

**Datalog+/-: Questions and Answers**

**Citation for published version:**

**Link:**
Link to publication record in Edinburgh Research Explorer

**Document Version:**
Publisher's PDF, also known as Version of record

**Published In:**

---

**General rights**
Copyright for the publications made accessible via the Edinburgh Research Explorer is retained by the author(s) and/or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

**Take down policy**
The University of Edinburgh has made every reasonable effort to ensure that Edinburgh Research Explorer content complies with UK legislation. If you believe that the public display of this file breaches copyright please contact openaccess@ed.ac.uk providing details, and we will remove access to the work immediately and investigate your claim.
Datalog+-: Questions and Answers

Georg Gottlob and Thomas Lukasiewicz and Andreas Pieris

Department of Computer Science, University of Oxford, UK
firstname.lastname@cs.ox.ac.uk

• What is Datalog± and what are its advantages?

Datalog± is a rule-based formalism that combines the advantages of logic programming in Datalog with features for expressing ontological knowledge and advanced data modeling constraints. Datalog± provides a uniform framework for query answering and reasoning with incomplete data. It generalizes disparate other KR formalisms such as plain Datalog, description logics (DLs), in particular, DL-Lite and \(EL\), F-Logic Lite, highly relevant fragments of UML class diagrams, data-exchange formalisms, graph query languages such as SPARQL, and so on. Datalog± is a conceptually very simple and clear-cut formalism that extends plain Datalog with features such as existential quantifiers, equalities, and the falsum (⊥) in rule heads. As will be pointed out later, Datalog± rules can be safely combined with arbitrary equality rules, a.k.a. equality-generating dependencies (tgds).

Such rules are also known as *tuple-generating dependencies* (tgds). All variables that are not existentially quantified are assumed to be universally quantified. We just omit the universal quantifiers in front of such rules. As will be pointed out later, Datalog± rules may also contain equalities and the falsum in rule heads.

• How do Datalog± rules look like?

A self-explanatory set of Datalog± rules follows:

\[
\begin{align*}
\sigma_1 & : \text{emp}(X) \rightarrow \exists Y \text{ hasMgr}(X, Y), \text{emp}(Y), \\
\sigma_2 & : \text{hasMgr}(X, Y), \text{emp}(X) \rightarrow \text{emp}(Y), \\
\sigma_3 & : \text{emp}(X), \text{emp}(Y) \rightarrow \text{colleagueOf}(X, Y), \\
\sigma_4 & : \text{emp}(X) \rightarrow \exists Y \text{ worksIn}(X, Y), \text{dept}(Y).
\end{align*}
\]

Such rules are also known as *tuple-generating dependencies* (tgds). All variables that are not existentially quantified are assumed to be universally quantified. We just omit the universal quantifiers in front of such rules. As will be pointed out later, Datalog± rules may also contain equalities and the falsum symbol (⊥) in their heads.

• Which reasoning tasks are considered with Datalog±?

The main reasoning task is query answering under the so-called *certain-answers semantics*. If \(Q(X)\) is a conjunctive query (CQ) or a union of conjunctive queries (UCQ) with free variables \(X\), \(D\) a database over a domain (universe) \(\Delta\) whose tuples are interpreted in the usual way as ground facts, and \(\Sigma\) a Datalog± program (a.k.a. Datalog± ontology), then the answer to \(Q\) consists of all those tuples \(a\) of \(\Delta\)-elements such that \(D \cup \Sigma \models Q(a)\). For Boolean queries, the answer is, accordingly, *true* or *false*. Other relevant reasoning tasks are instance checking, query containment and consistency (or satisfiability) checking. These latter tasks are, however, easily reduced to Boolean query answering.

• Which are the main decidable Datalog± languages?

There are so far three main paradigms which extend plain Datalog with existential quantifiers in rule heads, and also guarantee the decidability of query answering:

**Weakly-acyclic Datalog±**, based on weakly acyclic tgds introduced in the context of data exchange (Fagin et al. 2005), guarantees the existence of a finite universal model, which in turn implies the decidability of query answering. The rules \(\{\sigma_2, \sigma_3, \sigma_4\}\) given above form a weakly-acyclic set of Datalog± rules. Extensions of weak-acyclicity have been studied, e.g., in (Marnette 2009; Grau et al. 2013).

**Guarded Datalog±** ensures the existence of treelike universal models, which in turn implies the decidability of query answering (Cali, Gottlob, and Kifer 2013). A rule is guarded if it has an atom which contains all the body-variables; e.g., the rules \(\{\sigma_1, \sigma_2, \sigma_4\}\). An important subclass of guarded Datalog± is linear Datalog±, where rules have only one body-atom, e.g., the rules \(\{\sigma_1, \sigma_4\}\). Extensions of guardedness have been studied in (Baget et al. 2011).

**Sticky Datalog±** guarantees the termination of backward resolution, and thus the decidability of query answering (Cali, Gottlob, and Pieris 2012). The key idea underlying stickiness is that the body-variables which are in a join always are propagated (or “stick”) to the inferred atoms; e.g., the set of rules \(\{\sigma_1, \sigma_3, \sigma_4\}\). The main goal of stickiness was the definition of a language that allows for joins in rule bodies, which are not always expressible via guarded rules.

Interestingly, the consolidation of the above paradigms leads to more expressive decidable formalisms. Notable examples are *glut-guardedness* (Krötzsch and Rudolph 2011), obtained by combining weak-acyclicity and guardedness, *weak-stickiness* (Cali, Gottlob, and Pieris 2012), obtained by joining weak-acyclicity and stickiness, and *tameness*, obtained by combining guardedness and stickiness (Gottlob, Manna, and Pieris 2013). Another important language is *shy* Datalog± (Leone et al. 2012).

• What about equalities and the falsum (⊥) in rule heads?

Weakly-acyclic Datalog± can be safely combined with arbitrary equality rules, a.k.a. equality-generating dependencies...
Data Complexity

UB: (Calì, Gottlob, and Kifer 2013, Thm. 6.1)
LB: (Dantsin et al. 2001, Thm. 4.4)

UB: implicit in (Fagin et al. 2005)

UB: (Fagin et al. 2005, Cor. 4.3)

UB: (Calì, Gottlob, and Pieris 2012, Thm. 3.3)
UB: (Calì, Gottlob, and Lukasiewicz 2012, Thm. 6)
LB: (Calì, Gottlob, and Pieris 2012, Thm. 3.5)
LB: (Dantsin et al. 2001, Thm. 4.4)

UB: (Fagin et al. 2005, Cor. 4.3)

UB: (Calì, Gottlob, and Pieris 2012, Thm. 6.1)
LB: (Calì, Gottlob, and Pieris 2012, Thm. 3.4)

UB: (Calì, Gottlob, and Pieris 2012, Thm. 3.3)
LB: (Calì, Gottlob, and Pieris 2012, Thm. 3.4)

EXPTIME

EXPTIME-complete

PTIME-complete

PTIME-complete

Table 1: Complexity of query answering; UB and LB stands for upper and lower bound, respectively.

Table 1: Complexity of query answering; UB and LB stands for upper and lower bound, respectively.

Data Complexity

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-complete

PTIME-comple...
(SMS) (Gottlob et al. 2014b). Actually, it is not difficult to define a negation semantics for Datalog\(^\pm\), as is this inherited from its standard counterpart in normal logic programs with function symbols. In fact, existentially quantified variables can be replaced by Skolem terms in rule heads. The tricky part, however, is to show that Datalog\(^\pm\) with negation under the new semantics remains decidable, and pinpoint the exact complexity of query answering.

The WFS comes in two flavors depending on whether the unique name assumption (UNA) is applied to Skolem terms or not. The first approach leads to the “equality-friendly” WFS (Gottlob et al. 2012a), while the second one leads to the standard WFS (Hernich et al. 2013). The SMS (Gelfond and Lifschitz 1988) is another predominating semantics for nonmonotonic normal programs. In (Gottlob et al. 2014b), we have defined and studied the SMS for guarded Datalog\(^\pm\) under the UNA (note that an equality-friendly SMS with UNA also exists). There are cases in which the SMS leads to better query answers than the WFS. For example, given the fact FiveStar(ritz) and the set of rules

\[
\begin{align*}
\text{FiveStar}(X) & \rightarrow \text{Hotel}(X), \\
\text{FiveStar}(X), \neg\text{Pool}(X,Y) & \rightarrow \exists Z \text{ Beach}(X,Z), \\
\text{FiveStar}(X), \neg\text{Beach}(X,Y) & \rightarrow \exists Z \text{ Pool}(X,Z), \\
\text{Beach}(X,Y) & \rightarrow \exists Z \text{ SwimOpp}(X,Z), \\
\text{Pool}(X,Y) & \rightarrow \exists Z \text{ SwimOpp}(X,Z),
\end{align*}
\]

only the atoms FiveStar(ritz) and Hotel(ritz) are entailed under the WFS, while the atom SwimOpp(ritz) is additionally entailed under the SMS (as desired).

As several DLs can be embedded into Datalog\(^\pm\), the decidability results for Datalog\(^\pm\) with negation can also be applied to define decidable extensions of DLs with nonmonotonic negation. We have done this in particular for DL-Lite\(_R\) and DL-Lite\(_R,\sqcap\) (Calvanese et al. 2007), as well as ECHI (Baader, Brandt, and Lutz 2005). The above example corresponds to the following set of DL axioms, expressed in an extension of ECHI by nonmonotonic negation:

\[
\begin{align*}
\text{FiveStar} & \subseteq \text{ Hotel,} \\
\text{FiveStar} \sqcap \neg \exists \text{ Pool} & \subseteq \exists \text{ Beach}, \\
\text{FiveStar} \sqcap \neg \exists \text{ Beach} & \subseteq \exists \text{ Pool}, \\
\exists \text{ Beach} & \subseteq \exists \text{ SwimOpp}, \\
\exists \text{ Pool} & \subseteq \exists \text{ SwimOpp},
\end{align*}
\]

- **Is Datalog\(^\pm\) powerful enough for nondeterministic reasoning? In other words, can the existing languages enriched with disjunction in rule heads?**

  Weakly-acyclic and guarded Datalog\(^\pm\) (and their extensions) can be extended with disjunction, without sacrificing the decidability of query answering. In fact, the impact of disjunction on the complexity of query answering under guarded-based Datalog\(^\pm\) languages has been recently investigated in (Gottlob et al. 2012b; Bourhis, Morak, and Pieris 2013). Unfortunately, the above decidability result does not hold for sticky Datalog\(^\pm\). A recent (still unpublished) result shows that the combination of stickiness with disjunction leads to undecidability. This result gives rise to the question whether a restricted, yet meaningful fragment of sticky Datalog\(^\pm\) can be isolated, which can be safely extended with disjunction. This will be the subject of future research.

- **What other developments have been made related to Datalog\(^\pm\)?**

  1. Query rewriting has been extensively studied. Several algorithms have been designed with the aim of reducing the problem of query answering under Datalog\(^\pm\) languages into evaluation of database queries (Gottlob, Orsi, and Pieris 2011; Orsi and Pieris 2011; Gottlob and Schwentick 2012; Gottlob et al. 2014a; Gottlob, Manna, and Pieris 2014a).

  2. In (Gottlob et al. 2013), a probabilistic extension of Datalog\(^\pm\) that is based on Markov logic networks as underlying probabilistic semantics is presented. It may, e.g., be used in data extraction from the Web.

  3. In (Łukasiewicz, Martinez, and Simari 2012), a general framework for inconsistency management in Datalog\(^\pm\) ontologies based on incision functions from belief revision is developed. It may, e.g., be used for handling inconsistencies in the Semantic Web.

  4. Towards personalized semantic search (e.g., in social networks), in (Łukasiewicz, Martinez, and Simari 2013) an approach to preference-based query answering in ontologies, combining Datalog\(^\pm\) with preference management as in relational databases, is presented.

  5. Query answering under finite models has been also investigated. More precisely, query answering for most of the languages, without egds, mentioned above is finitely controllable. This means that the answers to a query remain exactly the same when, instead of considering arbitrary models, we restrict our attention to finite models. Finite controllability of query answering under (weakly-)guarded Datalog\(^\pm\) was shown in (Bárány, Gottlob, and Otto 2010) (extending the work by (Rosati 2011)), under sticky Datalog\(^\pm\) in (Gogacz and Marcinkowski 2013), and under tame Datalog\(^\pm\) in (Gottlob, Manna, and Pieris 2014b).

  - **Are there any other modeling features that will be added to Datalog\(^\pm\)?**

    We are planning to investigate more expressive equality assertions and transitivity. Let us give some more details:

    1. Though existing languages are powerful enough to express equality assertions, as said, this can only be done as long as the non-conflicting condition is satisfied. However, in real-life examples, new intensional knowledge may be inferred from the equality rules, which implies that the non-conflicting condition is violated. Hence, more general conditions, beyond the non-conflicting one, are needed.

    2. Transitivity is partially captured by some Datalog\(^\pm\) languages, e.g., weakly-sticky Datalog\(^\pm\), as long as the necessary join operations are performed on finitely many values. However, this is not powerful enough for expressing some natural transitivity axioms. We only know that guarded-based Datalog\(^\pm\) languages cannot be safely extended with transitivity since this leads to undecidability (Gottlob, Pieris, and Tendera 2013). Thus, the enrichment of Datalog\(^\pm\) with this feature will be the subject of future substantial research.

  - **Are there any implemented systems for reasoning over Datalog\(^\pm\) languages?**

    We know of three such systems: (1) Nyaya (Virgilio et al. 2012) is a system able to treat the first-order rewritable fragments of Datalog\(^\pm\), that is, linear and sticky Datalog\(^\pm\)
In fact, the given set of rules and query are compiled into an SQL query, which is then evaluated over the extensional database; (2) DLV$^3$ (Leone et al. 2012) implements a bottom-up evaluation strategy for shy Datalog$^\pm$ inside the well-known Answer Set Programming (ASP) system DLV (Leone et al. 2006); and (3) Alaska (König et al. 2012), similarly to Nyaya, is able to treat the first-order rewritable fragments of Datalog$^\pm$, and it is based on SQL-rewritings.

The above are interesting prototypes which show that a complete system, which is able to effectively answer queries over Datalog$^\pm$ ontologies, is realistic. The implementation of such a system will be the subject of future research.

Acknowledgements. Work funded by ERC Grant 246858 (DIADEM), and EPSRC grant EP/J008346/1 (PrOQAW).

References


Leone, N.; Manna, M.; Terracina, G.; and Veltri, P. 2012. Efficiently computable Datalog$^\pm$ programs. In Proc. of KR.

Lukasiewicz, T.; Martinez, M. V.; and Simari, G. I. 2012. Inconsistency handling in Datalog$^\pm$ ontologies. In Proc. of ECAI.


