instance $\mathbf{I}[\Delta]$ contains the tuple $\mathbf{d}[t] = \pi(\mathbf{v}_\alpha)$ in the relation $R[\alpha]$.

The relational sub-instance $\mathbf{I}^h[\Delta]$ of $\mathbf{I}[\Delta]$ which consists of all the tuples in relations of the schema $\mathbf{R}^h[\mathcal{A}]$ is called a *h-height bounded relational representation* of Δ.

By the observation above, a data term can have several corresponding tuples in the relational representation of an annotated instance. So, if the annotation schema is fixed, a h-height bounded relational representation of a term annotated instance can contain non-linear (but polynomial) number of tuples in the size of the annotated instance. However, as mentioned before, such a blowup is not realistic.

EXAMPLE 4.4. The relational representation $\mathbf{I}[\Delta]$ of the annotated instance Δ from Ex. 2.3 and 2.4 contains, e.g. the facts $R[\mathsf{VT}(\mathsf{Weight}(\mathsf{Person}(v_1,v_2),v_3),v_4)](123,\text{"Joe"},70,T_2)$ and $R[\alpha](123,\text{"Joe"},90,160,\text{"High"})$, where α is the constraint for BMI. ■

The following proposition shows that under the above representation of annotated instances as relational instances, annotations are represented by means of full inclusion dependencies. Recall that a *full inclusion dependency* σ (or *fid*) over a relational schema $\mathbf{R}$ is an expression of the form

$$R(\mathbf{u}) \to R'(\mathbf{u}'),$$

where R and R' are relational names from $\mathbf{R}$, $\mathbf{u}$ is a tuple of distinct variables and $\mathbf{u}'$ is a tuple of variables from $\mathbf{u}$. A relational instance $\mathbf{I}$ *satisfies* a fid σ iff for every mapping $\pi : \mathbf{u} \to \mathbf{D}$ it holds that $\pi(R(\mathbf{u})) \in \mathbf{I}$ implies $\pi(R'(\mathbf{u}')) \in \mathbf{I}$. A set of fids is *acyclic* if it does not contain a chain of fids $R(\mathbf{u}) \to R_1(\mathbf{u}_1), R_1(\mathbf{u}'_1) \to R_2(\mathbf{u}_2), \ldots, R_n(\mathbf{u}'_n) \to R(\mathbf{u}')$.

PROPOSITION 4.5. *Let $\mathcal{A}$ be an annotation schema and Δ an annotated instance over this schema.*

1. *Let α_1 and α_2 be distinct maximal simple constraints from $\mathcal{S}_\mathcal{A}$ for which there exist a sub-constraint α'_1 of α_1 and a mapping $\pi : \mathcal{V} \to \mathcal{V}$ such that $\pi(\alpha_2) = \alpha'_1$. Then the full inclusion dependency*

$$R[\alpha_1](\mathbf{v}_{\alpha_1}) \to R[\alpha_2](\pi(\mathbf{v}_{\alpha_2}))$$

holds in $\mathbf{I}[\Delta]$.

2. *The set of all these fids is acyclic.*

Moreover, given an annotated schema $\mathcal{A}$, if some fid holds in the relational representation of every annotated instance Δ over $\mathcal{A}$, then this fid is implied by the set of fids described in this proposition. Using these facts, one may consider yet another representation of an annotation schema $\mathcal{A}$, which we call *fid representation*. This is also a (possibly infinite) relational schema, which is isomorphic to $\mathbf{R}[\mathcal{A}]$, but instead of an explicit correspondence between maximal simple constraints and relations' names, it contains a set of fids (and the relations are "anonimized").

There are several questions that arise about fid representations. The first important problem is to understand whether a relational schema with a set of fids indeed has an annotation schema to represent as a fid representation. It is not that difficult to check that for any relational schema and any acyclic set of fids over this schema there exists an annotation schema that is represented by the relational schema and the set of fids. Acyclicity can be checked in linear time.

Another problem is to recover an annotation schema from its fid representation. However, several annotation schemas may have the same fid representation. Indeed, if we have four relations R_1, R_2, R_3, and R_4 with two fids $R_1 \to R_2$ and $R_3 \to R_4$, nothing tells us whether the anonimized maximal simple constraints of the relations, isomorphic to R_1 and R_3 have the same symbol in the head or different. Hence, full unambiguous recovering is not possible. However, it might be important just to understand whether a relation annotates another relation. In this case of course we do not need to consider the whole (possibly infinite) set of fids, but just the finite part of it which involves these relations and relations "below" them (recall that the set of fids is acyclic). The corresponding true-false question is the following.

Input: a (finite) relational schema $\mathbf{R}$, a set of fids Σ over it, and two relation names R_1 and R_2 from $\mathbf{R}$.

Question: is it true that for any annotation schema $\mathcal{A}$ with a fid representation $(\mathbf{R}', \Sigma')$ such that $\mathbf{R}$ is a sub-schema of $\mathbf{R}'$ and Σ is a subset of Σ', it holds that for relations $R[\alpha_1]$ and $R[\alpha_2]$ from $\mathbf{R}[\mathcal{A}]$, which are the isomorphic images of R_1 and R_2, it holds that α_1 is a sub-constraint of α_2?

It turns out, that this problem is equivalent to the *implication problem* for a set of fids, i.e. the problem of deciding whether a set of fids implies another fid. The following theorem says that, somehow counterintuitive, this problem is intractable. With author's permission, we refer this theorem to a personal communication.

THEOREM 4.6 ([11]). *Implication problem for an acyclic set of full inclusion dependencies is* NP*-complete.*

An immediate corollary is that even if the relational representation of an annotation schema is finite, its fid representation can be exponentially more succinct.

Similarly to the term instances themselves, relational and fid representations can be either complete or incomplete. In

the following section this will be important for algorithms of of evaluation of queries over usual and annotation semantics.

4.2 Representation of Term Queries

Next we show that it is possible to represent term UCQs over annotated instances as sets of relational UCQs over relational representations of these instances, first for the usual and then for the annotation semantics.

Let Θ be a set of RUCQs. Denote $\mathbf{I}_\Theta$ the relational instance $\mathbf{I}$ enriched with $\Phi(\mathbf{I})$, for every $\Phi \in \Theta$.

PROPOSITION 4.7. *Let $\mathcal{A}$ be an annotation schema and let Ψ be a TUCQ. Let $F(\mathbf{x})$ be the unique head of the TCQs in Ψ. There exists a (possibly infinite) set of (possibly infinite) RUCQs $\Theta[\Psi]$ over the relational schema $\mathbf{R}[\mathcal{A} \cup \{F(\mathbf{x})\}]$, such that for every annotated instance Δ over $\mathcal{A}$, every data term t, and every maximal simple constraint α from $\mathcal{S}_\mathcal{A}$ to which t conforms, the following is equivalent*

- $t \in \Delta_\Psi$,
- $R[\alpha](\mathbf{d}[t]) \in \mathbf{I}[\Delta]_{\Theta[\Psi]}$.

The construction of $\Theta[\Psi]$ is similar in spirit to the construction used to simulate nested relational algebra expressions by means of flat relational algebra expressions [8]. The main difference is because annotation schemas, in contrast to nested relational schemas, support sharing of subterms as well as recursion, and because, in contrast to nested relational algebra queries, TUCQS can be applied to annotated instances of unbounded height. Hence, the set $\Theta[\Psi]$ of RUCQs has the following peculiarities:

1. it can be infinite,
2. its RUCQs can contain infinite sets of RCQs,
3. its RUCQs may have different relations in the heads.

The possible infinities of the first two items is clearly a disadvantage of such a representation. While this cannot be avoided in general, the following proposition shows that if we limit input annotated instances by some bounded maximal height h then we can obtain a finite representation by means of a finite set of finite RUCQs.

PROPOSITION 4.8. *Let $\mathcal{A}$ be an annotation schema and let Ψ be a TUCQ. Let $F(\mathbf{x})$ be the unique head of the TCQs in Ψ and let h be a positive number. There exists a finite set of finite RUCQs $\Theta^h[\Psi]$ over the relational schema $\mathbf{R}^h[\mathcal{A} \cup \{F(\mathbf{x})\}]$, such that for every annotated instance Δ over $\mathcal{A}$ containing only data terms of height at most h, every data term t, and every maximal simple constraint α from $\mathcal{S}_\mathcal{A}$ to which t conforms, the following is equivalent*

- $t \in \Delta_\Psi$,
- $R[\alpha](\mathbf{d}[t]) \in \mathbf{I}^h[\Delta]_{\Theta^h[\Psi]}$.

Having this restriction, we can talk about complexity of evaluation of representations of TUCQs. Since the representation $\Theta^h[\Psi]$ may have several RUCQs with different numbers of free variables and different heads, it is a reasonable assumption that an input of a decision problem includes the particular RUCQ in $\Theta^h[\Psi]$ for which we check whether it has a tuple $\mathbf{d}$ in the result over a relational instance $\mathbf{I}^h[\Delta]$. But this implies that if the instance is complete then our decision problem is just a standard UCQ answering problem over relational databases, no matter whether we have relational or fid representation of the annotation schema. It is a little bit more interesting if our instance is incomplete, i.e. we need to answer UCQs with respect to fids. However, the following proposition says that the complexity does not change for incomplete fid representations. The immediate corollary is that it is still the same for incomplete relational representations.

PROPOSITION 4.9. *Let $\mathbf{I}$ be a (finite) relational instance over some (finite) relational schema, Σ be a set of fids, Φ be an RUCQ, and $\mathbf{d}$ be a tuple. Then checking whether $R_\Phi(\mathbf{d}) \in \mathbf{I}'_\Phi$ holds for any relational instance $\mathbf{I}'$, such that $\mathbf{I} \subseteq \mathbf{I}'$ and $\mathbf{I}'$ satisfies all fids in Σ, is* NP*-complete. It is in* P *if Σ and Φ are fixed.*

Completely analogous to Prop. 4.7 and 4.8 the following proposition shows that it is also possible to represent the annotation semantics of TUCQs by means of sets of RUCQs. Given a symbol F of arity n denote $\mathcal{A}_F$ the extension of an annotation schema $\mathcal{A}$ by the constraints obtained from constraints in $\mathcal{A}$ by replacement of every possible subterm by $F(\mathbf{x})$, where $\mathbf{x}$ is a tuple of term variables from $\mathcal{X}$.

PROPOSITION 4.10. *Let $\mathcal{A}$ be an annotation schema and let Ψ be a TUCQ. Let $F(\mathbf{x})$ be the unique head of the TCQs in Ψ.*
1. *There exists a (possibly infinite) set of (possibly infinite) RUCQs $\Theta^+[\Psi]$ over the relational schema $\mathbf{R}[\mathcal{A}_F]$ such that for each annotated instance Δ, data term t, and maximal simple constraint α from $\mathcal{S}_\mathcal{A}$, which t conforms, the following is equivalent*

   - $t \in \Delta^+_\Psi$,
   - $R[\alpha](\mathbf{d}[t]) \in \mathbf{I}[\Delta]_{\Theta^+[\Psi]}$.

2. *For each positive number h there exists a finite set of finite RUCQs $\Theta^{+,h}[\Psi]$ over the relational schema $\mathbf{R}^h[\mathcal{A}_F]$ such that for every annotated instance Δ conforming to $\mathcal{A}$ containing only terms height at most h, every data term t, and every maximal simple constraint α to which t conforms, the following is equivalent*

   - $t \in \Delta^+_\Psi$,
   - $R[\alpha](\mathbf{d}[t]) \in \mathbf{I}^h[\Delta]_{\Theta^{+,h}[\Psi]}$.

As a corollary, query answering over finite representations $\Theta^{+,h}[\Psi]$ of the annotation semantics of TUCQs has the same complexity as for the non-propagating case.

It should be noted that representation of TDPS in terms of relational datalog programs similar to Prop. 4.7, 4.8, and 4.10 is also possible. We forgo the formal proposition due to the technicality and lack of space.

5. RELATED WORK

There is a huge literature on specific kinds of annotation, especially the literature on temporal and belief databases, and what we have presented in this paper in no sense subsumes this work. The first attempt to find a uniform treatment of annotation was the *provenance semirings* of [16], which has been highly influential and has been extended [1, 2, 13, 15] to deal with update, aggregation, and negation, as

well as well as stimulating some practical prototypes. A related, and somewhat more complicated formalism has been developed for RDF/S [23]. The observation that two or more annotations could interact was made in [20], but it still makes a two-level distinction between data and annotation.

As observed in Sec. 4 there is a close relationship between our translation to relational model and the translation in the work [8] on nested relations or complex objects. However, in the complex object model, sets and tuples can be freely combined. In the annotation model we have one top-level, heterogenous, set. What it means to annotate a set is interesting, but future work.

6. CONCLUSIONS

We have argued that there is no sharp distinction between annotation and data, and we have formulated a general model of hierarchical annotation in which what is data and what is annotation depends on both the data and the query that is being applied to the data. We have described a hierarchical term model of annotation that allows for shared substructures. We have described a query language for this model and shown how annotations propagate through queries. We have shown how to "flatten" hierarchical schemas into possibly infinite systems of full inclusion dependencies and translate queries on terms into relational queries accordingly.

However, we feel that we have only scratched the surface of this general problem, and there are several interesting open problems concerning this treatment of annotation.

1. As already observed in Sec. 2, in Def. 2.2 of an annotation schema we made a rather awkward distinction between term and domain variables to keep several decision problems associated with annotation schemas (e.g. such as consistency) tractable. But there are cases in which one would like to express the sharing of arbitrary terms. For this reason, alternative models of constraints should be developed.

2. In our formalization of term datalog programs we did not look at one property of annotation, which is that one might require that adding an annotation to some set of data does not cause that set to change (though it might cause the inference of new annotations). This is a "not influence" relationship between data and annotation that should be captured in any model of annotation.

3. If we really adopt the philosophy that all data is annotation, we now need to account for other familiar properties of data in the annotation schema. The interaction of annotations, partly studied in [20] needs to be carried further. For example, suppose we take Height as an annotation and have terms like $\mathsf{Height}(joe, 180)$ and we also treat Student as an annotation so that we have $\mathsf{Student}(joe)$ one might argue that, according to some rule of inheritance or subclassing that one should also have $\mathsf{Height}(\mathsf{Student}(joe), 180)$. Can one extend annotation schemas to embrace the conventional constructs of database schemas, or is this going too far?

Acknowledgements. We are indebted to Jan Van den Bussche who contributed greatly to the ideas in this paper and to Floris Geerts for his proof of Thm. 4.6.

7. REFERENCES

[1] Y. Amsterdamer, D. Deutch, and V. Tannen. On the limitations of provenance for queries with difference. *CoRR* **abs/1105.2255**, 2011.

[2] Y. Amsterdamer, D. Deutch, and V. Tannen. Provenance for aggregate queries. *PODS* 2011, 153–164.

[3] T. Berners-Lee. *Multiuser considerations.* `http://www.w3.org/DesignIssues/Multiuser.html`.

[4] P. Buneman, S. Khanna, and W. Tan. Why and Where: A Characterization of Data Provenance. *ICDT 2001* 2001, *LNCS* 1973, 316–330. Springer.

[5] L. Chiticariu, W.-C. Tan, and G. Vijayvargiya. DBNotes: a post-it system for relational databases based on provenance. *Proceedings of SIGMOD '05* 2005, 942–944. ACM. `doi:http://doi.acm.org/10.1145/1066157.1066296`.

[6] H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. *Tree Automata Techniques and Applications.* Available on: `http://www.grappa.univ-lille3.fr/tata`, 2007. release October, 12th 2007.

[7] Y. Cui, J. Widom, and J. L. Wiener. Tracing the lineage of view data in a warehousing environment. *ACM Trans. Database Syst.* **25**, 179–227, June 2000. `doi:http://doi.acm.org/10.1145/357775.357777`.

[8] J. V. den Bussche. Simulation of the nested relational algebra by the flat relational algebra, with an application to the complexity of evaluating powerset algebra expressions. *Theor. Comput. Sci.*, 363–377, 2001.

[9] R. D. Dowell, R. M. Jokerst, A. Day, S. R. Eddy, and L. Stein. The Distributed Annotation System. *BMC Bioinformatics* **2**, p. 7, 2001.

[10] W. Gatterbauer, M. Balazinska, N. Khoussainova, and D. Suciu. Believe it or not: adding belief annotations to databases. *Proc. VLDB Endow.* **2**(1), 1–12, Aug. 2009.

[11] F. Geerts. Personal communication, 2010.

[12] F. Geerts, A. Kementsietsidis, and D. Milano. MONDRIAN: Annotating and Querying Databases through Colors and Blocks. *Proceedings of ICDE'06* 2006, p. 82. IEEE Computer Society. `doi:10.1109/ICDE.2006.102`.

[13] F. Geerts and A. Poggi. On database query languages for K-relations. *J. Applied Logic* **8**(2), 173–185, 2010.

[14] T. J. Green. Containment of conjunctive queries on annotated relations. *Theory of Computing Systems* **49**(2), 429–459, 2011. `doi:10.1007/s00224-011-9327-6`.

[15] T. J. Green, G. Karvounarakis, Z. G. Ives, and V. Tannen. Update exchange with mappings and provenance. *VLDB* 2007, 675–686.

[16] T. J. Green, G. Karvounarakis, and V. Tannen. Provenance semirings. *Proceedings of PODS '07* 2007, 31–40. ACM. `doi:http://doi.acm.org/10.1145/1265530.1265535`.

[17] N. Immerman. Relational queries computable in polynomial time. *Information and Control* **68**, 86–104, 1986.

[18] J. Kahan, M.-R. Koivunen, E. Prud'hommeaux, and R. R. Swick. Annotea: an open RDF infrastructure for shared web annotations. *Computer Networks* **39**(5), 589–608, 2002.

[19] C. Koch. On the complexity of nonrecursive XQuery and functional query languages on complex values. *ACM Trans. Database Syst.* **31**(4), 1215–1256, 2006. `doi:10.1145/1189769.1189771`.

[20] E. V. Kostylev and P. Buneman. Combining dependent annotations for relational algebra. *Proceedings of ICDT '12* 2012, 196–207. ACM. `doi:10.1145/2274576.2274597`.

[21] O. Shmueli. Decidability and expressiveness aspects of logic queries. *Proceedings of PODS 87* 1987, 237–249. ACM. `doi:10.1145/28659.28685`.

[22] M. Y. Vardi. The complexity of relational query languages (extended abstract). *Proceedings of STOC '82* 1982, 137–146. ACM. `doi:10.1145/800070.802186`.

[23] A. Zimmermann, N. Lopes, A. Polleres, and U. Straccia. A general framework for representing, reasoning and querying with annotated semantic web data. *Web Semantics: Science, Services and Agents on the World Wide Web* **11**(0), 72–95, 2012. `doi:10.1016/j.websem.2011.08.006`.