Turning The Partial-closed World Assumption Upside Down

Simon Razniewski, Ognjen Savković and Werner Nutt
{razniewski, savkovic, nutt}@inf.unibz.it

Free University of Bozen-Bolzano, Italy

Abstract The partial-closed world setting provides an intermediate ground between open-world and closed-world settings. Previous work on the partial-closed world assumption assumes that incompleteness is the default, i.e., that databases in general can be incomplete, and are only definitely complete in specified parts. In this work we turn this assumption around, and study databases where completeness is the default, and incompleteness only occurs in specified parts.
We present four languages for describing potentially incomplete parts of databases, full-table statements, patterns, local statements and query statements. We show that except for full-table statements, it is not possible to translate between the two settings without extending the languages. Finally, we present techniques to decide query completeness entailment for each of the languages both on the schema and instance level, finding that for queries under bag semantics, the complexity is sometimes easier than under the setting where complete parts are specified.

1 Introduction

Databases are traditionally considered under the closed-world assumption [15], while for many knowledge bases and the semantic web, the open-world assumption [10] is used. In many applications however, only parts of databases are complete, while in other parts, data is known to be missing, or potentially missing. One can call this the *partial-closed world assumption* (PCWA). A problem for databases under the PCWA is to decide whether a query returns all possible answers, in which case the query is called complete.

Previous work on query completeness under the PCWA used so-called completeness statements to describe complete parts, while assuming that the undescribed parts of the database can be incomplete [11,8,14,13,4,3]. We call this the *incompleteness-as-default (IAD) setting*.

In this paper, we turn this assumption around and instead assume that databases are only incomplete in explicitly described parts, while all other parts are complete. We call this setting *completeness-as-default (CAD)*. The study of this setting is motivated by the observation that in many business applications, data is generally of good quality, while only specific parts are questionable.

Motivating Example. As an example, consider a university database containing three tables:

- `student(name,degree)`
- `lecturer(name,faculty)`
- `takes(name,course)`.

Assume that for all faculties but Computer Science the course enrolment has already finished, so `takes` is only potentially incomplete for enrolments of CS students. Then clearly, queries for courses of students in the Philosophy degree will still produce complete results, while queries for subjects of students in the Computer Science degree might not.

In order to translate this situation into the IAD setting, we would need a completeness statement for the `takes` table for all entries for students that are not in the CS degree, which requires the use of negation.

Another motivation for incompleteness statements is that they can be naturally translated into tasks. In the research information system of our university, for instance, academics are periodically given tasks to complete information about publications in recent years. These tasks are the same as incompleteness statements for the respective periods. Also, the use of incompleteness statements allows to directly point out why query answers are not complete, which in general, in the incompleteness-as-default setting cannot be done.

Research questions. Motivated by the example above, we study three questions:

1. How can one describe potentially incomplete parts of a database?
2. Which language extensions are needed to translate descriptions from the CAD to the IAD setting?
3. How can one decide whether a query answer is complete?

Contribution. Our contribution is to study the completeness-as-default setting for partially complete databases. In particular we present (1) four different languages for describing potential incompleteness, which are not symmetric to the ones used in the IAD setting, (2) we show that except for a trivial case, it is not possible to translate between the two settings without extending the languages, and (3) we present techniques to decide query completeness entailment for each of the languages both on the schema and instance level.

2 Formalization

We assume a fixed set of relation symbols Σ, and we deal with *conjunctive queries* that are satisfiable and do not contain arithmetic comparisons.

Incompleteness Statement Languages We study four languages for specifying possibly incomplete parts of a database:

1. *Full-table statements*,
2. *Pattern statements* [13],
3. *Local statements*, similar to the local completeness statements in [8,14],

4. *Query statements*, similar to the query completeness statements in [11].

Intuitively, *full-table statements* can be used to assert that a whole table is potentially incomplete. They are a very coarse way to describe possible incompleteness, nevertheless, they are the only of the languages studied here for which it is possible to translate between completeness as default and incompleteness as default without extending the language. *Patterns* were introduced in [13], while the *local statements* closely resemble the local completeness statements introduced in [8], and the *query statements* somewhat the query completeness statements introduced in [11]. We next formalize the statements and their semantics.

Potential Incompleteness Statements.

A *full table statement* has the form $PotInc(R)$, where R is a table.

A *pattern statement* has the form $PotInc(\sigma_p(R))$, where p is a selection by constants, i.e., p is "$\bar{x} = \bar{c}$", with $\bar{x}$ being a subset of the attributes of R, and R being a table.

A *local statement* has the form $PotInc(R(\bar{x}); G)$, where R is a table and G is a conjunction of atoms.

A *query statement* has the form $PotInc(Q)$, where Q is a conjunctive query.

Example 1. Consider a database with three tables, `student`, `lecturer` and `takes`. A full-table statement $PotInc(\texttt{student})$ for `student` would assert that the student table is possibly incomplete.. A pattern statement $PotInc(\sigma_{degree=CS}(\texttt{student}))$ would assert that students in the CS degree are possibly incomplete. A local statement $PotInc(\texttt{student}(n, d); \texttt{takes}(n, \texttt{Databases}))$ would assert that records in `student` of students taking Databases are possibly incomplete. A query statement $PotInc(Q_{\texttt{Databases}}(n) \text{ :- } \texttt{student}(n, d), \texttt{takes}(n, \texttt{Databases}))$ would assert that the query for students taking Databases is potentially incomplete, which can be either due to missing records in `student` or to missing records in `takes`.

We next formalize the models of potential incompleteness statements. As common [14], an *incomplete database* is a pair (D^i, D^a) of an ideal and an available database with $D^a \subseteq D^i$. Given an ideal database D^i and a fact f, we say that a potential incompleteness statement *explains the absence* of f from D^a, if

- Full table statement $PotInc(R)$: f is a fact in relation R.
- Pattern statement $PotInc(\sigma_p(R))$: f is an R-fact that satisfies the selection condition p
- Local statement $PotInc(R(\bar{x}); G)$: f is in $Q_{R;G}(D^i)$, where $Q_{R;G}$ is the query $Q_{R;G}(\bar{x}) \text{ :- } R(\bar{x}), G$.
- Query statement $PotInc(Q)$: There exists a valuation for Q over D^i that maps some atom in the body of Q to f.

Intuitively, models of a set of incompleteness statements are those incomplete databases where incompleteness occurs only in parts described by incompleteness statements.

Definition 1. *Given a set of incompleteness statements $\mathfrak{S}$, models are all incomplete database (D^i, D^a) where the absence of each fact f in $D^i \setminus D^a$ is explained by some statement in $\mathfrak{S}$.*

Example 2. Consider the statement $PotInc(\texttt{student}(n, c); \texttt{takes}(n, \texttt{Databases}))$ from Ex. 1, which says that records in `student` of students taking Databases are possibly incomplete. Models of this statement are all incomplete databases (D^i, D^a) where the only difference between D^i and D^a are `student` facts for students that take Databases according to D^i.

Unlike for instance in propositional logic, models of potential incompleteness statements are *monotonic*. The incomplete database (D^i, D^a) with $D^i = \{\texttt{student}(John, CS), \texttt{lecturer}(Mary, ST)\}$ and $D^a = \emptyset$ is neither a model of $PotInc(\texttt{student})$ nor of $PotInc(\texttt{lecturer})$ alone, but of the two together. In general, supersets of potential incompleteness statements have more models than their subsets. We find the following order of expressivity of languages for potential incompleteness:

Proposition 1 (Expressiveness of Languages).

1. *Local statements subsume query statements.*
2. *Query statements subsume pattern statements.*
3. *Pattern statements subsume full table statements.*

Proof. Subsumption (2) holds because any pattern statement $PotInc(\sigma_p(R))$ can be expressed using a query statement $PotInc(Q() :\text{-} \sigma R)$. Similarly, (3) holds as any potential incompleteness of a table can be expressed equivalently with a pattern stating that any part of the table is potentially incomplete. Regarding (1), observe that given a query $Q :\text{-} R_1, \ldots, R_n$ that is potentially incomplete, one can equivalently formulate n local statements which assert $PotInc(R_i; R_1, \ldots, R_{i-1}, R_{i+1}, \ldots, R_n)$.

An illustration is shown in Fig. 1.

Query Completeness Under the OWA, conjunctive queries only return certain answers, while under the CWA, the returned answers are also all possible answers. Under the PCWA, one can investigate whether a query returns all possible answers, which is a property called *query completeness*. Formally, an incomplete database satisfies query completeness under bag (set) semantics, if the bag (set) of answers to the query over the ideal database is the same as over the available database. We then write $Compl(Q)$. Similarly, a set of incomplete databases satisfies query completeness, if each member satisfies query completeness. Note that query completeness inferences are *nonmonotonic* wrt. potential incompleteness statements: More potential incompleteness statements imply that fewer queries can be inferred to be complete.

In the rest of the paper, we study three problems.

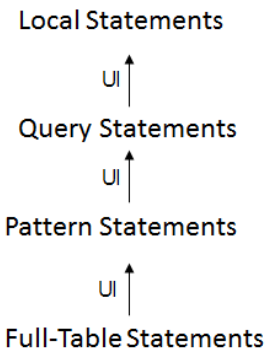

Figure 1. Entailment of potential incompleteness statement languages.

Problem 1 (Translation). Given a language of potential incompleteness statements, which additions are needed to translate any set of statements from the CAD to the IAD setting?

Problem 2 (Schema Reasoning). Given a set $\mathfrak{S}$ of statements and a query Q, is Q complete over all models of $\mathfrak{S}$, i.e., $\mathfrak{S} \models \mathit{Compl}(Q)$?

Problem 3 (Instance Reasoning). Given statements $\mathfrak{S}$, a database D^a and a query Q, is Q complete over all models of $\mathfrak{S}$ where D^a is fixed, i.e., $\mathfrak{S} \models_{D^a} \mathit{Compl}(Q)$?

3 Translation

In this section we study how to translate from the CAD to the IAD setting.

Example 3. Recall the full-table statement $\mathit{PotInc}(\texttt{student})$ from Ex. 1. Given that the database has only three tables, this can be translated into the IAD setting by stating that the tables `lecturer` and `takes` are complete, i.e., $\mathit{Compl}(\texttt{lecturer})$ and $\mathit{Compl}(\texttt{takes})$. In the motivating example we already saw that negation may be needed.

We next briefly formalize the semantics of completeness statements. An incomplete database (D^i, D^a) satisfies a completeness statement, if:

- Full table statement $\mathit{PotInc}(R)$: $R(D^i) = R(D^a)$.
- Pattern statement $\mathit{PotInc}(\sigma_p(R))$ [13]: $\sigma_p(R(D^i) \subseteq R(D^a)$
- Local statement $\mathit{PotInc}(R(\bar{x}); G)$ [8,14]: The result of $Q_{R;G}(\bar{x})$:- $R(\bar{x}), G$ over D^i is contained in $R(D^a)$.
- Query statement $\mathit{PotInc}(Q)$ [11]: $Q(D^i) = Q(D^a)$.

Unlike potential incompleteness statements, completeness statements are monotonic, i.e., an incomplete database satisfies a set of completeness statements, iff it satisfies each of them. We next show how translation between CAD and IAD can work in principle.

The next proposition shows that unless for full table statements, translation from the CAD to the IAD setting is not possible without additional assumptions.

Proposition 2 (Translatability from CAD to IAD).

1. *For full table statements, translation is possible.*
2. *For pattern statements, translation is possible if attribute domains are finite, or disequality is added to the pattern statements.*
3. *For local and query statements, translation is possible if (1) the domains of all attributes instantiated in the pattern statement are finite or disequality is added, and (2) if negation is added.*

Proof. (1): Translation is possible by creating a completeness statement for every relation in $\Sigma \setminus \{R \mid \mathit{PotInc}(R) \in \mathfrak{S}\}$.

(2): A translation is possible if instantiated domains are finite, e.g. if gender is either male or female, then $\mathit{PotInc}(\sigma_{\mathit{gender}=\mathit{male}}(\mathit{person}))$ is analogue to $\mathit{Compl}(\sigma_{\mathit{gender}=\mathit{female}}(\mathit{person}))$, otherwise, we would need an infinite set of statements. If disequality is allowed, one could assert $\mathit{Compl}(\sigma_{\mathit{gender}\neq\mathit{male}}(\mathit{person}))$.

(3): Additionally to finite domains or disequality, negations is needed, as shown in the introductory example.

4 Completeness Entailment for Queries under Bag Semantics

In the following, we develop techniques for deciding query completeness entailment for queries under bag semantics. Somewhat surprisingly, for all four languages, schema reasoning has the same complexity (PTIME), and instance reasoning has the same complexity (coNP), which is generally easier than in the IAD setting.

Let us see first how to reason with full-table and pattern statements:

Example 4 (Full-table and Pattern Statements). Consider the following query $Q_{\texttt{Databases}}(n)$:- $\texttt{student}(n, c)$, $\texttt{takes}(n, \texttt{Databases})$, and a potential incompleteness statement for the `takes` table, that is, $\mathit{PotInc}(\texttt{takes})$. Then, this statement does not entail query completeness, because missing records in `takes` could lead to an incomplete query result. Alternatively, assume a potential incompleteness statement $\mathit{PotInc}(\sigma_{\mathit{course}=\mathit{Algorithms}}(\texttt{takes}))$. Clearly, this statement has no influence on the query $Q_{\texttt{Databases}}$ that asks for students that take Databases, thus, query completeness is entailed.

For local statements, we find that, unlike in the IAD setting, even statements that contain relations not appearing in the query can influence query completeness.

Example 5 (Local Statements). Consider an incompleteness statement for `takes` for remote students, that is, $\mathit{PotInc}(\texttt{takes}(n, d); \texttt{remoteStudent}(n))$. This can lead to a potentially incomplete query result for $Q_{\texttt{Databases}}$, because remote students might take Databases, thus, no completeness holds.

As query statements are just collections of atoms that might be incomplete, they behave analogously to pattern statements.

Example 6 (Query Statements). Consider an incompleteness statement for a query for all lecturers that are also students, $\mathit{PotInc}(Q(n)$:- $\texttt{student}(n, d), \texttt{lecturer}(n, f))$. Clearly, such students could also take Databases and thus, if missing, could lead to an incomplete query result for $Q_{\texttt{Databases}}$, thus, no query completeness holds.

Formally, we find the following:

Theorem 1 (Schema Reasoning for Bag Semantics). *For all languages, the combined complexity of schema reasoning is PTIME.*

Proof. We only discuss local statements, as they entail all other statements. For local statements of the form $PotInc(R_i(\bar{x}_i), G_i)$, completeness holds if and only if none of the $R_i(\bar{x}_i)$ atoms can be unified with any atom in Q.

Wrt. the database instance, in general, more conclusions are possible.

Example 7 (Instance Reasoning). Consider again the situation from Ex. 4, where $PotInc(\texttt{takes})$ holds, but assume that we additionally know that `student` is empty. Then, no matter whether records are missing in `takes`, the result of $Q_{\texttt{Databases}}$ will always be empty and hence complete.

Theorem 2 (Instance Reasoning for Bag Semantics). *For all languages, instance reasoning is coNP-hard in combined complexity. For full table and pattern statements, the combined complexity is also coNP, and the data complexity is PTIME.*

Proof. (coNP-membership): To show that completeness does not hold, it suffices to guess an instantiation θ for the body B of Q such that at least one fact in θB is not in D^a, and such that $(D^a, D^a \cup \theta B)$ satisfies $\mathfrak{S}$. As the latter part can be done in PTIME, the entailment checking is in coNP.

(PTIME data complexity): Observe that for full-table and pattern statements, the guesses are especially easy, as it suffices to consider the constants in the database plus at most as many new ones as there are variables in the query, thus, there are polynomial many possible valuations to consider. For local and query statements, one also has to show that the new facts are explained by potential incompleteness statements, which may increase the complexity.

(coNP-hardness): We can reduce graph colorability to instance reasoning with full table statements as follows: Consider a graph G. We construct a Boolean query $Q() \text{:-} R(), G'$, where G' contains for each edge from node x to node y in G an atom $E(x, y)$ (where x and y are variables). Furthermore, let $D^a = \{E(r, g), E(r, b), E(g, r), E(g, b), E(b, r), E(b, g)\}$, that is, the database contains the six allowed colorings for adjacent vertices. Finally, let $\mathfrak{S} = \{PotInc(R)\}$. Then, $\mathfrak{S}$ and D^a entail $Compl(Q)$ if and only if G is not three-colorable.. Assume G is not colorable. Then there is no homomorphism from G' into the (complete) set of allowed colorings in any D^i, and hence, Q would return the false also over any D^i and hence Q would be complete. On the other hand, if G is three-colorable, a homomorphism from G' to D^a exists and hence we can construct a $D^i = D^a \cup R()$ where Q evaluates to true.

For local and query statements it is not yet known how to decide entailment, because new facts in θB might require justifications that use again new facts, which require again justifications, and so on, similarly to chase-procedures. All results for queries are summarized in Table 2, where we also list known complexities from the IAD setting.

5 Completeness Entailment for Queries under Set Semantics

Queries can be evaluated under bag or set semantics, and it is possible that a query is complete under set, but not under bag semantics.

	Bag semantics				Set semantics				Translation
	IAD		CAD		IAD		CAD		
	Schema Reasoning	Instance Reasoning	Schema Reasoning	Instance Reasoning	Schema Reasoning	Instance Reasoning	Schema Reasoning	Instance Reasoning	
Full-table statements	PTIME*	coNP-complete*	PTIME	coNP-complete	PTIME*	Π_2^p-complete*	PTIME	Π_2^p-complete	Possible
Pattern statments	PTIME*	coNP-complete*	PTIME	coNP-complete	NP-complete*	Π_2^p-complete*	coNP-hard, in Π_2^p	Π_2^p-complete	With finite domains or disequality
Local statements	NP-complete [14]	in Π_2^p*	PTIME	coNP-hard	NP-complete [14]	Π_2^p-complete [14]	coNP-hard	Π_2^p-hard	With finite domains or disequality, and negation
Query statements	NP-complete [14]	Π_2^p-complete [6]	PTIME	coNP-hard	Decidability open	Π_2^p-complete [6]	coNP-hard	Π_2^p-hard	With finite domains or disequality, and negation

Table 2. Combined complexity of completeness entailment under IAD and CAD, and requirements for translation. Straightforward new results for the IAD setting are marked with * and are not explained in the paper.

Example 8. Consider the query $Q(n)$:- $\texttt{takes}(n, \texttt{Databases})$, $\texttt{takes}(n, x)$ that, under bag semantics, asks for the number of courses that students taking databases take. Consider also the incompleteness statement $\mathit{PotInc}(\sigma_{\texttt{course=OS}}(\texttt{takes}))$. Then the above query might not return the correct count (*bag*), but it would still return the correct *set* of all students that take databases and something else.

For queries under set semantics, entailment is generally more difficult.

Theorem 3 (Schema Reasoning for Set Semantics). *Schema reasoning for full table statements in PTIME, for pattern statements in Π_2^p and coNP-hard for pattern, local and query statements in combined complexity.*

Proof. Membership for full table statements holds as one can just check whether any relation used in the query is potentially incomplete.

CoNP-hardness for pattern statements can be shown by reduction of 3-colorability. Consider a graph G encoded into a query Q as before. Consider furthermore potential incompleteness statements $\mathit{PotInc}(E(r, g))$, $\mathit{PotInc}(E(r, b))$ and so on (the six allowed colorings). Then Q is complete exactly if G is not colorable.

Π_2^p-membership can be shown by the same technique as used in the proof of coNP-membership in Thm. 2. Now however it needs to be shown that the newly retrieved fact is not already returned over D^a, thus, after the first guess, a call to an NP-oracle is needed.

Instance reasoning is hard even for full-table statements.

Theorem 4 (Instance Reasoning for Set Semantics). *Instance reasoning for all languages is Π_2^p-hard in combined complexity, and in Π_2^p for full-table and pattern statements.*

Proof. (Hardness): Can be shown by reduction of *3-coloring extension* problem [1].

(Membership): Can be shown using the same technique as in the proof of Thm. 2.

Interestingly, even reasoning for Boolean queries provides a challenge.

Proposition 3 (Boolean Queries). *Instance reasoning with full table statements for boolean queries is co-DP-complete.*

Proof. We show that incompleteness is DP-complete. We show hardness by encoding *critical 3-colorability* problem which is DP-complete [12]. To show membership we check incompleteness by calling two NP oracles: one that checks if a boolean query evaluates to *false* over a given database, and the other that checks if the query evaluates to *true* over a representative ideal database.

All hardness results hold also for linear queries (because the complexity comes from the choices for the mapping), as one could reduce satisfiability of quantified boolean formulas (QBF). So all complexities above hold even with fixed database. Also, if the number of variables in the query is bounded, entailment becomes polynomial, as there are only polynomially many possible mappings.

6 Related Work

The problem of query completeness entailment in the IAD setting has been introduced by Motro [11] and Halevy [8]. Most complexity results for this setting appeared in [14].

Query completeness in the CAD setting is closely related to the problem of *queries independent of updates* (QIU). QIU investigates whether a query result can change as a consequence of a database update. Intuitively, query completeness entailment holds, if the given query is independent from all insertion operations corresponding to the incompleteness statements.

QIU has been introduced by Blakeley et al. [2], who investigated QIU for conjunctive queries without selfjoins and deletion, insertion and modification updates that correspond to patterns. Both queries and statements were allowed to contain comparisons. They showed that deciding QIU wrt. a database instance in these cases is in coNP for selfjoin-free queries that may contain comparisons. Elkan [5] and Halevy and Sagiv [9] expanded this work to recursive queries and queries containing negation. They provided sufficient conditions for QIU for such cases in terms of query reachability, and showed that in special cases these conditions were also necessary. Recent work studies QIU also for the SPARQL language [7].

7 Discussion

Implementation Techniques Schema reasoning is straightforward for all four languages, each time requiring to check for unifiability between single atoms. Implementing instance reasoning for full table and pattern statements is possible as follows: Consider that the query has the form $Q() \text{:-} A_1, \ldots, A_n$, where atoms A_1 to A_m are complete, and atoms A_{m+1} to A_n are potentially incomplete. If some atom in A_{m+1} to A_n contains a variable that does not appear in A_1 to A_m, it suffices to check whether the A_1 to A_m can be evaluated over D^a. If none of the potentially incomplete atoms has a new variable, one can proceed as follows: For each atom $A_i(\bar{s}_i)$, $i = m+1 \ldots n$ one can build a query $Q(\bar{s}_i) \text{:-} A_1, \ldots, A_m, \neg A_i$. If this query returns any answer over D^a, this means that the query is not complete. To deal with pattern statements, one would add the condition to the query. Thus, completeness reasoning can be implemented using query evaluation.

Combining CAD and IAD One can imagine situations where some relations are interpreted under IAD and some under CAD. An example could be internal research evaluation by publication analysis, where publications of the members of an institute are in a local information system and thus complete by default, while publications of coauthors are taken from the web and thus generally incomplete. To reason about query completeness in such settings, one could proceed in three steps: (1) one would assert completeness for all relations under CAD assumption, (2) one would perform completeness reasoning using all completeness statements, and (3) one would perform completeness reasoning using only the potential incompleteness statements. Consequently, the complexity would be the union of the complexities of reasoning under CAD and IAD.

Incompleteness statements Instead of potential incompleteness statements, one can also image statements that state that a certain part of a database is *definitely* incomplete, for instance, that the `student` table is definitely missing some records. For the nonspecified parts, one may either assume completeness or potential incompleteness. In both cases, one could additionally check whether a query is known to be definitely incomplete, which would be a different reasoning problem. However, this inference is not very practical, as it is hardly ever possible. For instance, consider a query $Q(x) \text{:-} R(x), S(x)$, and consider that R and S are known to be definitely incomplete (miss some facts). Then this does not allow to conclude that Q is incomplete, because sets of facts missing from R and S might be disjoint. More generally, for pattern statements, inferring definite query incompleteness seems only possible in trivial cases where the query has exactly the same shape as a single definite incompleteness statement.

Future work will focus filling the gaps in the complexities of completeness entailment, and on the connections to data exchange.

Acknowledgment

This work has been partially supported by the projects "MAGIC", funded by the province of Bozen-Bolzano, and "TaDaQua", funded by the Free University of Bozen-Bolzano.

References

1. Miklós Ajtai, Ronald Fagin, and Larry J. Stockmeyer. The closure of monadic NP. *J. Comput. Syst. Sci.*, 60(3):660–716, 2000.
2. J. A. Blakeley, N. Coburn, and P.-A. Larson. Updating derived relations: Detecting irrelevant and autonomously computable updates. In *VLDB*, pages 457–466, 1986.
3. M. Denecker, A. Cortés-Calabuig, M. Bruynooghe, and O. Arieli. Towards a logical reconstruction of a theory for locally closed databases. *ACM TODS*, 35(3), 2010.
4. P. Doherty, W. Lukaszewicz, and A. Szalas. Efficient reasoning using the local closed-world assumption. In *AIMSA*, pages 49–58, 2000.
5. Ch. Elkan. Independence of logic database queries and updates. In *Proc. PODS*, pages 154–160, 1990.
6. R. E. Prasojo W. Nutt F. Darari, S. Razniewski. Enabling fine-grained RDF data completeness assessment. ICWE 2015.
7. N. Guido, P. Genevès, N. Layaïda, and C. Roisin. On query-update independence for SPARQL. CIKM '15, pages 1675–1678, New York, NY, USA, 2015. ACM.
8. Alon Y. Levy. Obtaining complete answers from incomplete databases. In *VLDB*, pages 402–412, 1996.
9. Alon Y. Levy and Yehoshua Sagiv. Queries independent of updates. In *Proc. VLDB*, pages 171–181, 1993.
10. J. Minker. On indefinite databases and the closed world assumption. In *Conference on Automated Deduction*, pages 292–308. Springer, 1982.
11. A. Motro. Integrity = Validity + Completeness. *ACM TODS*, 14(4):480–502, 1989.
12. Christos M. Papadimitriou. *Computational complexity*. Addison-Wesley, 1994.
13. S. Razniewski, F. Korn, W. Nutt, and D. Srivastava. Identifying the extent of completeness of query answers over partially complete databases. In *SIGMOD*, 2015.
14. S. Razniewski and W. Nutt. Completeness of queries over incomplete databases. In *VLDB*, 2011.
15. Raymond Reiter. On closed world data bases. In *Logic and Data Bases*, pages 55–76, 1977.