LACE: A Logical Approach to Collective Entity Resolution

Meghyn Bienvenu  
Univ. Bordeaux, CNRS, Bordeaux INP, LabRI, UMR 5800  
Talence, France  
meghyn.bienvenu@cnrs.fr

Gianluca Cima  
Univ. Bordeaux, CNRS, Bordeaux INP, LabRI, UMR 5800  
Talence, France  
gianluca.cima@u-bordeaux.fr

Victor Gutiérrez-Basulto  
School of Computer Science & Informatics, Cardiff University  
Cardiff, United Kingdom  
gutierrezbasultov@cardiff.ac.uk

ABSTRACT

In this paper, we revisit the problem of entity resolution and propose a novel, logical framework, LACE, which mixes declarative and procedural elements to achieve a number of desirable properties. Our approach is fundamentally declarative in nature: it utilizes hard and soft rules to specify conditions under which pairs of entity references must or may be merged, together with denial constraints that enforce consistency of the resulting instance. Importantly, however, rule bodies are evaluated on the instance resulting from applying the already ‘derived’ merges. It is the dynamic nature of our semantics that enables us to capture collective entity resolution scenarios, where merges can trigger further merges, while at the same time ensuring that every merge can be justified. As the denial constraints restrict which merges can be performed together, we obtain a space of (maximal) solutions, from which we can naturally define notions of certain and possible merges and query answers. We explore the computational properties of our framework and determine the precise computational complexity of the relevant decision problems. Furthermore, as a first step towards implementing our approach, we demonstrate how we can encode the various reasoning tasks using answer set programming.

CCS CONCEPTS

• Information systems → Entity resolution; • Theory of computation → Logic and databases; Constraint and logic programming; Problems, reductions and completeness.

KEYWORDS

Collective Entity Resolution, Declarative Framework, Logical Constraints, Complexity Analysis, Answer Set Programming

ACM Reference Format:


1 INTRODUCTION

Entity resolution (ER), which aims to identify different references to the same real-world entity, is one of the most fundamental problems in data quality management. In the context of relational databases, ER traditionally focused on matching records based on similarity of their fields [35], which is why ER sometimes goes by the name of record linkage [22]. For instance, in a bibliographical database, we might match author records if the email addresses are similar. However, ER can be more generally defined as the following problem: given a database \( D \) determine, for each pair \((c_1, c_2)\) of constants (of the same type) occurring in \( D \), whether they represent the same real-world entity, and can thus be merged [38]. This more general formulation naturally leads to considering the joint resolution of references of multiple entity types (so-called collective ER) and exploiting other types of evidence, such as the way entity references are related. For example, we may also resolve author references by looking at the papers they have written, and likewise, use the authors to determine if two references denote the same paper. The advantages of collective approaches to ER, especially those exploiting relational similarities, have been observed in several works [18, 27, 30]. Being both an important and multi-faceted problem, ER has been tackled using various approaches with different formal foundations: probabilistic approaches [9, 38], deep learning approaches [23, 31, 36] and logical approaches [1, 12, 19, 21]; see [16] for a recent survey.

In this paper, we propose LACE, a logical framework for ER that was designed to satisfy three main desiderata, namely, being collective, declarative, and justifiable. More precisely, our approach (i) supports complex interdependencies between merges of different entities, (ii) adopts a declarative language with logical rules and constraints, and (iii) is able to justify why two constants have been deemed to represent the same entity. While the declarative and collective aspects have received considerable attention in the literature, justifiability is less explored, despite being a key step towards developing more advanced explanation facilities, and thus responsible technologies [31].

In a nutshell, LACE is a declarative language that, inspired by the Dedupalog approach [1], employs hard and soft rules to specify conditions under which pairs of entity references must or may be merged. For example, the constraints every paper has a single corresponding author and conferences with similar names are likely to be the same can be captured using hard and soft rules, respectively. In addition to rules, LACE specifications may include denial constraints [6] to enforce consistency of the resulting instance and constrain the allowed combinations of merges. For example, we may require that the name of an author can appear in the list of author names a single time. We equip LACE with a ‘dynamic’ and ‘global’ semantics. In line with approaches to ER based on matching
dependencies (MDs) [8, 19, 21] and extensions thereof, such as relational MDs [3, 5]. LACE adopts a dynamic semantics in which rule bodies are evaluated on induced instances resulting from applying the already ‘derived’ merges. It is thanks to the dynamic nature of the semantics that we obtain a collective yet justifiable framework, in which merges can trigger further merges, possibly in a recursive fashion, while still being able to trace back the origins of each merge. We describe our semantics as ‘global’ since LACE globally merges constants by replacing one constant with the other everywhere in the database. This choice is motivated by the fact that our focus is on identifying pairs of constants that are entity references (e.g., authors, publications), rather than on merging attribute values (e.g., titles, addresses), for which a local semantics is typically more appropriate as the context in which a value occurs is crucial. In this respect, our work departs from MDs (which adopt a local semantics) and is more in line with Dedupalog and an elegant declarative framework for entity linking proposed by Burdick et al. [12], both of which implicitly work with a global semantics.

Taking further inspiration from the entity linking approach [12] (we henceforth refer to it as EL), we consider not only a single solution, but rather a space of maximal (w.r.t. set containment) solutions. In LACE, this space naturally emerges from the fact that denial constraints restrict which merges can be performed together, effectively creating choices. Also in line with EL, we can naturally define the notions of certain and possible merges, as those merges that belong to all, resp. some, maximal solutions.

While LACE shares common features with the Dedupalog, MDs, and the EL framework, we shall argue that none of these frameworks fully satisfies our three desiderata, nor can be easily adapted (e.g., titles, addresses), for which a local semantics is typically more appropriate as the context in which a value occurs is crucial. In this respect, our work departs from MDs (which adopt a local semantics) and is more in line with Dedupalog and EL frameworks makes it difficult to support collective, recursive scenarios while at the same ensuring all merges are properly justified.

Our paper focuses on the use of a dynamic, global semantics for merging entity references, which is a main novelty of LACE. However, we should emphasize that in practice many ER scenarios will naturally involve merging both entity references and values. Due to our dynamic semantics, we are confident that LACE can be suitably extended to handle local merges as in the MD approach.

Outline of main results. Our first contribution is LACE, a novel Logical Approach to Collective Entity resolution. We present its syntax and semantics and illustrate it via a running example.

With the framework in place, we present our second contribution, a comprehensive study of the data complexity of the relevant computational tasks. Our results can be summarized as follows:

1. The problems of existence and recognition of maximal solutions are intractable (complete for NP and coNP, respectively), but recognition of arbitrary solutions is tractable (P-complete).
2. The problem of recognizing certain merges lies at the second level of the polynomial hierarchy (P^P-complete), while the dual problem of identifying possible merges is NP-complete.
3. We also define certain and possible query answers (w.r.t. the set of maximal solutions) and show that the associated decision problems have the same complexity as the problems in Point 2.
4. We investigate the impact of imposing syntactic restrictions. We show that all of the hardness results hold if denial constraints consist solely of functional dependencies. By contrast, if one considers denial constraints without inequalities, several of the problems decrease in complexity. More drastic restrictions ensure tractability of all considered problems.

Towards the development of an ER system based on LACE, our third contribution is an encoding of solutions as stable models of logic programs, which we use to show how the various tasks can be handled using answer set programming (ASP) systems. The suitability of ASP techniques for implementing data quality approaches has been demonstrated in the ER context for (relational) MDs [3, 4], following earlier ASP encodings of consistent query answering [2].

As a final contribution, we explore the differences in the semantics of EL and LACE and their capability to capture recursive ER scenarios. In particular, we exhibit one such scenario that is easily captured in LACE, but is provably not expressible in EL.

Organization. Necessary background is provided in Section 2. In Section 3, we present the basics of LACE, and in Section 4 we introduce the central decision problems and investigate their computational complexity. In Section 5 we show how to encode these reasoning tasks using answer set programming. In Section 6 we position our approach w.r.t. the logical frameworks mentioned earlier. We conclude in Section 7 with some perspectives for future work. The appendix contains some sketches for omitted proofs.

2 PRELIMINARIES

A (relational) schema is a finite set \( S \) of relation symbols, with each \( R \in S \) having an associated arity and list of attributes. We use conventional notation \( R/k \) and \( R(A_1, \ldots, A_k) \) to indicate respectively that \( R \) has arity \( k \) and that its attributes are \( (A_1, \ldots, A_k) \). A database instance over schema \( S \) (or \( (S-) \)database for short) assigns to each \( k \)-ary relation symbol \( R \in S \) a finite \( k \)-ary relation over a fixed, infinite set of constants. Equivalently, we can view an \( S \)-database \( D \) as a finite set of facts of the form \( R(c_1, \ldots, c_k) \), where \( (c_1, \ldots, c_k) \) is a tuple of constants of the same arity as \( R \). In particular, we will use the notations \( R(c_1, \ldots, c_k) \in D \) and \( D \subseteq D' \), with the obvious meanings. The active domain of a database \( D \), denoted \( \text{dom}(D) \), is the set of all constants occurring in \( D \).

When we speak of queries in this paper, we mean a conjunctive query (CQ). Recall that a CQ over a schema \( S \) takes the form \( q(\bar{x}) = \exists \bar{y}. \phi(\bar{x}, \bar{y}) \), where \( \bar{x} \) and \( \bar{y} \) are disjoint lists of variables, and \( \phi \) is a conjunction of relational atoms of the form \( R(t_1, \ldots, t_k) \), with \( R \in S \) a \( k \)-ary relation symbol and each \( t_i \) is a constant or variable from \( \bar{x} \cup \bar{y} \). The arity of a CQ \( q(\bar{x}) \) is its number of distinguished variables \( \bar{x} \), and a query with arity 0 is called Boolean. Given an \( n \)-ary query \( q(x_1, \ldots, x_n) \) and \( n \)-tuple of constants \( \bar{c} = (c_1, \ldots, c_n) \), we denote by \( q[\bar{c}] \) the Boolean query obtained by replacing each \( x_i \) by \( c_i \). The answers to an \( n \)-ary CQ \( q(\bar{x}) \) over a database \( D \) is defined as usual as the set of \( n \)-tuples of constants \( \bar{c} \) from \( D \) such that \( q[\bar{c}] \) holds in \( D \). We use \( q(D) \) to denote the answers to \( q \) over \( D \).

When formulating entity resolution rules, we will consider CQs that may also contain atoms built from a set of externally defined binary similarity predicates. The preceding definitions and notions extend to such CQs, the only difference being that similarity predicates have a fixed meaning (typically defined by applying a similarity metric, e.g. edit distance, and keeping those pairs of values whose score exceeds a given threshold).
Our framework will also utilize denial constraints [6, 20] of the form \( \forall x \neg(\varphi(x)) \), where \( \varphi(x) \) is a finite conjunction of atoms, which are either relational atoms (over the considered schema) or inequality atoms \( t_1 \neq t_2 \), with variables drawn from \( x \). Denial constraints notably generalize the well-known class of functional dependencies (FDs). For example, the FD \( R : \{ A, B \} \rightarrow C \), for relation \( R(A, B, C) \), is captured by the denial constraint \( \forall x, y, z, z' (\neg(R(x, y, z) \land R(x, y, z') \land z \neq z')) \).

3 FRAMEWORK FOR ENTITY RESOLUTION

In this section, we introduce LACE, a Logical Approach to Collective Entity resolution that satisfies the desiderata laid out in Section 1.

Recall that the general aim is to identify pairs of database constants that refer to the same entity. We will use the term merge to speak about such pairs. The LACE framework employs hard and soft rules to indicate (required or potential) merges. A hard rule (w.r.t. schema \( S \)) takes the form

\[
q(x, y) \Rightarrow EQ(x, y),
\]

where \( q(x, y) \) is a CQ, whose atoms may use relation symbols in \( S \) as well as similarity predicates, and \( EQ \) is a special relation symbol (not in \( S \)) used to store merges. Intuitively, such a rule states that \( (c_1, c_2) \) being an answer to \( q \) provides sufficient conditions for concluding that \( c_1 \) and \( c_2 \) refer to the same entity. Soft rules have a similar form

\[
q(x, y) \rightarrow EQ(x, y),
\]

but state instead that \( (c_1, c_2) \) being an answer to \( q \) provides reasonable evidence for \( c_1 \) and \( c_2 \) denoting the same entity. Such rules suggest potential (but not mandatory) merges of constants. We use the notation \( q(x, y) \rightarrow EQ(x, y) \) for a generic (hard or soft) rule, and for the sake of brevity, we will sometimes omit the existential quantifiers of the variables appearing uniquely in the rule body.

Example 1. Figure 1 introduces the schema \( S_{ex} \) and ruleset \( \Gamma_{ex} \) of our running example. Hard rule \( \rho_2 \) states that if two conferences are held in the year, have the same chair, and have similar names, then they are actually the same conference. Soft rule \( \delta_2 \) states that two author ids likely refer to the same person if the affiliations match and the emails are similar.

We shall assume that rules are sensibly written, i.e. the rules only generate merges between pairs of constants with compatible entity types (e.g. person), and similarity atoms involve values of the required datatype (e.g. string). These requirements could be made formal by introducing types for attributes, constants, and similarity predicates, but we omit the details, as they have no impact on the technical development. The only syntactic restriction we place on rulesets is to forbid an attribute from participating both in merges and in similarity atoms. Formally, we call an attribute \( A_i \) of \( R(A_1, \ldots, A_k) \) a merge attribute of ruleset \( \Gamma \) if there exists a rule \( q(x, y) \rightarrow EQ(x, y) \in \Gamma \) and body atom \( R(t_1, \ldots, t_k) \) such that \( t_i \in \{ x, y \} \); we call \( A_i \) a sim attribute of \( \Gamma \) if there is a rule \( q(x, y) \rightarrow EQ(x, y) \in \Gamma \) and variable \( v \) that occurs both in position \( A_i \) of an \( R \)-atom of \( q \) and in a similarity atom of \( q \). We call \( \Gamma \) sim-safe if there is no attribute that is both a merge and sim attribute of \( \Gamma \).

Example 2. The sim attributes of \( \Gamma_{ex} \) are email, title, name, while the merge attributes are those with attribute names: id, pID, aID, cID. As there is no attribute in common, \( \Gamma_{ex} \) is sim-safe.

We can now formally define specifications, which consists of hard and soft rules, together with a set of denial constraints. The latter serve to define what counts as a legal (or consistent) database and can help to block incorrect identification of pairs of constants.

**Definition 1.** An ER specification over a schema \( S \) takes the form \( \Sigma = (\Gamma, \Delta) \), where \( \Gamma = \Gamma_h \cup \Gamma_s \) is a finite sim-safe set of hard and soft rules over \( S \), and \( \Delta \) is a finite set of denial constraints over \( S \).

Example 3. Our running example utilizes the ER specification \( \Sigma_{ex} \) from Figure 1. In addition to the ruleset \( \Gamma_{ex} \), it contains three denial constraints: \( \delta_1 \) and \( \delta_2 \) are FDs for Wrote, while \( \delta_3 \) states that the chair of a conference cannot co-author a paper at the same conference.

Each ER specification and database will give rise to a set of solutions, each corresponding to a set of merges that is coherent w.r.t. the specification. Intuitively, these are EQ-databases that are obtained by ‘deriving’ new EQ-facts via rule applications and closure operations, the latter serving to ensure that the resulting set of pairs is an equivalence relation. Importantly, rule bodies are evaluated on the database induced by the already derived merges, which makes it possible for new rules to become applicable that were not applicable in the original database. Satisfaction of the hard rules and denial constraints is also defined w.r.t. the induced database.

To simplify the presentation, we will define solutions directly as equivalence relations\(^1\) (rather than as EQ-databases). Given a set \( S \) of pairs of constants from \( D \), we denote by \( EQRel(S, D) \), the least equivalence relation \( E \supseteq S \) over \( dom(D) \). We assume that each equivalence relation \( E \) is equipped with a function \( rep_E \) that maps each element to a representative of its equivalence class in \( E \).

Given a database \( D \) and an equivalence relation \( E \) over \( dom(D) \), the database induced by \( D \) and \( E \), denoted \( D_E \), is the database obtained from \( D \) by replacing each constant \( c \) by \( rep_E(c) \). We then define the set \( q(D, E) \) of answers to a query \( q(x) \) w.r.t. \( D \) and \( E \) as follows:

\[
(c_1, \ldots, c_n) \in q(D, E) \text{ iff } (rep_E(c_1), \ldots, rep_E(c_n)) \in q(D_E).
\]

The sim-safe condition ensures that similarity predicates in rule bodies are handled correctly, e.g. if \( rep_E(d_1) \approx rep_E(d_2) \) is used to satisfy a rule body in \( D_E \), then \( rep_E(d_1) = d_1 \) (i.e. \( d_1 = d_2 \)).

A set of denial constraints \( \Delta \) is satisfied in \( (D, E) \), written \( (D, E) \models \Delta \), if \( D_E \models \delta \) for every \( \delta \in \Delta \). A rule \( r = q(x, y) \rightarrow EQ(x, y) \in \Gamma \) is satisfied in \( (D, E) \), written \( (D, E) \models r \), if \( q(D, E) \subseteq E \), and \( (D, E) \models \Gamma \) if all rules in \( \Gamma \subseteq \Gamma \) are satisfied. We call a pair \((c, c')\) of constants active in \((D, E)\) w.r.t. \( \Gamma \) if there exists a rule \( q(x, y) \rightarrow EQ(x, y) \in \Gamma \) such that \((c, c') \in q(D, E)\).

With these notions in hand, we are now able to formally define the semantics of ER specifications in terms of solutions.

**Definition 2.** Given a database \( D \) and ER specification \( \Sigma = (\Gamma, \Delta) \) over the same schema, we call \( E \) a candidate solution for \((D, \Sigma)\) if it satisfies one of the two conditions:

\((i)\) \( E = EQRel(\emptyset, D) \);

\((ii)\) \( E = EQRel(E' \cup \{a\}, D) \), where \( E' \) is a candidate solution for \((D, \Sigma)\) and \( a = (c_1, c_2) \) is active in \((D, E')\) w.r.t. \( \Gamma \).

A solution for \((D, \Sigma)\) is a candidate solution \( E \) for \((D, \Sigma)\) that further satisfies (a) \( (D, E) \models \Gamma_h \) and (b) \( (D, E) \models \Delta \). We denote by \( Sol(D, \Sigma) \) the set of solutions for \((D, \Sigma)\).

\(^1\)Also for simplicity, we use equivalence relations over the whole \( dom(D) \), but only constants occurring in merge attributes will belong to non-trivial equivalence classes.
δ₁ = ∀x, y, z, y′, ¬((Wrote(x, y, z) ∧ Wrote(x, y′, z) ∧ y ≠ y′)
δ₂ = ∀x, y, z, z′, ¬((Wrote(x, y, z) ∧ Wrote(x, y, z′) ∧ z ≠ z′)
δ₃ = ∀x, y, z, w, p, ¬((Paper(x, y, z) ∧ Wrote(x, w, p) ∧ Chair(z, w))
ρ₁ = ∃z, e, u, u′.CorrAuth(z, x) ∧ CorrAuth(z, y) ∧ Author(x, e, u) ∧ Author(y, e, u′) ⇒ EQ(x, y)
ρ₂ = ∃n, n′, ye, a.Conference(x, n, ye) ∧ Conference(y, n′, ye) ∧ Chair(x, a) ∧ Chair(y, a) ∧ n ≠ n′ ⇒ EQ(x, y)
σ₁ = ∃n, n′, ye, a.Conference(x, n, ye) ∧ Conference(y, n′, ye) ∧ Chair(x, a) ∧ Chair(y, a) ∧ n = n′ ⇒ EQ(x, y)
σ₂ = ∃e, e′, u. Author(x, u) ∧ Author(x, e′, u) ∧ e = e′ ⇒ EQ(x, y)
σ₃ = ∃t, t′, c, a, z.Paper(x, t, c) ∧ Paper(y, t′, c) ∧ Wrote(x, a, z) ∧ Wrote(y, a, z) ∧ t ≠ t′ ⇒ EQ(x, y)

Figure 1: A schema $S_{ex}$, database $D_{ex}$, and ER specification $S_{ex} = \{S_{ex}, \Delta_{ex}\}$ with $S_{ex} = \{\rho_1, \rho_2, \sigma_1, \sigma_2, \sigma_3\}$ and $\Delta_{ex} = \{\delta_1, \delta_2, \delta_3\}$. A fact of the form $Wrote(p, a, i)$ indicates that author $a$ appears at position $i$ in the list of the authors of paper $p$. The extension of the similarity predicate $\approx$ (restricted to dom($D_{ex}$)) is the symmetric and reflexive closure of $\{(e_1, e_2), (e_2, e_3), (e_3, e_4), (t_4, t_5), (n_2, n_3), (n_3, n_4)\}$, where $e_1, t_4, t_5$, and $n_2$ are the email of author $a_1$, title of paper $p_0$, and name of author $c_3$, respectively.

We note that a database–specification pair may admit zero, one, or several solutions. The absence of solutions arises from constraint violations (either initially present or introduced by the hard rules) which cannot be repaired solely through merges. When the original instance satisfies the constraints, the trivial equivalence relation (i.e., consisting only of pairs $(c, c)$) is always a solution. The existence of multiple solutions is due to some combinations of mergers not being possible without violating the constraints, leading to a choice of which possible merges to include.

Importantly, since rule bodies do not involve any kind of negation, a pair $\alpha$ that is active in $(D, E)$ remains active in $(D, E')$ for any $E \subseteq E'$. In particular, this means that if $\alpha$ is used to construct a solution $E$, then $\alpha$ remains active for $E$. Informally: later merges cannot invalidate the reasons for performing an earlier merge.

Rather than considering all solutions, it is natural to restrict attention to the ‘best’ ones. In this paper, we shall focus on solutions that are maximal w.r.t. set inclusion, i.e., they derive as many merges as possible, subject to the constraints. Alternative optimality criteria could also be considered, see Section 7 for discussion.

Definition 3. Given a database $D$ and ER specification $\Sigma = (\Gamma, \Delta)$ over the same schema, a solution $E$ for $D$ is called a maximal solution for $(D, \Sigma)$ if there is no solution $E'$ for $(D, \Sigma)$ with $E \subseteq E'$. We denote by MaxSol($D, \Sigma$) the set of maximal solutions for $(D, \Sigma)$.

Example 4. We now determine the maximal solutions for our example scenario $(D_{ex}, S_{ex})$. Initially, $E_0 = EqRel(\emptyset, D_{ex})$ and $D_{ex}E_0 = D_{ex}$. Note that $E_0$ is not a solution for $(D_{ex}, S_{ex})$ as $(D_{ex}, E_0) \not\models \delta_1$, due to $a_1, a_2, a_3$ appearing as first author of $p_1$. The following pairs are active in $(D_{ex}, E_0)$ w.r.t. $\Gamma_{ex}$: $\alpha = (a_1, a_2), \beta = (a_2, a_3), \chi = (a_6, a_7)$ due to $\sigma_2$, and $\xi = (c_2, c_3)$ and $\eta = (c_3, c_4)$ due to $\sigma_1$. By including $\alpha$ and $\beta$, we resolve the violation of $\delta_1$. In fact, it can be verified that every solution for $(D_{ex}, S_{ex})$ contains both $\alpha$ and $\beta$.

Adding $\alpha$ and $\beta$ (in either order) yields $E_1 = EqRel(\{a, b\}, D_{ex})$, which contains $(a_1, a_3)$ due to transitivity. Note that $E_1$ is not a solution as $(D_{ex}, E_1) \not\models \rho_2$. The only fix is to include $\xi$, which in turn blocks $\eta$. Indeed, if both $\xi$ and $\eta$ are present, then $c_2$ and $c_3$ are deemed the same, which means $a_1$ would be an author of the paper $p_0$ presented at the same conference that $(s)$he chaired, in violation of $\delta_3$.

With $\xi$ added, we get $E_2 = EqRel(\{a, b, \xi, \eta\}, D_{ex})$, which is a solution since $(D_{ex}, E_2) \models \{\rho_1, \rho_2\}$ and $(D_{ex}, E_2) \models \Delta_{ex}$. However, $E_2$ is not a maximal solution, as $\theta = (p_2, p_5)$ and $\lambda = (p_4, p_5)$ are now active due to $\sigma_3$, and $\chi$ remains active. Note however that extending $E_2$ by including both $\theta$ and $\chi$ would violate $\delta_3$.

Adding $\theta$ to $E_2$ leads us to $E_3 = EqRel(\{a, b, \xi, \theta\}, D_{ex})$. There is now a new active pair $\kappa = (a_3, a_4)\) in $(D_{ex}, E_3)$ due to $p_1$, which must be added to satisfy the hard rules. We can then obtain $M_1$ by adding $\lambda$ and $\kappa$ to $E_3$, i.e., $M_1 = EqRel(\{a, b, \xi, \theta, \lambda, \kappa\}, D_{ex})$, which is a maximal solution for $(D_{ex}, S_{ex})$ as no further active pair in $(D_{ex}, M_1)$ can be added without violating some denial constraint in $\Delta_{ex}$. Alternatively, we can obtain a second maximal solution $M_2$ for $(D_{ex}, S_{ex})$ by adding $\chi$ and $\kappa$ to $E_3$, i.e., $M_2 = EqRel(\{a, b, \xi, \theta, \chi, \kappa\}, D_{ex})$.

It can be verified that $M_1$ and $M_2$ are the only maximal solutions for $(D_{ex}, S_{ex})$, i.e., MaxSol($D_{ex}, S_{ex}$) = $\{M_1, M_2\}$.

The preceding example illustrates how constraint violations can be resolved using merges, and how the dynamic semantics enables us to obtain desirable merges. Observe the recursive dependencies: merging authors can lead to merging papers, which in turn may lead to further merges of authors.

Also note that all merges occurring in a solution are justified, in the sense that it is possible to trace back how each merge was obtained by a sequence of rule applications and closure steps.

Definition 4. Let $D$ be a database, $\Sigma = (\Gamma, \Delta)$ be an ER specification, $E \in Sol(D, \Sigma)$, and $(a, b) \in E$ (with $a \neq b$). A justification for $(a, b)$ w.r.t. $E$ and $(D, \Sigma)$ is a sequence $(e_1, e'_1), \ldots, (e_n, e'_n)$ such that
\{e_n, e'_n\} = \{a, b\} and for every 1 \leq i \leq n, (e_i, e'_i) \in E and one of the following conditions holds:
- e_j = e_k \land e'_j = e'_k for some j, k < i.
- there is P_1(v_1^1, \ldots, v_m^1) \land \cdots \land P_m(v_1^m, \ldots, v_m^m) \rightarrow EQ(x, y) \in \Gamma and facts P_1(c_1^1, c'_1^1), \ldots, P_m(c_1^m, c'_1^m) \in D such that:

Intuitively, the first item implements a single transitive closure step\(^5\), while the second corresponds to a rule application, in which previously derived merges may be used to 'join' database facts to satisfy the rule body. It follows easily from Definition 2 that every merge in a solution gives rise to at least one justification.

Example 5. In our running example, the sequence (c_2, c_3) is a one-step justification of the merge \(\zeta = (c_2, c_3)\ w.r.t. M_1 (D_{ex}, \Sigma_{ex}), as supported by the rule \(\sigma_1 \) and database facts

\[
Conference(c_2, n_2, 2019), Conference(c_3, n_3, 2019), n_2 \approx n_3,
\]

where we use \(n_i\) for the name of conference \(c_i, for i \in \{2, 3\}.

Another justification for \(\zeta\) is \((a_1, a_2), (a_2, a_3), (a_1, a_3), (c_2, c_3)\), supported by the following transitivity step and rule applications:

- apply \(\sigma_2\) using facts Author\((a_1, e_1, O)\), Author\((a_2, e_2, O)\), \(e_1 \approx e_2\)
- apply \(\sigma_2\) using facts Author\((a_2, e_2, O)\), Author\((a_3, e_3, O)\), \(e_2 \approx e_3\)
- transitively close \((a_1, a_2)\) and \((a_2, a_3)\)
- apply \(\rho_2\) using Conference\((c_2, n_2, 2019)\), Conference\((c_3, n_3, 2019)\), Chair\((c_2, a_1)\), Chair\((c_3, a_3)\), \(n_2 \approx n_3\) and joining via \((a_1, a_3)\)

where O stands for Oxford, and \(e_1\) and \(n_1\) are used for the email and name of author \(a_1\) and conference \(c_1\), respectively.

When solutions are numerous, it can be helpful to summarize them using the notions of certain and possible merges.

Definition 5. Given a database \(D\) and ER specification \(\Sigma\) over the same schema, we call \((c_1, c_2)\) a possible merge for \(D\ w.r.t. \Sigma\) if \((c_1, c_2)\) \(\in E\) for some \(e_1 \in MaxSol(D, \Sigma)\), and we call it a certain merge for \(D\ w.r.t. \Sigma\) if additionally \((c_1, c_2)\) \(\in E\) for every \(e \in MaxSol(D, \Sigma)\). We use \(\text{possMerge}(D, \Sigma)\) and \(\text{certMerge}(D, \Sigma)\) to denote the sets of possible and certain merges for \(D\ w.r.t. \Sigma\).

Observe that by requiring certain merges to be possible merges, we ensure that \(\text{certMerge}(D, \Sigma) = \emptyset\) whenever \(\text{Sol}(D, \Sigma) = \emptyset\).

Example 6. Continuing our running example, we can easily verify that \((i) \eta \notin \text{possMerge}(D_{ex}, \Sigma_{ex})\), \((ii) \chi \land \lambda \in \text{possMerge}(D_{ex}, \Sigma_{ex})\) but not to \(\text{certMerge}(D_{ex}, \Sigma_{ex})\), and \((iii) \alpha, \beta, \zeta, \theta, \text{and } \kappa\ \text{are all in certMerge}(D_{ex}, \Sigma_{ex})\).

Interestingly, since inequalities are allowed in denial constraints, each hard rule can be simulated with a soft one together with a denial constraint. Specifically, given \(\rho = q(x, y) \Rightarrow EQ(x, y)\) and \(q(x, y) = 3EQ(x, y, z)\), we use the soft rule \(\rho' = q(x, y) \Rightarrow EQ(x, y)\) and denial constraint \(\rho' = \neg q(x, y, z) \land z \neq y\).

Proposition 1. Let \(\Sigma = (\Gamma_1, \Gamma_2, \Delta)\) be an ER specification over \(S\), and let \(\Sigma' = (\Gamma'_1, \Delta')\) be the ER specification with \(\Gamma'_1 = \Gamma_1 \cup \{\sigma_0\}\) and \(\Delta' = \Delta \cup \{\delta_0\}\). Then \(\Sigma\) and \(\Sigma'\) are equivalent in the following sense: \(\text{Sol}(D, \Sigma) = \text{Sol}(D, \Sigma')\) for each \(S\)-database \(D\).

*Note that we deliberately omit reflexivity and symmetry steps from justifications, as we judge them as uninformative to users. This explains why we use \(\{e_n, e'_n\} = \{a, b\}\) rather than \(\{e_n, e'_n\} = \{a, b\}\) and similarly for \(\{e_j, e'_j\} = \{x, y\}\).*

4 COMPLEXITY ANALYSIS

In this section, we analyze the computational complexity of the central decision problems associated with the framework. All of the results are provided w.r.t. data complexity measure [40], i.e. the complexity is w.r.t. the size of the database \(D\) (and also the equivalence relation \(E\) for problems that require it). For the convenience of the reader, Table 1 summarizes the obtained results.

4.1 Solution Existence & Recognition

We first consider the solution recognition problem (Rec), which is to decide whether an input set of pairs is a solution.

Theorem 1. Rec is \(P\)-complete. The lower bound holds even for ER specifications consisting of a single hard rule.

Proof Sketch. Hardness is shown by a reduction from a variant of the propositional Horn entailment problem [10]. For the upper bound, we first check that \((D, \Sigma) \models \Gamma_1\) and \((D, \Sigma) \models \Delta\). If these checks succeed, then we verify whether \(E\) is a candidate solution as follows. Starting from \(E' = \text{EqRel}(0, D)\), we repeat the following step until a fixpoint is reached: if there is some pair \((c, c')\) \(\in E\) such that \((c, c')\) is active in \((D, E')\) w.r.t. \(\Gamma\) and \((c, c')\) \(\notin E'\), then set \(E' = \text{EqRel}(E' \cup \{(c, c')\})\). If \(E\) coincides with the computed \(E'\), then \(E\) is a solution.

By contrast, we show that the Existence problem of determining whether a given pair \((D, \Sigma)\) admits a solution is intractable.

Theorem 2. Existence is \(NP\)-complete.

Proof Sketch. Membership in \(NP\) is easy: guess a candidate and check if it is a solution. The lower bound is by reduction from the satisfiability problem for propositional 3CNF. Consider a 3CNF \(\phi = c_1 \land \cdots \land c_m\) over the variables \(x_1, \ldots, x_n\), where \(c_i = \ell_{i,1} \lor \ell_{i,2} \lor \ell_{i,3}\). Denote by \(x_{i,j}\) the variable of \(\ell_{i,j}\), and set \(s_{i,j} = i\) if \(\ell_{i,j} = x_{i,j}\) and \(s_{i,j} = j\) if \(\ell_{i,j} = \neg x_{i,j}\). We encode \(\phi\) using the following database:

\[
D^\phi = \{\{x_i\} | 1 \leq i \leq n\} \cup \{\text{Prec}(x_i, x_{i+1}) | 1 \leq i < n\}
\cup \{\text{R}_{s_{i,j}+1}(x_1, x_2) | 1 \leq i \leq m\}
\cup \{\text{FV}(x_1), \text{LV}(x_n), C_1(c_1), C_2(c_2), T(1), F(0), Q(0), Q(1)\}
\]

For instance, a clause \(x_k \lor \neg x_k\) is represented as \(R_{ij}(x_k, x_k)\). The fixed ER specification \(\Sigma_{SSAT}\) contains soft rules \(V(x) \land Q(y) \land \text{FV}(x) \Rightarrow EQ(x, y)\), \(V(x) \land Q(y) \land \text{Prec}(x_p, x) \land Q(x_p) \Rightarrow EQ(x, y)\), and \(C_1(x) \land C_2(y) \land Q(z) \land LV(z) \Rightarrow EQ(x, y)\). The first two enable each variable \(x_j\) to merge with either 0 or 1. Once every variable has been assigned a truth value in this manner, the third rule allows \(C_1\)- and \(C_2\)-marked clauses to merge together. Denial constraints ensure the merges yield a proper truth assignment \((\exists y \neg (F(y) \land T(y)))\) that does not violate any clause \((\forall y_1, y_2, y_3 \neg (R_{ij}(y_1, y_2, y_3) \land F(y_1) \land T(y_2) \land F(y_3)))\), and similarly for other clause types). A final constraint \((\forall y_1, y_2, \neg (C_1(y_1) \land C_2(y_2) \land y_3 \neq y))\) requires \(c_1\) and \(c_2\) to be merged, which means a full truth assignment is generated. It is not too hard to see that \(\phi\) is satisfiable iff \(\text{Sol}(D^\phi, \Sigma_{SSAT}) \neq \emptyset\).

The problem \(\text{MaxRec}\) of recognizing maximal solutions can also be shown to be intractable using a similar reduction.

Theorem 3. \(\text{MaxRec}\) is \(coNP\)-complete.
Theorem 4. CertMerge is $\Pi_2^P$-complete.

Proof Sketch. For the upper bound, we can show a pair $(d, e)$ is not certain by guessing $E$ with $(d, e) \not\in E$ and checking that $E$ is a maximal solution. The lower bound is by reduction from QBF validity problem for $\forall \exists$-3CNF \cite{39}. Given a $\forall \exists$-3CNF instance $\Phi = \forall X, \exists Y, \exists c_1 \land \ldots \land c_p$ over variables $X = (x_1, \ldots, x_m)$ and $Y = (y_1, \ldots, y_n)$, we construct the database $D^\Phi$ that contains: $V_X(x_i)$ for each $1 \leq i \leq n$, $V_Y(y_j)$ for each $1 \leq i \leq m$, $\text{Pred}(y_i, y_{i+1})$ for each $1 \leq i < m$, a $R_{x_1, x_2, \ldots, x_m}$-fact to encode each clause $c_k$ (as in the proof of Theorem 2), and the facts $FV_Y(y_1), LV_Y(y_m), C_1(c_1), C_2(c_2), C(c), C'(c'), T(1), F(0), Q(0), \text{and } Q(1)$.

The specification $\Sigma_{\forall \exists}$ borrows ideas from the proof of Theorem 2, e.g. only allowing $c_1$ and $c_2$ to merge if every variable from $Y$ has merged with either 0 or 1. The modified constraint $V_{y_1, y_2, \neg(C'(y) \land C''(y) \land C_1(y_1) \land C_2(y_2))}$ is now violated only if $c$ and $c'$ already merged, as made possible by $C(x) \land C'(x) \rightarrow \text{Eq}(x, y)$. The merging of $Y$ variables with 0 or 1 can only begin once $c$ and $c'$ are merged: $V_Y(x) \land Q(y) \land FV_Y(x) \land C(z) \land C'(z) \rightarrow \text{Eq}(x, y)$. No such requirement is imposed on the $X$ variables: $V_X(x) \land Q(y) \rightarrow \text{Eq}(x, y)$. Denial constraints are again employed to check the induced assignment does not falsify any clause. It can be shown that $\Phi$ is valid iff $(c, c') \in \text{certMerge}(D^\Phi, \Sigma_{\forall \exists})$. \hfill $\Box$

The dual problem PossMerge of recognizing possible merges has lower complexity: as every solution is contained in a maximal one, it suffices to exhibit any solution that contains the target merge.

Theorem 5. PossMerge is NP-complete.

Proof Sketch. The upper bound is based upon guessing a solution $E$ that contains the input pair. For the lower bound, we consider the specification $\Sigma'_{\forall \exists}$ (proof of Theorem 2) by dropping the constraint $V_{y_1, y_2, \neg(C_1(y_1) \land C_2(y_2))}$ in $\Sigma_{\forall \exists}$. Then $\phi$ is satisfiable iff $(c_1, c_2) \in \text{possMerge}(D^\Phi, \Sigma'_{\forall \exists})$. \hfill $\Box$

4.3 Query Answering

It may not always be feasible to examine the possible merges to identify which maximal solution corresponds to the true state of affairs. However, we can still obtain useful information by considering those answers which hold in some (resp. all) databases induced by a maximal solution. Formally:

Definition 6. Given a database $D$, ER specification $\Sigma$, and query $q$, all over the same schema, a tuple $\bar{a}$ of constants from $D$ is a possible answer to $q$ on $D$ w.r.t. $\Sigma$ if $\bar{a} \in q(D, \varepsilon)$ for some $E \in \text{MaxSol}(D, \Sigma)$; it is a certain answer to $q$ on $D$ w.r.t. $\Sigma$ if additionally $\bar{a} \in q(D, \varepsilon)$ for every $E \in \text{MaxSol}(D, \Sigma)$. We use possAns$(q, D, \Sigma)$ and certAns$(q, D, \Sigma)$ to denote the sets of possible and certain answers to $q$ on $D$ w.r.t. $\Sigma$.

We consider the decision problems CertAnswer and PossAnswer of testing whether a tuple belongs to certAns$(q, D, \Sigma)$, resp. possAns$(q, D, \Sigma)$, and show that they have the same complexity as the corresponding problems for merges. The upper bounds hold not only for CQs but for all classes of queries that can be evaluated in P and which are preserved under homomorphisms \cite{?}.

Theorem 6. CertAnswer is $\Pi_2^P$-complete.

Theorem 7. PossAnswer is NP-complete.

We remark that Definition 6 ensures that certAns$(q, D, \Sigma) = \text{possAns}(q, D, \Sigma) = \emptyset$ when $\text{Sol}(D, \Sigma) = \emptyset$. Alternatively, we could follow the ex falso sequitur quodlibet principle and deem all tuples of constants occurring in the database as possible and certain when $\text{Sol}(D, \Sigma) = \emptyset$. This has no impact on the complexity for certain answers, but it would cause PossAnswer to become coNP-complete.

4.4 Restricted Settings

Given the intractability results, we explore the impact of placing different syntactic restrictions on ER specifications. A first idea may be to use FDs in lieu of arbitrary denial constraints. However, all of our lower bounds can in fact be modified to work with ER specifications whose set of constraints contains only FDs. While FDs are central in traditional database settings, denial constraints without $\neq$-atoms figure prominently in ontology-mediated query answering, e.g. to express the disjointness axioms found in popular ontology languages like DL-Lite \cite{15} and the OWL 2 profiles \cite{34} and to express policies in controlled query evaluation \cite{17}. As the next result shows, adopting restricted ER specifications, whose denial constraints do not use any inequality atoms, brings a decrease in complexity (under the usual complexity assumptions) for several of our problems. Intuitively, this is due to constraint violations being preserved under merges, i.e. if $\delta$ is a denial constraint without $\neq$-atoms, then $D_E \not\models \delta$ implies $D_{E'} \not\models \delta$ when $E \subseteq E'$. 

Theorem 8. For restricted ER specifications, we have that:
- both Existence and MaxRec are P-complete;
- both CertMerge and CertAnswer are coNP-complete.

We also identify more severe restrictions that ensure that there is at most one maximal solution, computable in a deterministic fashion, thereby rendering all of the considered problems tractable.

Theorem 9. For ER specifications $(\Gamma, \Delta)$ such that either $\Gamma = \emptyset$ or $\Delta = \emptyset$, Rec, MaxRec, Existence, CertMerge, PossMerge, CertAnswer, and PossAnswer are all P-complete.

\footnote{We recall that coDP contains those decision problems that are the union of a NP problem and coNP problem \cite{37}.}
5  ANSWER SET PROGRAMMING ENCODING

To lay the grounds for implementing our framework, we show how the computational problems can be solved using answer set programming (ASP) [24, 33], a well-studied paradigm for declarative problem solving for which there exist highly optimized systems\(^4\).

5.1  ASP Basics

We briefly recall logic programs and stable model semantics, which form the core of ASP\(^1\). A disjunctive rule has the form

\[ r = H_1 \lor \ldots \lor H_f \leftarrow B_1, \ldots, B_m, \text{not } C_1, \ldots, \text{not } C_n \]

where \( \ell, m, n \geq 0 \), \( 1 + m + n \geq 0 \), \( \text{Head}(r) = \{H_1, \ldots, H_f\} \), \( \text{Body}^+(r) = \{B_1, \ldots, B_m\} \), and \( \text{Body}^-(r) = \{C_1, \ldots, C_n\} \) are sets of relational atoms, and every variable occurs in an atom in \( \text{Body}^+(r) \).

Disjunctive rules extend classical Datalog rules by allowing negated body atoms and disjunctive ruleheads. A (disjunctive logic) program \( \Pi \) is a finite set of disjunctive rules. A program is called normal if its every rule contains at most one head atom, and it is ground if all of its atoms are ground (i.e. variable-free).

We give the semantics first for ground programs. An interpretation \( I \) for a ground program \( \Pi \) is a subset of the ground atoms occurring in \( \Pi \); it is a model of \( \Pi \) if for every rule \( r \in \Pi \), either \( \text{Head}(r) \cap I \neq \emptyset \) or \( \text{Body}^+(r) \not\subseteq I \lor I \cap \text{Body}^-(r) \neq \emptyset \). The reduct of a ground program \( \Pi \) w.r.t. interpretation \( I \), denoted \( \text{reduct}(\Pi, I) \), is the program obtained by: (i) removing rules \( r \) with \( \text{Body}^+(r) \cap I \neq \emptyset \), and (ii) removing negated body atoms from all other rules. An interpretation \( M \) is a stable model of a ground program \( \Pi \) if \( M \) is a \( \subseteq \)-minimal model of \( \text{reduct}(\Pi, M) \). In the case of normal programs, stable models are the sets of positive atoms \( M \) which can be derived from \( \Pi \) by assuming the negation of atoms not present in \( M \).

The semantics is extended to general programs via grounding. Given a program \( \Pi \) and database \( D \), we use \( \text{gr}(\Pi, D) \) to denote the ground program that consists of all facts in \( D \) together with all ground instantiations of rules from \( \Pi \) with constants from \( \Pi \cup D \). The stable models of \( (\Pi, D) \) (aka answer sets) are the stable models of \( \text{gr}(\Pi, D) \); we call \( (\Pi, D) \) coherent if it has a stable model.

5.2  Generating Solutions

We first define a normal logic program \( \Pi_{\text{Sol}} \) that can be used to generate solutions of \((D, \Sigma)\). It will consist of the following rules:

- for every hard rule \( q(x, y) \rightarrow E\text{Q}(x, y) \):
  \[ E\text{Q}(x, y) \leftarrow q^+ \]
- for every denial constraint \( \forall x. \neg (\varphi(x)) \):
  \[ \neg \varphi^+ \]
- for every soft rule \( q(x, y) \rightarrow E\text{Q}(x, y) \):
  \[ E\text{Q}(x, y) \leftarrow q^+ \]
- two rules capturing the choice to add or omit an active pair:
  \[ E\text{Q}(x, y) \leftarrow \text{Active}(x, y), \text{not } Ne\text{Q}(x, y) \]
  \[ Ne\text{Q}(x, y) \leftarrow \text{Active}(x, y), \text{not } E\text{Q}(x, y) \]
- the following rules to enforce that \( E\text{Q} \) is an equivalence relation:
  \[ E\text{Q}(y, x) \leftarrow E\text{Q}(x, y) \]
  \[ E\text{Q}(x, z) \leftarrow E\text{Q}(x, y), E\text{Q}(y, z) \]
  \[ E\text{Q}(x, y) \leftarrow \text{Adom}(x) \]

\(^4\)Examples include clingo and wasp, see also a recent survey on ASP systems [26].

\(^1\)Modern ASP systems support logic programs with a rich syntax and many expressive features: function symbols, arithmetic operators, aggregation, optimization, etc.

\(^6\)Facts correspond to rules with empty bodies.

- the following rules to compute the active domain:
  \[ \text{Adom}(x_i) \leftarrow P(x_1, \ldots, x_n) \quad (P/n \in S, 1 \leq i \leq n) \]

To complete the definition of \( \Pi_{\text{Sol}} \), we explain how \( q^+ \) and \( \varphi^+ \) are defined. Intuitively they weaken the original queries by allowing two occurrences of the same variable to be mapped to different constants if they have been determined to denote the same entity. Given a CQ \( q(x, y) = \bigwedge_1 a_i \), we obtain \( q^+ \) from \( q \) as follows:

- each \( a_i = P(v_1, \ldots, v_j) \) is replaced by \( a_i^+ = P(v_1, \ldots, v_j^i) \) (where \( v_j^i = v_j \) if \( v_j \) is a constant, else \( v_j^i \) is a fresh copy of variable \( v_j \))
- for every pair \( a_i, a_k \in q \) such that variable \( v \) occurs in both \( a_i \) and \( a_k \), we add the atom \( Eq(v^i, v^k) \)
- pick a single copy \( v^i \) of the distinguished variable \( x \) and replace all occurrences of \( v^i \) by \( x \); we proceed analogously for \( y \).

We define \( \varphi^+ \) in a similar manner except: (i) we skip the third item, and (ii) we additionally include ‘not \( Eq(v^i, u^k) \)’ for every inequality \( v \neq u \in \varphi \) and pair of fresh variables of the forms \( u^i, u^k \).

Example 7. The encoding of \( \delta_1 \) of our running example is:

\[ \neg \text{Wrote}(x^1, y^1, z^1), \text{Wrote}(x^2, y^1, z^2), \]
\[ Eq(x^3, x^2), Eq(z^1, z^2), \text{not} Eq(y^1, y^2) \]

The solutions of \( \Sigma \) then correspond to the projection of the stable models of \( \Pi_{\text{Sol}} \) onto the predicate \( Eq \).

Theorem 10. For every database \( D \) and ER specification \( \Sigma : E \in \text{Sol}(D, \Sigma) \) iff \( E = \{(a, b) \mid Eq(a, b) \in M\} \) for some stable model \( M \) of \( (\Pi_{\text{Sol}}, D) \). In particular, \( \text{Sol}(D, \Sigma) \neq \emptyset \) iff \( (\Pi_{\text{Sol}}, D) \) is coherent.

It follows that we can use the enumeration facilities of ASP solvers to generate one or all solutions, and variants of \( \Pi_{\text{Sol}} \) can be used to solve the PossMerge and PossAnswer decision problems. To produce all possible merges / answers, we can employ the brave reasoning mode of ASP systems, which allows one to compute the union of all stable models (without naïvely computing all of them). Furthermore, we can employ off-the-shelf explanation tools for ASP\(^2\) to produce derivations of \( Eq \)-facts appearing in stable models, from which we can extract justifications as defined in Section 3.

5.3  Maximal Solutions

To capture maximal solutions, we need a way to restrict our attention to stable models \( M \) of \( (\Pi_{\text{Sol}}, D) \) such that there is no stable model \( M' \) with strictly more \( Eq \)-facts. Rather than crafting an encoding from scratch, we can take advantage of a line of work on incorporating preferences into ASP:

- The meta-programming approach from [25] provides a method for constructing, for any normal program \( \Pi \) and target relation \( T \), a program \( \Pi^C_T \) whose stable models correspond to the stable models of \( \Pi \) which have a \( \subseteq \)-maximal set of \( T \)-facts. The stable models of \( (\Pi^C_T, D) \) thus capture the maximal solutions of \((D, \Sigma)\), so we can use cautious reasoning (identifying facts common to all stable models) to return all certain merges / answers.
- The Aspirin framework [11] also provides native support for inclusion-based preferences, but uses iterative calls to an ASP solver to more efficiently compute preferred stable models.

\(^2\)A recent tool for explaining conclusions of ASP programs is xclingo [14].
Both approaches are implemented and available for use.

6 CONNECTION TO RELATED APPROACHES

In this section, we investigate the relationship between LACE and three logic-based frameworks for entity resolution. We mainly focus on these frameworks as they share certain features with LACE.

6.1 A Declarative Approach to Entity Linking

LACE shares with the entity-linking framework (EL) [12] the idea of specifying descriptions in a declarative way using rules, providing a rigorous semantics, and generating a space of solutions. Rules in EL are given as matching constraints (MCs) of the form L(x, y) → Condition, where Condition is a first-order formula expressing requirements for a pair to be considered a link. Such rules do not force the creation of any links and thus behave more similarly to our soft rules. However, Condition may include universal quantification, which is not expressible in our rule bodies. The most expressive core dialect of EL, L2, is geared to collective entity linking scenarios and allows link relations to be used in Condition and for recursion between link predicates. Formally, a specification in L2 is a triple H = (L, S, Ω), where for each link symbol L ∈ L, Ω contains at most one MC with left side L(x, y), two inclusion dependencies for L, and zero, one, or two FDs over L. In the semantics of EL, MCs are used to ‘statically’ ensure that solutions together with the database fulfill desired properties, rather than constructing solutions dynamically step by step. More precisely, an L-database J is a solution for D w.r.t. H if (D, J) |= Ω, where (D, J) is the S ∪ L database D ∪ J. Maximal solutions (w.r.t. ⊆) are considered, as well as quantitative notions of optimality based upon weights.

It is claimed [12] that the EL framework can be specialized to the ER task. Alas, the details on how exactly to carry out such a specialization are not provided. Naturally, one could consider adding a post-processing step to close solutions. For a more faithful comparison, we will consider the sg property graph over the subclass of directed bidirectional chain graphs (dgbc). We do so as solutions in EL are not necessarily equivalence relations, and over dgbc graphs, it is not needed to close the pairs transitively. We delay the formal definition of dgbc graphs to the appendix as it is rather technical and directly state the obtained result.

Theorem 11. There is no entity-linking specification H = (Ω, L, S) in L2 that expresses the sg property over dgbc graphs, i.e. such that, for every SG-database DG representing a dgbc graph G, L(a, b) is a certain link if (a, b) is a pair of sg nodes in G.

By contrast, in LACE, we are still able to capture the sg property over dgbc graphs even if we weaken the semantics to omit the closure operation (and thus no longer require solutions to be equivalence relations). Indeed, it suffices to add a further soft rule: V(x) → EQ(x, x). We can thus conclude that the expressive power of our ER specifications is not subsumed by the expressive power of entity-linking specifications in L2, even if we adopt a pared-down version of our semantics to aid the comparison. Note that the result holds irrespective of whether expressivity is measured in terms of certain links/merges as done in [13], or using maximal solutions.

6.2 Dedupalog

Dedupalog [1] is another logic-based framework for collective entity resolution. LACE borrows from Dedupalog the idea of including soft and hard rules. Interestingly, Dedupalog also allows for soft rules with negated heads, to indicate likely non-merges. As in the EL case, Dedupalog adopts a static view of the semantics. In addition to this, as further crucial differences we mention that: (i) while LACE aims at merging as many references as possible, Dedupalog aims at minimizing the number of violations of soft rules; and (ii) Dedupalog further requires the mandatory presence of soft-complete rules, which provide ‘soft’ sufficient and necessary conditions for two entity references to be merged. This leads to cases where it turns out to be convenient to not merge two references, despite the existence of a soft rule which supports the merge without contradicting any further constraint.

In light of these considerations, we believe that a formal comparison of expressive power between LACE and Dedupalog would not be informative, although similar arguments as the one provided for the EL case apply due to the static view of the semantics. Furthermore, due to requirement (ii) and the semantics adopted, we also argue that it is hard to devise Dedupalog specifications for inherently recursive scenarios, or for scenarios in which there can be more than one reason to merge two references.

Finally, note that the actual Dedupalog implementation is an algorithm that produces an approximately optimal solution, and makes its own choices of which merges to apply. Thus, as also noted

---

\[\text{Without (3), the query would be the so-called same-generation query [32].}\]
6.3 Matching Dependencies

Matching dependencies (MDs) have been introduced to specify that a pair of attribute values occurring as arguments in two database facts have to be matched [19, 21], i.e., made equal, if certain similarity conditions hold between (possibly other) values occurring in those facts. Formally, an MD takes the form \( R_1[\bar{X}_1] = R_2[\bar{X}_2] \), and states that if the projections of an \( R_1 \)-fact \( f_1 \) and \( R_2 \)-fact \( f_2 \) onto the attributes \( \bar{X}_1 \) and \( \bar{X}_2 \) respectively are pairwise similar, then the \( Y_1 \)-value of \( f_1 \) and the \( Y_2 \)-value of \( f_2 \) must be made equal. Relational MDs [3, 5] allow for additional atoms in the bodies of MDs to provide context and are thus more similar to LACE rules.

(Relational) MDs are equipped with a dynamic semantics, which can be defined via a chase-like procedure that fixes the violation of MDs. While the first works on MDs did not specify how values are made equal, Bertossi et al. [8] introduced matching functions to define what new value results from matching a pair of values. Although (relational) MDs can be seen as hard rules (since they must be satisfied), the order of rule application matters due to the modification of values, leading to a set of possible solutions. By contrast, in LACE, the existence of multiple solutions stems from the combination of soft rules and denial constraints.

Due to the different settings, the ASP encodings developed for (relational) MDs [3, 4] differ from ours in several respects. Most notably, the MD encodings employ an ordering to keep track of different versions of a tuple (nothing like this appears in our encoding), whereas a key challenge for us is that the certainty problems are at the second level of the polynomial hierarchy (the certain answer problem for MDs is ‘only’ coNP-complete [4, 8]).

As already noted, a fundamental difference between LACE and MDs is that our merges apply globally, throughout the database, while MDs only change the two \( \text{occurrences} \) of the constants involved in a specific MD. This difference is explained by the fact that we focus on merging constants denoting references, whereas MDs target constants denoting attribute values. For instance, local merges allow for some occurrences of ‘ISWC’ to be matched to ‘Int. Semantic Web Conf.’ and others to ‘Int. Symp. on Wearable Computing’, without (wrongly) equating the latter two constants.

7 CONCLUSION AND FUTURE WORK

We presented LACE, a new logical framework for entity resolution which employs declarative specifications to handle complex collective ER settings, while ensuring that all merges can be justified. We have argued that this trio of desirable features (collectivity, declarativity, and justifiability) is better supported in LACE than in other existing logical approaches to ER. We explored the computational properties of our framework, establishing the precise data complexity of the main decision problems related to reasoning over LACE specifications and identifying some syntactic restrictions that lead to lower complexity. To lay the groundwork for implementation, we further showed how the reasoning tasks could be realized using the functionalities of modern ASP solvers. This promising initial investigation opens up many interesting research directions:

Local merges. We believe that the LACE and MD approaches are complementary, and it would be fruitful to combine them to obtain a framework that allows for both global and local merges. Indeed, local merges may trigger global merges by making similarity atoms hold or could resolve FD violations which would otherwise block desirable global merges (e.g. two author IDs cannot merge due to different variants of the author’s name). Likewise, global merges could enable local merges. Such an extension could be accomplished by adding a local version of EQ which is an equivalence relation over value occurrences (with tuple identifiers to identify where a value occurs), allowing hard and soft rules for the local EQ relation, and adopting a strategy for evaluating similarity predicates over sets of equivalent cell values (e.g. take the minimal similarity value).

Quantitative extensions. Using maximal set containment to define good solutions might in some cases be too coarse, as it does not take into account the strength of evidence for a merge. It would thus be interesting to equip rules with quantitative information and use it to assign weights or probabilities to merges and solutions. One might further include soft rules with negative heads (for likely non-merges) and compare the evidence for and against a merge.

Tractable subclasses. It would also be worthwhile to investigate other restricted forms of specifications that may yield lower complexity. In particular, we may consider whether the various syntactic and semantic restrictions presented in [3, 5, 7] for (relational) MDs (e.g. the so-called SFAI class) can be adapted to our setting.

Explanation facilities. Our notion of justification provides the basis for explaining to a user how a given merge was obtained. It would be interesting however to consider how best to present justifications to users and also to explore more sophisticated forms of explanations that concern the whole space of (maximal) solutions, e.g. explaining why a given pair is (or is not) a certain merge.

Implementation. We want to develop an efficient prototype based on the presented ASP encodings and experiment it on existing ER benchmarks [28, 29]. Significant tuning and specialized optimizations will likely be required to achieve reasonable performance. In particular, we plan to develop static analysis techniques for reducing the number of references to be compared (blocking). It could also be interesting to explore the interaction with machine learning techniques, as done in ERBlox [5].

Ontologies. We could also extend LACE with ontological information. We believe that for ontology languages supporting first-order rewritability, such as those in the DL-Lite family [15], our complexity results could be lifted. The effect on the complexity of considering other more expressive ontology languages is unclear.

Repairing and deduplicating. While merges can resolve some constraint violations (i.e. those resulting from different representations of the same entity), a holistic framework for data quality will need to combine ER with traditional database repair operations [6]. How do we extend the LACE framework to simultaneously tackle both ER and database repairs, and how do we handle the interaction between fact deletions and merges?

ACKNOWLEDGEMENTS

This work has been supported by the ANR AI Chair INTENDED (ANR-19-CHIA-0014), and by the Royal Society (IES/R3/193236).
Let $\Sigma = (\Gamma_h \cup \Gamma_h', \Delta)$ be an ER specification over $S$, and let $\Sigma' = (\Gamma'_h, \Delta')$ be the ER specification with $\Gamma'_h = \Gamma_h \cup \{\sigma_\rho | \rho \in \Gamma_h\}$ and $\Delta' = \Delta \cup \{\delta_\rho | \rho \in \Gamma_h\}$. Then $\Sigma$ and $\Sigma'$ are equivalent in the following sense: $\text{Sol}(D, \Sigma) = \text{Sol}(D, \Sigma')$ for each $S$-database $D$.

**Proof Sketch.** To prove the claim, note that it is enough to show the following: given two ER specifications $\Sigma = (\Gamma, \Delta)$ and $\Sigma' = (\Gamma', \Delta')$, if $\Gamma'_h = \Gamma_h \setminus \{\rho \} \cup \{\sigma_\rho | \rho \in \Gamma_h\}$ and $\Delta' = \Delta \cup \{\delta_\rho | \rho \in \Gamma_h\}$, then $\Sigma$ and $\Sigma'$ are equivalent.

Consider any database $D$. We shall show that $\text{Sol}(D, \Sigma) = \text{Sol}(D, \Sigma')$. First note that $(D, \Sigma)$ and $(D, \Sigma')$ have the same set of candidate solutions, since the notion of ‘active’ rule does not distinguish between hard rules and soft rules. Let $E \subseteq \text{Sol}(D, \Sigma)$, i.e. $E$ is a candidate solution for $(D, \Sigma)$ satisfying $(D, E) \models \Gamma_h$ and $(D, E) \models \Delta$. From $(D, E) \models \rho$, we obtain $D_E \models \delta_\rho$. Since $D_E \models \delta_\rho$ for each $\delta_\rho \in \Delta_h$, this yields $(D, E) \models \Delta'_h$ as well. Therefore, $E$ is a solution for $(D, \Sigma')$ (condition $(D, E) \models \Gamma'_h \in \Sigma'$).

So, $\text{Sol}(D, \Sigma) \subseteq \text{Sol}(D, \Sigma')$. Let $E \subseteq \text{Sol}(D, \Sigma')$, i.e. $E$ is a candidate solution for $(D, \Sigma')$ satisfying $(D, E) \models \Gamma'_h$ and $(D, E) \models \Delta'$. Satisfaction of the latter, and in particular the fact that $D_E \models \delta_\rho$ implies $(D, E) \models \rho$. Since $(D, E) \models \rho'$ for each $\rho' \in \Gamma'_h$ holds by the assumption $(D, E) \models \Gamma'_h$, we derive that $(D, E) \models \Gamma_h$ as well. Therefore, $E$ is a solution for
Theorem 2. the second literal is negative, and third literal is positive (every such
and eight additional constraints
input Horn clauses, with each clause represented twice, using two
P
D

We shall consider the schema
It is well known that the satisfiability and entailment problems
propositional Horn 3CNF formulas are P-hard. For our reduction,
more convenient to work with a variant, Horn-Atl, which
takes as input a Horn formula \( \phi = \lambda_1 \land \ldots \land \lambda_m \), with variables
\( V = \{v_1, \ldots, v_n\} \) and with each \( \lambda_i \) taking the form \( v_j \land v_k \rightarrow v_h \) or
\( T \land T \rightarrow v_h \), and decides whether it is the case that \( \phi \models v_1 \land \ldots \land v_n \).
It can be shown that this variant remains P-hard.
We shall consider the schema \( S = \{R/4\} \) and use \( R \) to store the
input Horn clauses, with each clause represented twice, using two
copies of the head variable. The fixed specification \( S_{HORN-ALL} \) will consist of a single hard rule:
\[
\rho = \exists x \lambda \cdot x_1, x_2, R(z, x_1, x_2, x) \land R(z, x_1, x_2, y) \models EQ(x, y),
\]
which will be used to merge a propositional variable \( x \) with its copy
\( y \) whenever both variables \( x_1 \) and \( x_2 \) have been already merged with their respective copies.
Now take some Horn-all instance \( \phi = \lambda_1 \land \ldots \land \lambda_m \) of the
form described earlier. We construct an \( S \)-database \( D^\phi \) as follows:
- for each \( \lambda_i = T \land T \rightarrow v_h \), we include the facts \( R(t_i, t, t, v_h) \) and
\( R(t_i, t, t, v_h') \), which will force \( v_h \) and \( v_h' \) to be merged due to \( \rho \);
- for each \( \lambda_i = v_j \land v_k \rightarrow v_h \), we include the facts \( R(t_i, v_j, v_k, v_h) \) and
\( R(t_i, v_j', v_k', v_h') \), thus merging constants \( v_h \) and \( v_h' \) if both
pairs \( (v_j, v_j') \) and \( (v_k, v_k') \) have been previously merged.
As candidate solution, we consider the equivalence relation
\[
E_V \equiv \{(t, t) \cup \{(t, i) \mid 1 \leq i \leq m\} \cup \{(v_j, v_j'), (v_k, v_k'), (v_j', v_j), (v_k', v_k) \mid 1 \leq j \leq n\}
\]
Clearly, \( D^\phi \) and \( E_V \) can be constructed in LogSpace in \( \phi \). It can be shown that \( \phi \models v_1 \land \ldots \land v_n \) iff \( E_V \models D^\phi \). □

Theorem 2. Existence is \( \text{NP} \)-complete.

Proof. We provide further details for the sketched reduction. Recall that the specification \( S_{\text{SAT}} \) contains three soft rules:
\[
- \sigma_1 = V(x) \land Q(y) \land FV(x) \rightarrow EQ(x, y),
- \sigma_2 = \exists x, V(x) \land Q(y) \land Prec(x, y) \land Q'(x) \rightarrow EQ(x, y),
- \sigma_3 = \exists x, C_1(x) \land C_2(y) \land Q(x) \land LV(R) \rightarrow EQ(x, y),
\]
and its set \( \Delta \) contains ten denial constraints, which are:
\[
- \delta_1 = \forall y_1, y_2 \cdot \neg(C_1(y_1) \land C_2(y_2) \land y_1 \neq y_2),
- \delta_2 = \forall y \cdot \neg F(y) \land \neg T(y),
\]
and eight additional constraints \( \delta_3, \ldots, \delta_9 \), one for each relation \( R_{s_1, s_2, s_3, s_4, s_5} \). For example, \( v_1, y_1, y_2, \neg(R_1(y_1, y_2, v_1) \land F(y_1) \land T(y_2) \land F(y_2)) \) is the constraint for \( R_1 \), which serves to forbid truth assignments that would violate clauses whose first literal is positive, second literal is negative, and third literal is positive (every such clause being represented by a \( R_{\text{ff}} \)-fact).

To complete the proof, we need to show that \( \phi \) is satisfiable iff \( \text{Sol}(D^\phi, S_{\text{SAT}}) \neq \emptyset \).
First suppose that \( \phi = c_1 \land \ldots \land c_m \) is satisfiable, and let \( b_1, \ldots, b_n \in \{0,1\}^n \) be a truth assignment for \( x_1, \ldots, x_n \) that satisfies \( \phi \). Set \( E = \text{EqRel}((x_1, b_1), \ldots, (x_n, b_n), (c_1, c_2), D^\phi) \). It is straightforward to verify that \( E \) is a candidate solution for \( (D^\phi, S_{\text{SAT}}) \) and that \( (D, E) \models \Delta \). Thus, \( E \in \text{Sol}(D^\phi, S_{\text{SAT}}) \).
Conversely, suppose that \( E \in \text{Sol}(D^\phi, S_{\text{SAT}}) \). It follows that \( D^\phi \models \delta_i \), which means that \( (c_1, c_2) \in E \). By examining the available rules, only \( \sigma_3 \) can be used to add \( (c_1, c_2) \). In turn, \( \sigma_3 \) can only be applied if the last variable \( x_n \) has been previously merged with either constant 0 or 1. Due to the structure of \( \sigma_3 \), it can be shown by a simple inductive argument that each of the preceding variables \( x_1, \ldots, x_{n-1} \) has also been merged with 0 or 1. Since by assumption, \( D^\phi \models \delta_2 \), it follows that 0 and 1 have not been merged, so for every \( x_i \), either \( (x_i, 0) \in E \) or \( (x_i, 1) \in E \) but not both. Moreover, due to the fact that \( D^\phi \models \delta_i \) for \( i \in [3,10] \), the truth assignment that assigns 0 to \( x_j \) if \( (x_j, 0) \in E \) and 1 to \( x_j \) if \( (x_j, 1) \in E \) cannot violate any of the clauses of \( \phi \), i.e. \( \phi \) is satisfiable.

As stated in the body of the paper, not all our lower bounds can be modified to work with ER specifications whose set of denial constraints contains only FDs. We show how to adapt the reduction for the Existence problem (Theorem 2), and similar modifications can be used for the other lower bounds.

Theorem 12. Existence is \( \text{NP} \)-hard even for ER specifications whose set of denial constraints contains only FDs.

Proof. The proof is again by reduction from 3SAT. The schema will be similar to the one from Theorem 2, except that the \( R_{s_1, s_2, s_3, s_4, s_5} \) and C relations will have an extra attribute, and we use a binary relation \( FT \) in place of the unary relations \( T \) and \( F \).
Given a 3CNF instance \( \phi = c_1 \land \ldots \land c_m \) over variables \( x_1, \ldots, x_n \), we consider the following database \( D_{\phi}^\text{FD} \):
\[
\{ \langle x_1 \rangle \mid 1 \leq i \leq n \} \cup \{ \langle F(x_1) \rangle \cup \{ \langle \text{Prec}(x_1, x_2) \rangle \mid 1 \leq i < n \} \cup \{ \langle C_1(c_1), C_2(c_2), FT(0, c), FT(1, c), FT(0, c'), FT(1, c), Q(1) \rangle \cup \{ \langle R_{\text{ff}}(1, 1, c', c'_R), R_{\text{ff}}(1, 0, c', c'_R), R_{\text{ff}}(0, 1, c', c'_R), R_{\text{ff}}(0, 0, c', c'_R) \rangle \cup \{ \langle x_1, x_2, x_3, x_4 \rangle \mid 1 \leq i \leq m \}
\]
where the polarities \( s_1, s_2, s_3 \) of literals in a clause are defined as in Theorem 2.

The specification \( S_{\text{SAT}}^{\text{FD}} \) will contain the soft rules \( \sigma_1 \) and \( \sigma_2 \), as well as the modified soft rule
\[
\sigma'_1 = \exists z, z'. C(z, x) \land C(z, x) \land Q'(z') \land LV(z') \rightarrow EQ(x, y)
\]
The denial constraints (all corresponding to FDs) are as follows:
\[
- \delta_C = V(x, y_1, y_2, \neg(C(x, y_1) \land C(x, y_2) \land y_1 \neq y_2),
- \delta_{FT} = V(x, y_1, y_2, \neg(F(x, y_1) \land FT(x, y_2) \land y_1 \neq y_2)
- \delta_\ell = \forall x_1, x_2, x_3, y_1, y_2, \neg(R_1(x_1, x_2, x_3, y_1) \land R_1(x_1, x_2, x_3, y_2) \land y_1 \neq y_2),
\]
Intuitively, \( \delta_C \) forces \( c_1 \) and \( c_2 \) to merge and thus replaces \( \delta_1 \) from Theorem 2, whereas \( \delta_{\ell} \) replaces \( \delta_2 \) and serves to forbid 0 and 1 merging (so that each variable receives a unique truth value).
constraints $\delta_7$ are used in place of $\delta_3, \ldots, \delta_{10}$ to ensure that the obtained truth assignment satisfies all clauses. For example, if the data contains $R_{\text{rel}}(x_7, x_k, x_l, c_P)$ (representing the clause $x_j \lor \neg x_k \lor x_l$) and we merge $x_j$ with 0, $x_k$ with 1, and $x_l$ with 0 (thus falsifying the clause), then $\delta_{\text{rel}}$ will be violated due to fact $R_{\text{rel}}(0, 1, 0, c_P)$.

Following an argument similar to the one used for Theorem 2, it can be shown that $\phi$ is satisfiable iff $\text{Sol}(D^\phi_{FD}, \Sigma^\text{FD}_{\text{SAT}}) \neq \emptyset$. □

Theorem 3. MaxRec is coNP-complete.

Proof Sketch. For the upper bound, we can decide $E \not\in \text{MaxSol}(D, \Sigma)$ in NP by first guessing $E'$ and then checking that either (i) $E \not\in \text{Sol}(D, \Sigma)$ or (ii) $E \subseteq E'$ and $E' \in \text{Sol}(D, \Sigma)$ holds.

The lower bound can be obtained with a modification of the lower bound proof of Theorem 2. The intuition is to introduce two new constants $c$ and $c'$, which can be merged by an additional soft rule, and the variable $x_1$ of the SAT instance $\phi$ can be merged with either 0 or 1 only if $c$ and $c'$ have been previously merged.

Recall the database $D^\phi$ encoding the SAT instance $\phi$ as in the proof of Theorem 2, and let $D^\phi_{c,c'} = D^\phi \cup \{C(c), C'(c')\}$. The ER specification $\Sigma^\text{SAT}_{\phi}$ is obtained from $\Sigma^\text{SAT}$ (from the proof of Theorem 2) by (i) replacing $V(x) \land Q(y) \land FV(x) \rightarrow EQ(x, y)$ with $\exists z. V(x) \land Q(y) \land FV(x) \land C(z) \land C'(z) \rightarrow EQ(x, y)$, including soft rule $C(z) \land C'(z) \rightarrow EQ(x, y)$, and (ii) replacing $\forall y_1, y_2, -C_2(y_1) \land C_2(y_2) \land y_1 \neq y_2$ with $\forall y_1, y_2, -C_2(y_1) \land C'(y') \land C_2(y_2) \land y_1 \neq y_2$, where $C_1$ and $C_2$ are merged only if $c$ and $c'$ already merged. Finally, let $E = \text{EqRel}(0, D^\phi_{c,c'})$.

With the correctness of the reduction provided in the proof of Theorem 2 at hand, it is not hard to see that $\phi$ is unsatisfiable if and only if $E \in \text{MaxSol}(D^\phi_{c,c'}, \Sigma^\text{SAT}_{\phi})$. □

Theorem 6. CertAnswer is P^2-complete.

Proof Sketch. Membership is by guess-and-check, and the lower bound adapts the proof of Theorem 4, by using the Boolean CQ $q = \exists z. C(z) \land C'(z)$ in place of the merge $(c, c')$. □

Theorem 7. PossAnswer is NP-complete.

Proof Sketch. The upper bound again exploits the fact that it is sufficient to consider (not necessarily maximal) solutions. For the lower bound, we employ a reduction similar to the one used for Theorem 5 but use $q = \exists z. C_1(z) \land C_2(z)$ in place of $(c_1, c_2)$. □

Theorem 8. For restricted ER specifications, we have that:
- both Existence and MaxRec are P-complete;
- both CertMerge and CertAnswer are coNP-complete.

Proof Sketch. We only provide the P upper bound for MaxRec, which we believe to be the most interesting. As a first step, we check whether $E \in \text{Sol}(D, \Sigma)$ using the technique illustrated in the proof sketch of Theorem 1. If $E \not\in \text{Sol}(D, \Sigma)$, then we return falsese; otherwise, we continue as follows. We collect in set $S$ all those pairs of constants $\alpha = (c, c')$ such that $\alpha$ is active in $(D, E)$ w.r.t. $\Gamma_1$ and $\alpha \not\in E$. For each $\alpha \in S$, we proceed as follows. Starting from $E' := \text{EqRel}(E \cup \{\alpha\}, D)$, we repeat the following step until a fixpoint is reached: if there exists a pair $(c, c')$ such that $(c, c')$ is active in $(D, E')$ w.r.t. $\Gamma_2$ and $(c, c') \not\in E'$, then set $E' := \text{EqRel}(E' \cup \{(c, c')\}, D)$. Once the fixpoint is reached, we simply check whether $(D, E') \models \Delta$. If this is the case for some $\alpha \in S$, then we return false; otherwise, we return true.

The intuition is that for $E \not\in \text{MaxSol}(D, \Sigma)$, it is enough to ‘minimally’ extend $E$ and see whether such a minimal extension leads to a solution for $(D, \Sigma)$. This is because, if $\Delta$ is a set of denial constraints without inequality atoms, then $(D, E) \not\models \Delta$ implies $(D, E') \not\models \Delta$ whenever $E \subseteq E'$ (thus making futile the consideration of ‘non-minimal’ extensions of $E$). □

C PROOF DETAILS FOR SECTION 5

Theorem 10. For every database $D$ and ER specification $\Sigma$: $E \in \text{Sol}(D, \Sigma)$ iff $E = \{(a, b) \mid \text{Eq}(a, b) \in M\}$ for some stable model $M$ of $(\Pi^\text{Sol}, D)$. In particular, $\text{Sol}(D, \Sigma) \neq \emptyset$ iff $(\Pi^\text{Sol}, D)$ is coherent.

Proof Sketch. Given $E \in \text{Sol}(D, \Sigma)$, let $M_E$ extend $D$ with the following facts: $\text{Adom}(d)$ for every $d \in \text{dom}(D)$, $\text{Eq}(a, b)$ for every $(a, b) \in E$, $\text{Active}(a, b)$ for every pair $(a, b)$ that is active in $(D, E)$ due to a soft rule, and $\text{Neg}(a, b)$ if $\text{Active}(a, b) \in M_E$ but $\text{Eq}(a, b) \not\in M_E$. We claim that $M_E$ is a stable model, i.e. the unique minimal model of $\text{reduct}(gr(\Pi^\text{Sol}, D))$. We describe a key part of the argument, which is to show that every fact in $M_E$ is entailed from $\text{reduct}(gr(\Pi^\text{Sol}, D))$, focusing on Eq-facts. We fix a sequence $E_0, a_0, E_1, \ldots, a_n, E_n$ such that $E_0 = \text{EqRel}(\emptyset, D)$, and for every $1 \leq i < n$, $E_{i+1} = \text{EqRel}(E_i \cup \{a_i\}, D)$ for some $a_i \not\in E_i$ that is active in $(D, E_i)$. By suitably enumerating the pairs in $E_0$ and each $E_{i+1} \setminus (E_i \cup \{a_i\})$, we obtain an enumeration of the $\text{Eq}$-facts in $M_E: \text{Eq}(e_1^0), \ldots, \text{Eq}(e_n^0), \text{Eq}(a_1), \text{Eq}(e_1^1), \ldots, \text{Eq}(e_1^n), \text{Eq}(a_2), \ldots, \text{Eq}(a_n), \text{Eq}(e_n^0), \ldots, \text{Eq}(e_n^n)$ such that each fact can be derived from $D$ and the preceding facts in the enumeration using the ground rules in $\text{reduct}(gr(\Pi^\text{Sol}, D))$. Intuitively, each $\text{Eq}(e_i^0)$ is obtained using groundings of $\text{Eq}(x, x) \leftrightarrow \text{Adom}(x)$ and the $\text{Adom}$ rules, each $\text{Eq}(e_i^j)$ (with $j > 0$) by applying the (instantiated) symmetry or transitivity rule, each $\text{Eq}(a_i)$ with $a_i$ added to $E_i$ due to $p = q(x, y) \rightarrow \text{Eq}(x, y)$ by the instantiation of the rule $\text{Eq}(a_i) \leftarrow q^*(a_i)$, and each $\text{Eq}(a_i)$ with $a_i$ added due to $\sigma = q(x, y) \rightarrow \text{Eq}(x, y)$ by the combination of (instantiations of) $\text{Active}(a_i) \leftarrow q^*(a_i)$ and $\text{Eq}(a_i) \leftarrow \text{Active}(a_i)$ (note that ‘not $\text{Neg}(a_i)$’ is dropped in the reduct since $\text{Neg}(a_i) \not\in M_E$).

Conversely, suppose that $M$ is a stable model of $(\Pi^\text{Sol}, D)$, and let $E_M = \{(a, b) \mid \text{Eq}(a, b) \in M\}$. As $M$ is a stable model, there exists an enumeration $\beta_1, \ldots, \beta_n$ of the facts in $M$ such that for every $1 \leq i \leq n$, there exists a ground rule $r_i \in \text{reduct}(gr(\Pi^\text{Sol}, D))$ whose head is $\beta_i$ and whose body consists of facts from $\text{dU}\{\beta_1, \ldots, \beta_{i-1}\}$. We may assume w.l.o.g. that the enumeration respects the following strategy: (i) first apply all instantiations of rules of the form $\text{Adom}(x_i) \leftarrow p(x_1, \ldots, x_n)$ (ii) next apply all instantiations of the reflexivity rule, (iii) next apply instantiations of the symmetry and transitivity rules, as long as possible, (iv) next apply a single instantiation of a rule associated with a hard or soft rule, (v) next apply any applicable instantiations of the ‘add’ and ‘omit’ rules, and finally repeat (iii)–(v) until all facts in $M$ have been produced. This ensures that the $\text{Eq}$-facts are suitably ordered so as to be grouped into a sequence $E_0, a_0, E_1, \ldots, a_n, E_n$ such that $E_0 = \text{EqRel}(\emptyset, D)$, and for every $1 \leq i < n$, $E_{i+1} = \text{EqRel}(E_i \cup \{a_i\}, D)$ for some $a_i$ active in $(D, E_i)$. As $M$ is a model of $(\Pi^\text{Sol}, D)$, and $\Pi^\text{Sol}$ contains analogs of the hard
rules and denial constraints from $S$, i.e. $E_M$ is a solution. □

D PROOF DETAILS FOR SECTION 6

Recall that we say that a pair $(v, v')$ of nodes is sg in a digraph $D$ just in the case that $(v, v') \in q_{SG}(D_G)$, where $D_G$ is the database that represents $G$ and $q_{SG}$ is the Datalog query with goal predicate $sg$ and the rules $(1) (sg(x, y) \leftarrow V(x, y) \land E(x, y) \land E(z, y) \land sg(z, z'))$ and $(3) (sg(y, x) \leftarrow sg(y, z) \land sg(z, y))$.

We first detail our claim that the ER specification $\Sigma_{SG} = ((\exists x. E(z, x) \land E(z, y) \rightarrow EQ(x, y)), \emptyset)$ expresses the sg property over digraphs, i.e. certMerge($D_G, \Sigma_{SG}$) = \{(v, v') \mid (v, v') \in sg \in G\} for every $S_{SG}$-database $D_G$ representing a digraph $G$.

Proposition 2. $\Sigma_{SG}$ expresses the sg property over digraphs.

Proof Sketch. As $\Sigma_{SG}$ contains no denial constraints, for every $S_{SG}$-database $D_G$, there is a unique maximal solution $M_G$ for ($D_G, \Sigma_{SG}$). It follows that certMerge($D_G, \Sigma_{SG}$) = $M_G$. To see why $M_G$ contains precisely the sg pairs of $G$, observe that rules (1) and (3) of $\Sigma_{SG}$ are handled directly by our semantics, which requires $M_G$ to be an equivalence relation over dom($D_G$), whereas rule (2) of $q_{SG}$ is captured by $\exists x. E(z, x) \land E(z, y) \rightarrow EQ(x, y)$, with our dynamic semantics ensuring that the rule is applied until fixpoint. □

We now formally define the class of dbgc graphs. Given $n, m \geq 0$, we define the digraph $G^m_n$ = ($V, E$) as follows:
- if $n = 0$: $V = \{u_1, \ldots, u_m\}$ and $E = \emptyset$;
- if $n > 1$: $V = \{g, g', v_1, v'_1, \ldots, v_n, v'_n, u_1, \ldots, u_m\}$ and $E = \{(g, g'), (g', g'') \mid C \cup C' \cup C'' \land (v_1, v_2), \ldots, (v_{n-1}, v_n) \land C'' = \{(g', v'_1), (v'_1, v'_2), \ldots, (v'_{n-1}, v'_{n})\}$. Intuitively, $G^m_n$ contains $m$ isolated nodes and, if $n \geq 1$, two length-$n$ chains originating from $g$ and $g', g''$-loop. By directed bidirectional chain graph (dbgc), we shall mean a digraph $G^m_n$ for some $n, m \geq 0$.

It can be easily verified that the pairs of sg nodes in $G^m_n$ are:

$q_{SG}(D_G)$ = \{(v, v') \mid v \in V \cup \{(v_1, v'), (v'_1, v) \mid i \leq n\}$.

Observe that rule (3) of $q_{SG}$ is irrelevant for dbgc graphs, i.e. $q_{SG}(D_G) = q'_{SG}(D_G)$ for every dbgc graph $G$, where $q'_{SG}$ is obtained from $q_{SG}$ by dropping rule (3). From this observation, one can immediately see that if even we weaken our semantics by omitting the closure operation (and thus no longer require solutions to be equivalence relations), the ER specification $q_{DBGC}^{\Sigma_{SG}}$ which adds to $\Sigma_{SG}$ the soft rule $V(x) \rightarrow EQ(x, x)$ expresses the sg property over digraphs, i.e. certMerge($D_G, \Sigma_{SG}$, $q_{DBGC}^{\Sigma_{SG}}$) = \{(v, v') \mid (v, v') \in sg \in G\} for every $S_{DBGC}$-database $D_G$ representing a dbgc graph $G$.

Now let us consider the EL framework. We shall say that an EL specification $\mathcal{H} = (\mathcal{L}, S_{SG}, \Omega)$ expresses the sg property over digraphs, if for every $S_{SG}$-database $D_G$ representing a dbgc graph $G$, the extension of $\mathcal{L}$ in the set of certain links for $D_G$ w.r.t. $\mathcal{H}$ contains exactly the pairs of sg nodes in $G$. One might be tempted to think that we can capture the sg property over dbgc graphs using the EL specification $\mathcal{H}' = (\mathcal{L}, S_{SG}, \Omega)$, where $\Omega$ is as follows: the inclusion dependencies are such that $L$ ranges over all constants from $V$, there are no FDs over $L$, and the unique MC for $L$ is:

$L(x, y) \rightarrow V(x) \lor V(y) \land x = y \lor \exists x, z'. E(x, z) \land E(z', y) \land L(z, z')$

We can prove that this is the case. Since there are no FDs over $L$, for every $S_{LG}$-database $D$, the set of certain links for $D$ w.r.t. $\mathcal{H}'$ coincides with the unique $\leq$-maximal $(\mathcal{L})$-database $J$ such that $(D, J) \models \Omega$. Consider the $S_{LG}$-database $D_G = (\mathcal{L}, S_{LG}, \Omega)$, $V(v_1), V(g, g'), E(g', g), E(g', v_1), E(g, v_1), E(g, v'_1)$ representing $C^0_G$, and let $J$ be the $\leq$-maximal $(\mathcal{L})$-database $J$ such that $(D_G, J) \models \Omega$. We claim that $(L(g, g') \in J$ and $L(g, v_1) \in J$. Since neither pair is sg in $C^0_G$, it follows that $\mathcal{H}'$ does not express the sg property over dbgc graphs. Let us thus suppose for a contradiction that either $L(g, g') \notin J$ or $L(g, v_1) \notin J$, and consider $J' = J \cup \{(L(g, g'), L(g', g))\}$. Observe that the instantiated MCs $(L(g, g') \rightarrow \exists z, z'. E(z, g) \land E(z', g') \land L(z, z')$ and $(L(g, v_1) \rightarrow \exists z, z'. E(z, g) \land E(z', g') \land L(z, z')$ are satisfied in $D_G \cup J'. When combined with $(D_G, J') \models \Omega$, this yields $(D_G, J') \models \Omega$. Thus, $J'$ is a solution for $D_G$ w.r.t. $\mathcal{H}'$ with $J \subseteq J'$, contradicting the maximality of $J$.

We now provide the proof sketch of Theorem 11.

Theorem 11. There is no entity-linking specification $\mathcal{H} = (\mathcal{L}, S_{SG}, \Omega)$ in $L_2$ that expresses the sg property over dbgc graphs, i.e. such that, for every $S_{SG}$-database $D_G$ representing a dbgc graph $G$, $L(a, b)$ is a certain link if $(a, b)$ is a pair of sg nodes in $G$.

Proof Sketch. Suppose that the EL specification $\mathcal{H} = (\mathcal{L}, S_{SG}, \Omega)$ (in the most expressive $L_2$ dialect of [12]) expresses the sg property over dbgc graphs, and its unique MC for $L$ is

$\omega = L(x, y) \rightarrow \forall a_i (x, y, a_i) \rightarrow a_1 \lor \ldots \lor a_k$,

where $\forall (x, y, a_i)$ is a (possible empty) conjunction of relational atoms over $S_{SG}$, and each disjunct $a_i$ is a conjunctive query over $S_{SG} \cup \{L\}$, possibly with equality atoms and built-in predicates.

The proof proceeds by systematically imposing additional structure upon $\mathcal{H}$ until the specification is sufficiently constrained so that we can show that it does not express the required property. More specifically, the key steps are to show the following:

(1) There are no FDs over $L$.

(2) The inclusion dependencies are $L(X) \subseteq V(A)$ and $L(Y) \subseteq V(A)$, with $(X, Y)$ and $A$ the attributes of $L$ and $V$, respectively.

(3) We may assume w.l.o.g. that $\omega = L(x, y) \rightarrow a_1 \lor \ldots \lor a_k$.

(4) No specification satisfying the preceding restrictions can express the sg property over dbgc graphs.

For points (1) and (2), consider the $S_{LG}$-database $D_G = (\mathcal{L}, S_{LG}, \Omega)$, $V(g), V(v_1), V(g'), V(v'_1), E(g, g'), E(g', g), E(g, v_1), E(g, v'_1), E(v_1, v'_1)$ As $(v_1, v_1)$ and $(v'_1, v'_1)$ are pairs of sg nodes in $G^1$, they must belong to the extension of $L$, hence we cannot have the FD $L$: $\{X\} \rightarrow Y$ (resp. $L$: $\{Y\} \rightarrow X$). Further note that $(u_1, u_1)$ is a pair of sg nodes in $G^1$. Thus, $\mathcal{H}$ must have the inclusion dependencies $L(X) \subseteq V(A)$ and $L(Y) \subseteq V(A)$, otherwise no solution for $D_G$ w.r.t. $\mathcal{H}$ can contain $L(u_1, u_1)$, and $(u_1, u_1)$ would not be a certain link. Points (3) and (4) require intricate arguments, which cannot be adequately summarized in a few lines. Let us simply note that unlike the arguments seen so far, they employ graphs $G^m_n$ where $m$ and $n$ are not bounded by a fixed constant and depend instead on the number of variables occurring in $\omega$. □