Function Symbols in Tuple-Generating Dependencies: Expressive Power and Computability*

Georg Gottlob[1,2], Reinhard Pichler[1], and Emanuel Sallinger[2]

[1]TU Wien and [2]University of Oxford

Tuple-generating dependencies (tgds, for short) have been a staple of database research throughout most of its history. Initially, they provided a unifying notion for database constraints such as inclusion dependencies. Meanwhile, tgds can be found – under various names – as fundamental concepts in several areas of computer science. For instance, in artificial intelligence, tgds occur as existential rules; in logic programming, tgds are the rules that make up the family of languages often called Datalog$^\pm$; in logic, tgds can be described as a particular fragment of Horn first-order logic; etc.

Yet one of the central aspects of tgds, namely the role of existential quantifiers, has not seen much investigation so far. When studying dependencies, existential quantifiers and – in their Skolemized form – function symbols are often seen as two ways to express the same concept. But in fact, tgds are quite restrictive in the way that functional terms can occur. Consider the following tgd based on employees, their departments and the department managers:

$$\forall e, d\ \ \mathsf{Emp}(e, d) \rightarrow \exists dm\, \mathsf{Mgr}(e, dm)$$

It expresses that for every employee e in department d, there exists a department manager dm. To understand the exact form of existential quantification, let us look at its Skolemized form, where the implicit dependence of the existential quantifier is made explicit using function symbols. That is, the variable dm is replaced by a term based on a function symbol f_{dm}.

$$\exists f_{\mathsf{dm}}\, \forall e, d\ \ \mathsf{Emp}(e, d) \rightarrow \mathsf{Mgr}(e, f_{\mathsf{dm}}(e, d))$$

Observe that any functional term contains the full set of universally quantified variables from the antecedent. More concretely, the function f_{dm} representing the department manager depends on both the department and the employee.

In contrast, what we would probably like to express is that the department manager only depends on the department. That is, the dependency

$$\exists f_{\mathsf{dm}}\, \forall e, d\ \ \mathsf{Emp}(e, d) \rightarrow \mathsf{Mgr}(e, f_{\mathsf{dm}}(d))$$

We can show that this dependency cannot be expressed by a logically equivalent set of tgds. However, there are more powerful dependency languages than tgds, most importantly SO tgds [2]. Note that, as originally defined, SO tgds are

* This is an extended abstract of a paper presented at PODS 2015 [4].

required to be so-called source-to-target (s-t) dependencies. In the context of this work, we do not restrict any dependency formalism to s-t unless explicitly mentioned. The key feature of SO tgds is the use of function symbols, and indeed, the above formula is an SO tgd. SO tgds were introduced to capture the composition of tgds. However, the power of SO tgds comes at a cost: many reasoning tasks (such as, e.g., logical equivalence) become undecidable.

Yet, there is a middle ground between tgds and SO tgds: nested tgds [3]. They were introduced as part of IBM's Clio system, which is now part of the InfoSphere BigInsights suite. It has recently been shown that nested tgds have a number of advantages in terms of decidability of reasoning tasks (such as, e.g., the equivalence of s-t nested tgds). Let us now return to our running example. If our schema in addition contains a relation Dep representing departments, then we can express our dependency as the following nested tgd:

$$\forall d\ \mathsf{Dep}(d) \rightarrow \exists dm\, [\forall e\, \mathsf{Emp}(e,d) \rightarrow \mathsf{Mgr}(e,dm)]$$

Nested tgds in Skolemized form can always be "normalized" to a set of unnested implications – each corresponding to an SO-tgd. In our example, this normalization yields the following single implication:

$$\exists f_{\mathsf{dm}}\, \forall e,d\ \ \mathsf{Dep}(d) \wedge \mathsf{Emp}(e,d) \rightarrow \mathsf{Mgr}(e, f_{\mathsf{dm}}(d))$$

We have seen that nested tgds are one way to avoid the complexity of SO tgds, while still being able to model interesting domains. They have however one major restriction: they can only model hierarchical relationships (i.e., the argument lists of Skolem functions must form a tree). Nevertheless, there are natural relationships that cannot be captured by nested tgds. Let us extend our example as follows: for every employee, we want to create an employee ID. We can express this as an SO tgd:

$$\exists f_{\mathsf{eid}}, f_{\mathsf{dm}}\, \forall e,d\ \ \mathsf{Emp}(e,d) \rightarrow \mathsf{Mgr}(f_{\mathsf{eid}}(e), f_{\mathsf{dm}}(d))$$

It can be shown that nested tgds are unable to express this dependency. So how can we gain a more fine-grained understanding of, in general, not hierarchical relationships without resorting to SO tgds? To answer this question, let us look at a well-known formalism from logic which can help us in that regard. In logic, Henkin quantifiers [1] are a tool to gain a fine-grained control over the way function symbols can occur in the Skolemized form of formulas. Let us write our dependency using a Henkin quantifier:

$$\begin{pmatrix} \forall d\, \exists dm \\ \forall e\, \exists eid \end{pmatrix} \mathsf{Emp}(e,d) \rightarrow \mathsf{Mgr}(eid, dm)$$

where the quantifier prefix means that the existential variable *dm* only depends on the department d, and the existential variable *eid* only depends on the employee e. We shall call such Henkin-quantified rules "Henkin tgds". The above tgd contains a particularly simple form of a Henkin quantifier, called a *standard* Henkin quantifier, where the variables in different rows are disjoint and the

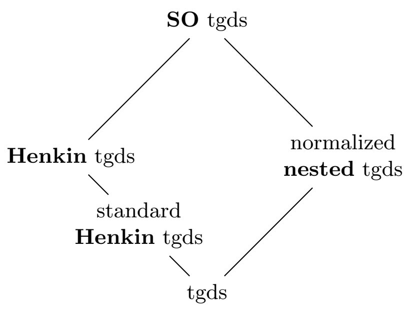

Fig. 1. *Hasse diagram of syntactical inclusions.*

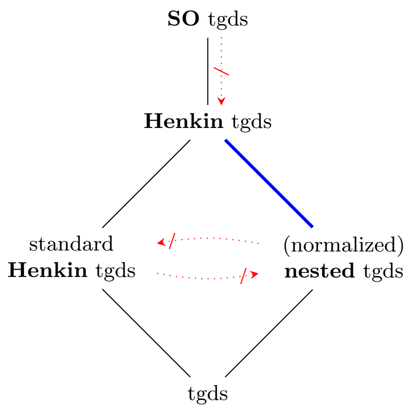

Fig. 2. *Hasse diagram of semantical inclusions.*[2]

quantification in each row consists of universal quantifiers followed by existential ones. We shall refer to such tgds as “standard Henkin tgds”.

Altogether, we have described five classes of tgds so far with tgds as the least expressive and SO tgds as the most expressive. In Figure 1, we summarize these classes of tgds in a Hasse diagram which shows the syntactical inclusion between dependency classes in their Skolemized form. An edge denotes that every set of dependencies of the lower class is also a set of dependencies of the upper class. The **first goal** pursued in this work is to study the relative *expressive power* of these classes of tgds. That is, can we represent sets of tgds from one class as equivalent sets of tgds in another class?

A central task for database systems is *query answering*. Given a database, a set of dependencies, and typically a conjunctive query, the task of query answering is to compute the set of certain answers to that conjunctive query (i.e., the answers common to *all* databases that contain the original database and satisfy the dependencies). However, query answering for tgds is in general undecidable. Hence, numerous criteria for ensuring decidability have been introduced throughout the last few years. Due to lack of space, we cannot recall all of them here. We just mention that there are essentially three families of criteria to ensure decidable query answering in the presence of tgds, namely: *acyclic* tgds, *guarded* tgds, with *linear* tgds as an important special case, and *sticky* tgds. All these classes naturally generalize to SO tgds (and thus also to nested tgds and Henkin tgds), but of course there is no immediate guarantee that query answer-

[2] A solid edge denotes that every set of dependencies from the lower class can be expressed as a logically equivalent set of dependencies from the upper class. The bold (blue) edge highlights the one containment that is not syntactical (i.e., that normalized nested tgds are not a syntactical subset of Henkin tgds). The dotted (red) edges in the Hasse diagram indicate that for a given set of dependencies, not even a CQ-equivalent set of dependencies exists.

ing for these classes is also decidable. Hence, the **second goal** of our work is to pinpoint the decidability/undecidability border for nested, (standard) Henkin, and SO tgds under the three "families" of criteria: acyclic, guarded, and sticky.

Apart from query answering, another central problem with any logical formalism is *model checking*, i.e.: given a database and a set of dependencies, the task of model checking is to decide whether or not the database satisfies all dependencies. For tgds, query and combined complexity are known to be $\Pi_2\mathsf{P}$-complete while – as tgds are first-order formulas – data complexity is in AC_0. For SO tgds, data complexity is NP-complete and query and combined complexity is NEXPTIME-complete. From these results, we can already derive some bounds for other formalisms, since lower bounds propagate along generalizations and upper bounds propagate along specialization. So the **third goal** of our work was to identify the precise (data/query/combined) complexity of model checking for nested tgds and Henkin tgds.

Main Results of This Work

Expressive Power. A complete picture of the relative expressive power of the various classes of tgds is depicted in Figure 2. Interestingly, we obtain that nested tgds can always be transformed into a logically equivalent set of Henkin tgds. In contrast, we show that a number of inclusions are proper for other classes and that two classes (namely standard Henkin tgds and nested tgds) are incomparable.

Query Answering. Our results for query answering are mainly negative. In particular, we show that even when assuming our dependencies to be *both* guarded and sticky, atomic query answering is undecidable for standard Henkin tgds and nested tgds, the two lowest extensions of tgds given in Figure 1. For standard Henkin tgds, undecidability holds even for linear dependencies. In contrast, acyclicity (actually, already the relaxed notion of *weak* acyclicity) guarantees decidability of query answering even for SO tgds. Likewise, imposing a further restriction on linear Henkin tgds leads to decidability.

Model Checking. We show that Henkin tgds are NEXPTIME-complete in query and combined complexity and NP-complete in data complexity. Hardness holds even for standard Henkin tgds. We also show that nested tgds are PSPACE-complete in query and combined complexity, while data complexity clearly is in AC_0. We thus complete the picture of the complexity of model checking for all the dependency classes considered here.

Future work. For future work, the most burning question is how to narrow the gaps between the islands of decidability of query answering under Henkin, nested, and SO tgds. We thus want to analyze known decidable fragments of various logics (such as, e.g., the two-variable fragment) and investigate their applicability to our setting of query answering under tgds. Moreover, we also have to explore new paradigms that are tailor-made for Henkin, nested, and SO tgds and that go beyond known decidability criteria for other logics. Moreover, it would be interesting to see whether frameworks such as independence-friendly logic and dependence logic yield additional suitable subclasses of SO tgds.

Acknowledgements. This work was supported by the Engineering and Physical Sciences Research Council (EPSRC), Programme Grant EP/M025268/ "VADA: Value Added Data Systems – Principles and Architecture" as well as by the Austrian Science Fund (FWF):P25207-N23 and (FWF):P25518-N23.

References

1. A. Blass and Y. Gurevich. Henkin quantifiers and complete problems. *Ann. Pure Appl. Logic*, 32:1–16, 1986.
2. R. Fagin, P. G. Kolaitis, L. Popa, and W. C. Tan. Composing schema mappings: Second-order dependencies to the rescue. *ACM Trans. Database Syst.*, 30(4):994–1055, 2005.
3. A. Fuxman, M. A. Hernández, C. T. H. Ho, R. J. Miller, P. Papotti, and L. Popa. Nested mappings: Schema mapping reloaded. In U. Dayal, K. Whang, D. B. Lomet, G. Alonso, G. M. Lohman, M. L. Kersten, S. K. Cha, and Y. Kim, editors, *Proceedings of the 32nd International Conference on Very Large Data Bases, Seoul, Korea, September 12-15, 2006*, pages 67–78. ACM, 2006.
4. G. Gottlob, R. Pichler, and E. Sallinger. Function symbols in tuple-generating dependencies: Expressive power and computability. In T. Milo and D. Calvanese, editors, *Proceedings of the 34th ACM Symposium on Principles of Database Systems, PODS 2015, Melbourne, Victoria, Australia, May 31 - June 4, 2015*, pages 65–77. ACM, 2015.