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

Víctor 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:
Meghyn Bienvenu, Gianluca Cima, and Víctor Gutiérrez-Basulto. 2022. LACE: A Logical Approach to Collective Entity Resolution. In *Proceedings of the 41st ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS '22), June 12–17, 2022, Philadelphia, PA, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3517804.3526233

PODS '22, June 12–17, 2022, Philadelphia, PA, USA.

ACM ISBN 978-1-4503-9260-0/22/06…$15.00
https://doi.org/10.1145/3517804.3526233

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 [36], which is why ER sometimes goes by the name of *record linkage* [23]. 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 [39]. 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 [19, 28, 31]. Being both an important and multi-faceted problem, ER has been tackled using various approaches with different formal foundations: probabilistic approaches [10, 39], deep learning approaches [24, 32, 37] and logical approaches [1, 13, 20, 22]; see [17] 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 [32].

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 [7] 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) [9, 20, 22] and extensions thereof, such as relational MDs [4, 6], 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. [13], both of which implicitly work with a global semantics.

Taking further inspiration from the entity linking approach [13] (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 to do so. In particular, the 'static' semantics of 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 (Π_2^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 [4, 5], 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 $\mathcal{S}$ of relation symbols, with each $R \in \mathcal{S}$ 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* $\mathcal{S}$ (or *($\mathcal{S}$-)database* for short) assigns to each k-ary relation symbol $R \in \mathcal{S}$ a finite k-ary relation over a fixed, infinite set of constants. Equivalently, we can view an $\mathcal{S}$-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 $\mathsf{dom}(D)$, 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* $\mathcal{S}$ takes the form $q(\vec{x}) = \exists \vec{y}.\varphi(\vec{x}, \vec{y})$, where $\vec{x}$ and $\vec{y}$ are disjoint lists of variables, and φ is a conjunction of relational atoms of the form $R(t_1, \ldots, t_k)$, with $R \in \mathcal{S}$ a k-ary relation symbol and each t_i is a constant or variable from $\vec{x} \cup \vec{y}$. The *arity* of a CQ $q(\vec{x})$ is its number of *distinguished variables* $\vec{x}$, 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 $\vec{c} = (c_1, \ldots, c_n)$, we denote by $q[\vec{c}]$ the Boolean query obtained by replacing each x_i by c_i. The *answers to an n-ary CQ $q(\vec{x})$ over a database D* is defined as usual as the set of n-tuples of constants $\vec{c}$ from D such that $q[\vec{c}]$ 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 notations 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* [7, 21] of the form $\forall \vec{x}.\neg(\varphi(\vec{x}))$, where $\varphi(\vec{x})$ 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 $\vec{x}$. Denial constraints notably generalize the well-known class of *functional dependencies (FDs)*. For example, the FD $R\colon \{A, B\} \rightarrow C$, for relation $R(A, B, C)$, is captured by the denial constraint $\forall x, y, z, z'.\neg(R(x, y, z) \wedge R(x, y, z') \wedge z \neq z')$.

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 $\mathcal{S}$)* takes the form

$$q(x, y) \Rightarrow \mathsf{EQ}(x, y),$$

where $q(x, y)$ is a CQ, whose atoms may use relation symbols in $\mathcal{S}$ as well as similarity predicates, and EQ is a special relation symbol (not in $\mathcal{S}$) 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

$$q(x, y) \dashrightarrow \mathsf{EQ}(x, y),$$

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 $q(x, y) \rightarrow \mathsf{EQ}(x, y)$ 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 $\mathcal{S}_{\mathsf{ex}}$ and ruleset Γ_{ex} of our running example. Hard rule ρ_2 states that if two conferences are held in the year, have the same chair, and have similar names (according to similarity predicate $\approx$), then they are actually the same conference. Soft rule σ_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* Γ if there exists a rule $q(x, y) \rightarrow \mathsf{EQ}(x, y) \in \Gamma$ and body atom $R(t_1, \ldots, t_k)$ such that $t_i \in \{x, y\}$; we call A_i a *sim attribute of* Γ if there is a rule $q(x, y) \rightarrow \mathsf{EQ}(x, y) \in \Gamma$ 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 Γ *sim-safe* if there is no attribute that is both a merge and sim attribute of Γ.

Example 2. *The sim attributes of* Γ_{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,* Γ_{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 $\mathcal{S}$ takes the form $\Sigma = \langle \Gamma, \Delta \rangle$, where $\Gamma = \Gamma_h \cup \Gamma_s$ is a finite sim-safe set of hard and soft rules over $\mathcal{S}$, and Δ is a finite set of denial constraints over $\mathcal{S}$.*

Example 3. *Our running example utilizes the ER specification Σ_{ex} from Figure 1. In addition to the ruleset Γ_{ex}, it contains three denial constraints: δ_1 and δ_2 are FDs for Wrote, while δ_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 $\mathsf{EqRel}(S, D)$ the least equivalence relation $E \supseteq S$ over $\mathrm{dom}(D)$. 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 $\mathrm{dom}(D)$, the *database induced by D and E*, denoted D_E, is the database obtained from D by replacing each constant c by $\mathsf{rep}_E(c)$. We then define the set $q(D, E)$ of *answers to a query $q(\vec{x})$ w.r.t. D and E* as follows:

$$(c_1, \ldots, c_n) \in q(D, E) \text{ iff } (\mathsf{rep}_E(c_1), \ldots, \mathsf{rep}_E(c_n)) \in q(D_E).$$

The sim-safe condition ensures that similarity predicates in rule bodies are handled correctly, e.g. if $\mathsf{rep}_E(d_1) \approx \mathsf{rep}_E(d_2)$ is used to satisfy a rule body in D_E, then $\mathsf{rep}_E(d_i) = d_i$ $(i = 1, 2)$, so $d_1 \approx d_2$.

A set of denial constraints Δ is satisfied in (D, E), written $(D, E) \models \Delta$, if $D_E \models \delta$ for every $\delta \in \Delta$. A rule $\gamma = q(x, y) \rightarrow \mathsf{EQ}(x, y) \in \Gamma$ is satisfied in (D, E), written $(D, E) \models \gamma$, 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.* Γ if there exists a rule $q(x, y) \rightarrow \mathsf{EQ}(x, y) \in \Gamma$ 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 $\Sigma = \langle \Gamma, \Delta \rangle$ over the same schema, we call E a* candidate solution for (D, Σ) *if it satisfies one of the two conditions:*

(i) $E = \mathsf{EqRel}(\emptyset, D)$*;*

(ii) $E = \mathsf{EqRel}(E' \cup \{\alpha\}, D)$*, where E' is a candidate solution for (D, Σ) and $\alpha = (c_1, c_2)$ is active in (D, E') w.r.t. Γ.*

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

[1] Also for simplicity, we use equivalence relations over the whole $\mathrm{dom}(D)$, but only constants occurring in merge attributes will belong to non-trivial equivalence classes.

Author(id, email, institution)

a_1	wchen@gm.com	Oxford
a_2	wchen@ox.uk	Oxford
a_3	chenw@ox.uk	Oxford
a_4	gln@nyu.us	NYU
a_5	gln@nyu.us	New York
a_6	mnk@tku.jp	Tokyo
a_7	mnk@gm.com	Tokyo

Paper(id, title, cID)

p_1	A Survey on Logic in CS	c_1
p_2	Declarative ER	c_2
p_3	Declarative ER (Ext. Abst.)	c_3
p_4	Semantic Data Integration	c_2
p_5	Data Integration	c_3
p_6	Basics of Data Science	c_4

Wrote(pID, aID, pos)

p_1	a_1	1
p_1	a_2	1
p_1	a_3	1
p_2	a_4	1
p_3	a_4	1
p_4	a_5	1
p_5	a_5	1
p_4	a_6	2
p_5	a_7	3
p_6	a_1	1

Conference(id, name, year)

c_1	PODS	2021
c_2	Conf. on Data Eng.	2019
c_3	Data Eng. Conf.	2019
c_4	Data Eng. & Analytics	2019

Chair(cID, aID)

c_2	a_1
c_3	a_3

CorrAuth(pID, aID)

p_2	a_4
p_3	a_5

$\delta_1 = \forall x, y, z, y'.\neg(\text{Wrote}(x, y, z) \wedge \text{Wrote}(x, y', z) \wedge y \neq y')$
$\delta_2 = \forall x, y, z, z'.\neg(\text{Wrote}(x, y, z) \wedge \text{Wrote}(x, y, z') \wedge z \neq z')$
$\delta_3 = \forall x, y, z, w, p.\neg(\text{Paper}(x, y, z) \wedge \text{Wrote}(x, w, p) \wedge \text{Chair}(z, w))$
$\rho_1 = \exists z, e, u, u'.\text{CorrAuth}(z, x) \wedge \text{CorrAuth}(z, y) \wedge \text{Author}(x, e, u) \wedge \text{Author}(y, e, u') \Rightarrow \mathsf{EQ}(x, y)$
$\rho_2 = \exists n, n', \text{ye}, a.\text{Conference}(x, n, \text{ye}) \wedge \text{Conference}(y, n', \text{ye}) \wedge \text{Chair}(x, a) \wedge \text{Chair}(y, a) \wedge n \approx n' \Rightarrow \mathsf{EQ}(x, y)$
$\sigma_1 = \exists n, n', \text{ye}.\text{Conference}(x, n, \text{ye}) \wedge \text{Conference}(y, n', \text{ye}) \wedge n \approx n' \dashrightarrow \mathsf{EQ}(x, y)$
$\sigma_2 = \exists e, e', u.\text{Author}(x, e, u) \wedge \text{Author}(y, e', u) \wedge e \approx e' \dashrightarrow \mathsf{EQ}(x, y)$
$\sigma_3 = \exists t, t', c, a, z.\text{Paper}(x, t, c) \wedge \text{Paper}(y, t', c) \wedge \text{Wrote}(x, a, z) \wedge \text{Wrote}(y, a, z) \wedge t \approx t' \dashrightarrow \mathsf{EQ}(x, y)$

Figure 1: A schema $\mathcal{S}_{\text{ex}}$, database D_{ex}, and ER specification $\Sigma_{\text{ex}} = \langle \Gamma_{\text{ex}}, \Delta_{\text{ex}} \rangle$ with $\Gamma_{\text{ex}} = \{\rho_1, \rho_2, \sigma_1, \sigma_2, \sigma_3\}$ and $\Delta_{\text{ex}} = \{\delta_1, \delta_2, \delta_3\}$. 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 $\{(e_1, e_2), (e_2, e_3), (e_6, e_7), (t_2, t_3), (t_4, t_5), (n_2, n_3), (n_3, n_4)\}$, where e_i, t_i, and n_i are the email of author a_i, title of paper p_i, and name of conference c_i, 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 permitted 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 merges 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 α that is active in (D, E) remains active in (D, E') for any $E \subseteq E'$. In particular, this means that if α is used to construct a solution E, then α 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 $\Sigma = \langle \Gamma, \Delta \rangle$ over the same schema, a solution E for (D, Σ) is called a* maximal solution for (D, Σ) *if there is no solution E' for (D, Σ) with $E \subsetneq E'$. We denote by $\mathsf{MaxSol}(D, \Sigma)$ the set of maximal solutions for (D, Σ).*

Example 4. *We now determine the maximal solutions for our example scenario $(D_{\text{ex}}, \Sigma_{\text{ex}})$. Initially, $E_0 = \mathsf{EqRel}(\emptyset, D_{\text{ex}})$ and $D_{\text{ex}E_0} = D_{\text{ex}}$. Note that E_0 is not a solution for $(D_{\text{ex}}, \Sigma_{\text{ex}})$ as $(D_{\text{ex}}, E_0) \not\models \delta_1$, due to a_1, a_2, a_3 all appearing as 1st author of p_1. The following pairs are active in (D_{ex}, E_0) w.r.t. Γ_{ex}: $\alpha = (a_1, a_2)$, $\beta = (a_2, a_3)$, and $\chi = (a_6, a_7)$ due to σ_2, and $\zeta = (c_2, c_3)$ and $\eta = (c_3, c_4)$ due to σ_1. By including α and β, we resolve the violation of δ_1. In fact, it can be verified that every solution for $(D_{\text{ex}}, \Sigma_{\text{ex}})$ contains both α and β.*

Adding α and β (in either order) yields $E_1 = \mathsf{EqRel}(\{\alpha, \beta\}, D_{\text{ex}})$, which contains (a_1, a_3) due to transitivity. Note that E_1 is not a solution as $(D_{\text{ex}}, E_1) \not\models \rho_2$. The only fix is to include ζ, which in turn blocks η. Indeed, if both ζ and η are present, then c_2 and c_4 are deemed the same, which means a_1 would be an author of the paper p_6 presented at the same conference that (s)he chaired, in violation of δ_3.

With ζ added, we get $E_2 = \mathsf{EqRel}(\{\alpha, \beta, \zeta\}, D_{\text{ex}})$, which is a solution since $(D_{\text{ex}}, E_2) \models \{\rho_1, \rho_2\}$ and $(D_{\text{ex}}, E_2) \models \Delta_{\text{ex}}$. However, E_2 is not a maximal solution, as $\theta = (p_2, p_3)$ and $\lambda = (p_4, p_5)$ are now active due to σ_3, and χ remains active. Note however that extending E_2 by including both λ and χ would violate δ_2.

Adding θ to E_2 leads us to $E_3 = \mathsf{EqRel}(\{\alpha, \beta, \zeta, \theta\}, D_{\text{ex}})$. There is now a new active pair $\kappa = (a_4, a_5)$ in (D_{ex}, E_3) due to ρ_1, which must be added to satisfy the hard rules. We can then obtain M_1 by adding λ and κ to E_3, i.e. $M_1 = \mathsf{EqRel}(\{\alpha, \beta, \zeta, \theta, \lambda, \kappa\}, D_{\text{ex}})$, which is a maximal solution for $(D_{\text{ex}}, \Sigma_{\text{ex}})$ as no further active pair in (D_{ex}, M_1) can be added without violating some denial constraint in Δ_{ex}. Alternatively, we can obtain a second maximal solution M_2 for $(D_{\text{ex}}, \Sigma_{\text{ex}})$ by adding χ and κ to E_3, i.e. $M_2 = \mathsf{EqRel}(\{\alpha, \beta, \zeta, \theta, \chi, \kappa\}, D_{\text{ex}})$.

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

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, $\Sigma = \langle \Gamma, \Delta \rangle$ be an ER specification, $E \in \mathsf{Sol}(D, \Sigma)$, and $(a, b) \in E$ (with $a \neq b$). A* justification for (a, b) w.r.t. E and (D, Σ) *is a sequence $(e_1, e_1'), \ldots, (e_n, e_n')$ such that*

$\{e_n, e'_n\} = \{a, b\}$ *and for every* $1 \le i \le n$, $(e_i, e'_i) \in E$ *and one of the following conditions holds:*

- $e_i = e_j$, $e'_j = e_k$, *and* $e'_k = e'_i$ *for some* $j, k < i$,
- *there is* $P_1(v_1^1, \ldots, v_1^{\ell_1}) \wedge \ldots \wedge P_m(v_m^1, \ldots, v_m^{\ell_m}) \rightarrow \mathsf{EQ}(x, y) \in \Gamma$ *and facts* $P_1(c_1^1, \ldots, c_1^{\ell_1}), \ldots, P_m(c_m^1, \ldots, c_m^{\ell_m}) \in D$ *such that: if* $v_r^s = v_t^u$ *and* $c_r^s \neq c_t^u$, *then* $\{c_r^s, c_t^u\} = \{e_j, e'_j\}$ *for some* $j < i$.

Intuitively, the first item implements a single transitive closure step[2], 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 *and* $(D_{\text{ex}}, \Sigma_{\text{ex}})$, *as supported by the rule* σ_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 ζ *is* $(a_1, a_2), (a_2, a_3), (a_1, a_3), (c_2, c_3)$, *supported by the following transitivity steps and rule applications:*

- *apply* σ_2 *using facts* $Author(a_1, e_1, O), Author(a_2, e_2, O), e_1 \approx e_2$
- *apply* σ_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* ρ_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_i *and* n_i *are used for the email and name of author* a_i *and conference* c_i, *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* Σ *over the same schema, we call* (c_1, c_2) *a* possible merge for D w.r.t. Σ *if* $(c_1, c_2) \in E$ *for some* $E \in \mathsf{MaxSol}(D, \Sigma)$, *and we call it a* certain merge for D w.r.t. Σ *if additionally* $(c_1, c_2) \in E$ *for every* $E \in \mathsf{MaxSol}(D, \Sigma)$. *We use* $possMerge(D, \Sigma)$ *and* $certMerge(D, \Sigma)$ *to denote the sets of possible and certain merges for D w.r.t.* Σ.

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

Example 6. *Continuing our running example, we can easily verify that (i)* $\eta \notin possMerge(D_{\text{ex}}, \Sigma_{\text{ex}})$, *(ii)* χ *and* λ *belong to* $possMerge(D_{\text{ex}}, \Sigma_{\text{ex}})$ *but not to* $certMerge(D_{\text{ex}}, \Sigma_{\text{ex}})$, *and (iii)* α, β, ζ, θ, *and* κ *are all in* $certMerge(D_{\text{ex}}, \Sigma_{\text{ex}})$.

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 $\rho = q(x, y) \Rightarrow \mathsf{EQ}(x, y)$ with $q(x, y) = \exists \vec{z}.\varphi(x, y, \vec{z})$, we use the soft rule $\sigma_\rho = q(x, y) \dashrightarrow \mathsf{EQ}(x, y)$ and denial constraint $\delta_\rho = \forall x, y, \vec{z}.\neg(\varphi(x, y, \vec{z}) \wedge x \neq y)$.

Proposition 1. *Let* $\Sigma = \langle \Gamma_h \cup \Gamma_s, \Delta \rangle$ *be an ER specification over* $\mathcal{S}$, *and let* $\Sigma' = \langle \Gamma'_s, \Delta' \rangle$ *be the ER specification with* $\Gamma'_s = \Gamma_s \cup \{\sigma_\rho \mid \rho \in \Gamma_h\}$ *and* $\Delta' = \Delta \cup \{\delta_\rho \mid \rho \in \Gamma_h\}$. *Then* Σ *and* Σ' *are equivalent in the following sense:* $\mathsf{Sol}(D, \Sigma) = \mathsf{Sol}(D, \Sigma')$ *for each* $\mathcal{S}$*-database D.*

[2]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 $\{c_r^s, c_t^u\} = \{e_j, e'_j\}$.

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 [41], 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 [11]. For the upper bound, we first check that $(D, E) \models \Gamma_h$ and $(D, E) \models \Delta$. If these checks succeed, then we verify whether E is a candidate solution as follows. Starting from $E' := \mathsf{EqRel}(\emptyset, D)$, 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. Γ and $(c, c') \notin E'$, then set $E' := \mathsf{EqRel}(E' \cup \{(c, c')\}, D)$. 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, Σ) 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 $\phi = c_1 \wedge \ldots \wedge c_m$ over the variables $x_1, \ldots, x_n$, where $c_i = \ell_{i,1} \vee \ell_{i,2} \vee \ell_{i,3}$. Denote by $x_{i,j}$ the variable of $\ell_{i,j}$, and set $s_{i,j} = t$ if $\ell_{i,j} = x_{i,j}$ and $s_{i,j} = f$ if $\ell_{i,j} = \neg x_{i,j}$. We encode ϕ using the following database:

$$\begin{aligned} D^\phi = {} & \{V(x_i) \mid 1 \le i \le n\} \cup \{Prec(x_i, x_{i+1}) \mid 1 \le i < n\} \\ & \cup \{R_{s_{i,1}s_{i,2},s_{i,3}}(x_{i,1}, x_{i,2}, x_{i,3}) \mid 1 \le i \le m\} \\ & \cup \{FV(x_1), LV(x_n), C_1(c_1), C_2(c_2), T(1), F(0), Q(0), Q(1)\} \end{aligned}$$

For instance, a clause $x_k \vee \neg x_z \vee x_w$ is represented as $R_{tft}(x_k, x_z, x_w)$.

The fixed ER specification $\Sigma_{3\text{SAT}}$ contains soft rules $V(x) \wedge Q(y) \wedge FV(x) \dashrightarrow \mathsf{EQ}(x, y)$, $V(x) \wedge Q(y) \wedge Prec(x_p, x) \wedge Q(x_p) \dashrightarrow \mathsf{EQ}(x, y)$, and $C_1(x) \wedge C_2(y) \wedge Q(z) \wedge LV(z) \dashrightarrow \mathsf{EQ}(x, y)$. The first two enable each variable x_i 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 $(\forall y.\neg(F(y) \wedge T(y)))$ that does not violate any clause $(\forall y_1, y_2, y_3.\neg(R_{tft}(y_1, y_2, y_3) \wedge F(y_1) \wedge T(y_2) \wedge F(y_3))$, and similarly for other clause types). A final constraint $\forall y_1, y_2.\neg(C_1(y_1) \wedge C_2(y_2) \wedge y_1 \neq y_2)$ 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 ϕ is satisfiable iff $\mathsf{Sol}(D^\phi, \Sigma_{3\text{SAT}}) \neq \emptyset$. □

The problem MaxRec of recognizing maximal solutions can also be shown to be intractable using a similar reduction.

Theorem 3. *MaxRec is coNP-complete.*

Note that the preceding results imply that there cannot exist any efficient algorithm for enumerating (maximal) solutions.

	Rec	MaxRec	Existence	CertMerge	PossMerge	CertAnswer	PossAnswer
General Specifications	P-c	coNP-c	NP-c	Π_2^p-c	NP-c	Π_2^p-c	NP-c
Restricted Specifications	P-c	P-c	P-c	coNP-c	NP-c	coNP-c	NP-c

Table 1: Data Complexity of Decision Problems. We use '-c' as an abbreviation for '-complete'.

4.2 Certain and Possible Merges

We next consider the problem CertMerge of determining whether a given pair is a certain merge and show that it lies at the second level of the polynomial hierarchy.

Theorem 4. *CertMerge is Π_2^p-complete.*

Proof Sketch. For the upper bound, we can show a pair (d, e) is not certain by guessing E with $(d, e) \notin E$ and checking that E is a maximal solution. The lower bound is by reduction from QBF validity problem for $\forall\exists$-3CNF [40]. Given a $\forall\exists$-3CNF instance $\Phi = \forall X.\exists Y.(c_1 \wedge \ldots \wedge c_p)$ over variables $X = (x_1, \ldots, x_n)$ and $Y = (y_1, \ldots, y_m)$, we construct the database D^Φ that contains: $V_X(x_i)$ for each $1 \leq i \leq n$, $V_Y(y_i)$ for each $1 \leq i \leq m$, $Prec(y_i, y_{i+1})$ for each $1 \leq i < m$, a $R_{s_{k,1}s_{k,2},s_{k,3}}$-fact to encode each clause c_k (as in the proof of Theorem 2), and the facts $FV_Y(y_1)$, $LV_Y(y_m)$, $C_1(c_1)$, $C_2(c_2)$, $C(c)$, $C'(c')$, $T(1)$, $F(0)$, $Q(0)$, 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 $\forall y, y_1, y_2.\neg(C(y) \wedge C'(y) \wedge C_1(y_1) \wedge C_2(y_2) \wedge y_1 \neq y_2)$ is now violated only if c and c' already merged, as made possible by $C(x) \wedge C'(x) \dashrightarrow \mathsf{EQ}(x, y)$. The merging of Y variables with 0 or 1 can only begin once c and c' are merged: $V_Y(x) \wedge Q(y) \wedge FV_Y(x) \wedge C(z) \wedge C'(z) \dashrightarrow \mathsf{EQ}(x, y)$. No such requirement is imposed on the X variables: $V_X(x) \wedge Q(y) \dashrightarrow \mathsf{EQ}(x, y)$. Denial constraints are again employed to check the induced assignment does not falsify any clause. It can be shown that Φ is valid iff $(c, c') \in \mathsf{certMerge}(D^\Phi, \Sigma_{\forall\exists})$. □

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 Σ'_{3SAT} obtained from Σ_{3SAT} (proof of Theorem 2) by dropping the constraint $\forall y_1, y_2.\neg(C_1(y_1) \wedge C_2(y_2) \wedge y_1 \neq y_2)$. Then ϕ is satisfiable iff $(c_1, c_2) \in \mathsf{possMerge}(D^\phi, \Sigma'_{3SAT})$. □

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 Σ, and query q, all over the same schema, a tuple $\vec{a}$ of constants from D is a* possible answer to q on D w.r.t. Σ *if $\vec{a} \in q(D, E)$ for some $E \in \mathsf{MaxSol}(D, \Sigma)$; it is a* certain answer to q on D w.r.t. Σ *if additionally $\vec{a} \in q(D, E)$ for every $E \in \mathsf{MaxSol}(D, \Sigma)$. We use $\mathsf{possAns}(q, D, \Sigma)$ and $\mathsf{certAns}(q, D, \Sigma)$ to denote the sets of possible and certain answers to q on D w.r.t. Σ.*

We consider the decision problems CertAnswer and PossAnswer of testing whether a tuple belongs to $\mathsf{certAns}(q, D, \Sigma)$, resp. $\mathsf{possAns}(q, D, \Sigma)$, 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* [3].

Theorem 6. *CertAnswer is Π_2^p-complete.*

Theorem 7. *PossAnswer is NP-complete.*

We remark that Definition 6 ensures that $\mathsf{certAns}(q, D, \Sigma) = \mathsf{possAns}(q, D, \Sigma) = \emptyset$ when $\mathsf{Sol}(D, \Sigma) = \emptyset$. 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 $\mathsf{Sol}(D, \Sigma) = \emptyset$. This has no impact on the complexity for certain answers, but it would cause PossAnswer to become coDP-complete[3].

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 [16] and the OWL 2 profiles [35] and to express policies in controlled query evaluation [18]. 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 δ 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 $\langle \Gamma, \Delta \rangle$ such that either $\Gamma_s = \emptyset$ or $\Delta = \emptyset$, Rec, MaxRec, Existence, CertMerge, PossMerge, CertAnswer, and PossAnswer are all P-complete.*

[3] We recall that coDP contains those decision problems that are the union of a NP problem and coNP problem [38].

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) [25, 34], 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[5]. A *disjunctive rule* has the form

$$r = H_1 \vee \ldots \vee H_\ell \leftarrow B_1, \ldots, B_m, \text{not } C_1, \ldots, \text{not } C_n$$

where $\ell, m, n \geq 0$, $l + m + n \geq 0$, $Head(r) = \{H_1, \ldots, H_\ell\}$, $Body^+(r) = \{B_1, \ldots, B_m\}$, and $Body^-(r) = \{C_1, \ldots, C_n\}$ are sets of relational atoms, and every variable occurs in an atom in $Body^+(r)$. Disjunctive rules extend classical Datalog rules by allowing negated body atoms and disjunctive ruleheads. A *(disjunctive logic) program* Π 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 Π is a subset of the ground atoms occurring in Π; it is a *model* of Π if for every rule $r \in \Pi$, either $Head(r) \cap I \neq \emptyset$ or $Body^+(r) \nsubseteq I$ or $I \cap Body^-(r) \neq \emptyset$. The *reduct* of a ground program Π w.r.t. interpretation I, denoted $reduct(\Pi, I)$, is the program obtained by: (i) removing rules r with $Body^-(r) \cap I \neq \emptyset$, then (ii) removing negated body atoms from all other rules. An interpretation M is a *stable model* of a ground program Π if M is a $\subseteq$-minimal model of $reduct(\Pi, M)$. In the case of normal programs, stable models are the sets of positive atoms M which can be derived from Π by assuming the negation of atoms not present in M.

The semantics is extended to general programs via grounding. Given a program Π and database D, we use $gr(\Pi, D)$ to denote the ground program that consists of all facts in D[6] together with all ground instantiations of rules from Π with constants from $\Pi \cup D$. The stable models of (Π, D) (aka *answer sets*) are the stable models of $gr(\Pi, D)$; we call (Π, D) *coherent* if it has a stable model.

5.2 Generating Solutions

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

- for every hard rule $q(x, y) \Rightarrow \mathsf{EQ}(x, y)$: $\quad Eq(x, y) \leftarrow q^+$
- for every denial constraint $\forall \vec{x}. \neg(\varphi(\vec{x}))$: $\quad \leftarrow \varphi^+$
- for every soft rule $q(x, y) \dashrightarrow \mathsf{EQ}(x, y)$: $\quad Active(x, y) \leftarrow q^+$
- two rules capturing the choice to add or omit an active pair:

$$Eq(x, y) \leftarrow Active(x, y), \text{not } Neq(x, y)$$
$$Neq(x, y) \leftarrow Active(x, y), \text{not } Eq(x, y)$$

- the following rules to enforce that Eq is an equivalence relation:

$$Eq(y, x) \leftarrow Eq(x, y) \qquad Eq(x, z) \leftarrow Eq(x, y), Eq(y, z)$$
$$Eq(x, x) \leftarrow Adom(x)$$

- the following rules to compute the active domain:

$$Adom(x_i) \leftarrow P(x_1, \ldots, x_n) \qquad (P/n \in \mathcal{S}, 1 \leq i \leq n)$$

To complete the definition of Π_{Sol}, we explain how q^+ and φ^+ 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 $q(x, y) = \bigwedge_i \alpha_i$, we obtain q^+ from q as follows:

- each $\alpha_i = P(v_1, \ldots, v_k)$ is replaced by $\alpha_i' = P(v_1^i, \ldots, v_k^i)$ (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 $\alpha_i, \alpha_k \in q$ such that variable v occurs in both α_i and α_k, we add the atom $Eq(v^i, v^k)$
- pick a single copy x^i of the distinguished variable x and replace all occurrences of x^i by x; we proceed analogously for y.

We define φ^+ 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 atom $v \neq u \in \varphi$ and pair of fresh variables of the forms v^i, u^k.

Example 7. *The encoding of δ_1 of our running example is:*

$$\leftarrow Wrote(x^1, y^1, z^1), Wrote(x^2, y'^1, z^2),$$
$$Eq(x^1, x^2), Eq(z^1, z^2), \text{not } Eq(y^1, y'^1)$$

The solutions of Σ then correspond to the projection of the stable models of Π_{Sol} onto the predicate Eq:

Theorem 10. *For every database D and ER specification Σ: $E \in \mathsf{Sol}(D, \Sigma)$ iff $E = \{(a, b) \mid Eq(a, b) \in M\}$ for some stable model M of (Π_{Sol}, D). In particular, $\mathsf{Sol}(D, \Sigma) \neq \emptyset$ iff (Π_{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 Π_{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[7] 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 (Π_{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 [26] provides a method for constructing, for any normal program Π and target relation T, a program $\Pi^{\subseteq T}$ whose stable models correspond to the stable models of Π which have a $\subseteq$-maximal set of T-facts. The stable models of $(\Pi_{Sol}^{\subseteq Eq}, D)$ thus capture the maximal solutions of (D, Σ), so we can use *cautious reasoning* (identifying facts common to all stable models) to return all certain merges / answers.
- The Aspirin framework [12] also provides native support for inclusion-based preferences, but uses iterative calls to an ASP solver to more efficiently compute preferred stable models.

[4] Examples include **clingo** and **wasp**, see also a recent survey on ASP systems [27].
[5] 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.
[7] A recent tool for explaining conclusions of ASP programs is **xclingo** [15].

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) [13] the idea of describing specifications 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 $\mathsf{L}(x, y) \rightarrow \mathit{Condition}$, where $\mathit{Condition}$ 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, $\mathit{Condition}$ may include universal quantification, which is not expressible in our rule bodies. The most expressive core dialect of EL, $\mathcal{L}_2$, is geared to collective entity linking scenarios and allows link relations to be used in $\mathit{Condition}$ and for recursion between link predicates. Formally, a specification in $\mathcal{L}_2$ is a triple $\mathcal{H} = \langle L, \mathcal{S}, \Omega \rangle$, where for each link symbol $\mathsf{L} \in L$, Ω contains at most one MC with left side $\mathsf{L}(x, y)$, 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 fulfil desired properties, rather than constructing solutions dynamically step by step. More precisely, an L-database J is a solution for D w.r.t. $\mathcal{H}$ if $(D, J) \models \Omega$, where (D, J) is the $\mathcal{S} \cup L$-database $D \cup J$. Maximal solutions (w.r.t. $\subseteq$) are considered, as well as quantitative notions of optimality based upon weights.

It is claimed [13] 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 to deal with ER one would need to specialize a general link relation L to an equivalence relation, or find a way to capture the fact that we are dealing with 'equality'. Probably the most natural way to adapt the EL approach to ER is to axiomatize equality using MCs, e.g. including $x = y$, $\mathsf{L}(y, x)$, and $\exists z.\mathsf{L}(x, z) \wedge \mathsf{L}(z, y)$ as disjuncts on the right side of a MC for $\mathsf{L}(x, y)$. However, we could then end up with all candidate pairs appearing in the maximal solution, mutually supporting each other's presence, leading to unjustified merges. Another possibility would be to modify the definition of solution to only accept as solutions those databases that are equivalence relations, but such a strong requirement would miss many expected solutions. Finally, one could consider adding a post-processing step to close solutions under symmetry, transitivity, and reflexivity, but the solutions produced could still be 'incomplete' since rules are not re-evaluated over the new merges to potentially derive more new merges.

Leaving aside the issue with equivalence relations, more fundamentally, we would like to understand what are the consequences of adopting either a static or dynamic view of the semantics, in particular for capturing inherently recursive scenarios. To this aim, we introduce the *transitive same-generation* property on directed graphs (digraphs). Given a digraph G, we denote by D_G the database over schema $\mathcal{S}_G = \{V/1, E/2\}$ that represents G, defined as expected. We say that a pair of nodes $(u, v) \in V^2$ are *sg* in a digraph G if they belong to the answers to the following Datalog query[8] (with goal predicate *sg*) over D_G: (1) $sg(x, x) \leftarrow V(x)$; (2) $sg(x, y) \leftarrow E(z, x) \wedge E(z', y) \wedge sg(z, z')$; (3) $sg(x, y) \leftarrow sg(x, z) \wedge sg(z, y)$. Let us suppose we have an $\mathcal{S}_G$-database D_G representing a digraph G such that v and u are actually denoting the same real-world entity iff (v, u) is a pair of *sg* nodes in G. In LACE, this can be done using a single soft rule: $\exists z.E(z, x) \wedge E(z, y) \dashrightarrow \mathsf{EQ}(x, y)$.

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* $\mathcal{H} = \langle \{\mathsf{L}\}, \mathcal{S}_G, \Omega \rangle$ *in* $\mathcal{L}_2$ *that expresses the sg property over dgbc graphs, i.e. such that, for every* $\mathcal{S}_G$*-database* D_G *representing a dgbc graph* G, $\mathsf{L}(a, b)$ *is a certain link iff* (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: $V(x) \dashrightarrow \mathsf{EQ}(x, x)$. We can thus conclude that the expressive power of our ER specifications is not subsumed by the expressive power of entity-linking specifications in $\mathcal{L}_2$, 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 [14], 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

[8]Without (3), the query would be the so-called *same-generation* query [33].

in [13], Dedupalog is not an entirely declarative approach since the user must be aware of how the rules are treated by the system.

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* [20, 22], i.e. made equal, if certain similarity conditions hold between (possibly other) values occurring in those facts. Formally, an MD takes the form $R_1[\vec{X}_1] \approx R_2[\vec{X}_2] \rightarrow R_1[Y_1] \doteq R_2[Y_2]$ and states that if the projections of an R_1-fact f_1 and R_2-fact f_2 onto the attributes $\vec{X}_1$ and $\vec{X}_2$ 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 [4, 6] 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. [9] 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 [4, 5] 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 [5, 9]).

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 *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 [4, 6, 8] 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 [29, 30]. 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 [6].

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 [16], 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 [7]. 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).

REFERENCES

[1] Arvind Arasu, Christopher Ré, and Dan Suciu. 2009. Large-Scale Deduplication with Constraints Using Dedupalog. In *Proc. of the Twenty-Fifth International Conference on Data Engineering (ICDE 2009)*. 952–963.

[2] Marcelo Arenas, Leopoldo E. Bertossi, and Jan Chomicki. 2003. Answer sets for consistent query answering in inconsistent databases. *Theory Pract. Log. Program.* 3, 4-5 (2003), 393–424.

[3] Albert Atserias, Anuj Dawar, and Phokion G. Kolaitis. 2006. On preservation under homomorphisms and unions of conjunctive queries. *J. ACM* 53, 2 (2006), 208–237.

[4] Zeinab Bahmani and Leopoldo E. Bertossi. 2017. Enforcing Relational Matching Dependencies with Datalog for Entity Resolution. In *Proc. of the Thirtieth International Florida Artificial Intelligence Research Society Conference (FLAIRS 2017)*. 718–723.

[5] Zeinab Bahmani, Leopoldo E. Bertossi, Solmaz Kolahi, and Laks V. S. Lakshmanan. 2012. Declarative Entity Resolution via Matching Dependencies and Answer Set Programs. In *Proc. of the Thirteenth International Conference on the Principles of Knowledge Representation and Reasoning (KR 2012)*.

[6] Zeinab Bahmani, Leopoldo E. Bertossi, and Nikolaos Vasiloglou. 2017. ERBlox: Combining matching dependencies with machine learning for entity resolution. *Int. J. Approx. Reason.* 83 (2017), 118–141.

[7] Leopoldo E. Bertossi. 2011. *Database Repairing and Consistent Query Answering*. Morgan & Claypool Publishers.

[8] Leopoldo E. Bertossi and Jaffer Gardezi. 2014. Tractable vs. Intractable Cases of Query Answering under Matching Dependencies. In *Proc. of the Eighth International Conference on Scalable Uncertainty Management (SUM 2014)*. 51–65.

[9] Leopoldo E. Bertossi, Solmaz Kolahi, and Laks V. S. Lakshmanan. 2013. Data Cleaning and Query Answering with Matching Dependencies and Matching Functions. *Theory Comput. Syst.* 52, 3 (2013), 441–482.

[10] Indrajit Bhattacharya and Lise Getoor. 2007. Collective entity resolution in relational data. *ACM Trans. Knowl. Discov. Data* 1, 1 (2007), 5.

[11] Egon Börger, Erich Grädel, and Yuri Gurevich. 1997. *The Classical Decision Problem*. Springer.

[12] Gerhard Brewka, James P. Delgrande, Javier Romero, and Torsten Schaub. 2015. asprin: Customizing Answer Set Preferences without a Headache. In *Proc. of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI 2015)*. 1467–1474.

[13] Douglas Burdick, Ronald Fagin, Phokion G. Kolaitis, Lucian Popa, and Wang-Chiew Tan. 2016. A Declarative Framework for Linking Entities. *ACM Trans. Database Syst.* 41, 3 (2016), 17:1–17:38.

[14] Douglas Burdick, Ronald Fagin, Phokion G. Kolaitis, Lucian Popa, and Wang-Chiew Tan. 2019. Expressive power of entity-linking frameworks. *J. Comput. Syst. Sci.* 100 (2019), 44–69.

[15] Pedro Cabalar, Jorge Fandinno, and Brais Muñiz. 2020. A System for Explainable Answer Set Programming. In *Proc. of the Thirty-Sixth International Conference on Logic Programming (ICLP 2020)*. 124–136.

[16] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. 2007. Tractable Reasoning and Efficient Query Answering in Description Logics: The *DL-Lite* Family. *J. Autom. Reason.* 39, 3 (2007), 385–429.

[17] Vassilis Christophides, Vasilis Efthymiou, Themis Palpanas, George Papadakis, and Kostas Stefanidis. 2021. An Overview of End-to-End Entity Resolution for Big Data. *ACM Comput. Surv.* 53, 6 (2021), 127:1–127:42.

[18] Gianluca Cima, Domenico Lembo, Lorenzo Marconi, Riccardo Rosati, and Domenico Fabio Savo. 2020. Controlled Query Evaluation in Ontology-Based Data Access. In *Proc. of the Nineteenth International Semantic Web Conference (ISWC 2020)*. 128–146.

[19] Xin Dong, Alon Y. Halevy, and Jayant Madhavan. 2005. Reference Reconciliation in Complex Information Spaces. In *Proc. of the 2005 ACM SIGMOD International Conference on Management of Data (SIGMOD 2005)*. 85–96.

[20] Wenfei Fan. 2008. Dependencies revisited for improving data quality. In *Proc. of the Twenty-Seventh ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS 2008)*. ACM, 159–170.

[21] Wenfei Fan and Floris Geerts. 2012. *Foundations of Data Quality Management*. Morgan & Claypool Publishers.

[22] Wenfei Fan, Xibei Jia, Jianzhong Li, and Shuai Ma. 2009. Reasoning about Record Matching Rules. *Proc. of the VLDB Endowment* 2, 1 (2009), 407–418.

[23] Ivan P. Fellegi and Alan B. Sunter. 1969. A Theory for Record Linkage. *J. Amer. Statist. Assoc.* 64, 328 (1969), 1183–1210.

[24] Cheng Fu, Xianpei Han, Jiaming He, and Le Sun. 2020. Hierarchical Matching Network for Heterogeneous Entity Resolution. In *Proc. of the Twenty-Ninth International Joint Conference on Artificial Intelligence (IJCAI 2020)*. 3665–3671.

[25] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. 2012. *Answer Set Solving in Practice*. Morgan & Claypool Publishers.

[26] Martin Gebser, Roland Kaminski, and Torsten Schaub. 2011. Complex optimization in answer set programming. *Theory Pract. Log. Program.* 11, 4-5 (2011), 821–839.

[27] Martin Gebser, Nicola Leone, Marco Maratea, Simona Perri, Francesco Ricca, and Torsten Schaub. 2018. Evaluation Techniques and Systems for Answer Set Programming: a Survey. In *Proc. of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI 2018)*. 5450–5456.

[28] Dmitri V. Kalashnikov and Sharad Mehrotra. 2006. Domain-independent data cleaning via analysis of entity-relationship graph. *ACM Trans. Database Syst.* 31, 2 (2006), 716–767.

[29] Pradap Konda, Sanjib Das, Paul Suganthan G. C., AnHai Doan, Adel Ardalan, Jeffrey R. Ballard, Han Li, Fatemah Panahi, Haojun Zhang, Jeffrey F. Naughton, Shishir Prasad, Ganesh Krishnan, Rohit Deep, and Vijay Raghavendra. 2016. Magellan: Toward Building Entity Matching Management Systems. *Proc. of the VLDB Endowment* 9, 12 (2016), 1197–1208.

[30] Hanna Köpcke, Andreas Thor, and Erhard Rahm. 2010. Evaluation of entity resolution approaches on real-world match problems. *Proc. of the VLDB Endowment* 3, 1 (2010), 484–493.

[31] Pigi Kouki, Jay Pujara, Christopher Marcum, Laura M. Koehly, and Lise Getoor. 2019. Collective entity resolution in multi-relational familial networks. *Knowl. Inf. Syst.* 61, 3 (2019), 1547–1581.

[32] Yuliang Li, Jinfeng Li, Yoshihiko Suhara, Jin Wang, Wataru Hirota, and Wang-Chiew Tan. 2021. Deep Entity Matching: Challenges and Opportunities. *ACM J. Data Inf. Qual.* 13, 1 (2021), 1:1–1:17.

[33] Leonid Libkin. 2003. Expressive power of SQL. *Theor. Comput. Sci.* 296, 3 (2003), 379–404.

[34] Vladimir Lifschitz. 2019. *Answer Set Programming*. Springer.

[35] Boris Motik, Bernardo Cuenca Grau, Ian Horrocks, Zhe Wu, Achille Fokoue, and Carsten Lutz. 2012. *OWL 2 Web Ontology Language Profiles*. W3C Recommendation. Available at http://www.w3.org/TR/owl2-profiles/.

[36] H. B. Newcombe, J. M. Kennedy, S. J. Axford, and A. P. James. 1959. Automatic Linkage of Vital Records. *Science* 130, 3381 (1959), 954–959.

[37] Hao Nie, Xianpei Han, Ben He, Le Sun, Bo Chen, Wei Zhang, Suhui Wu, and Hao Kong. 2019. Deep Sequence-to-Sequence Entity Matching for Heterogeneous Entity Resolution. In *Proc. of the Twenty-Eighth ACM International Conference on Information and Knowledge Management (CIKM 2019)*. 629–638.

[38] Christos H. Papadimitriou and Mihalis Yannakakis. 1984. The Complexity of Facets (and Some Facets of Complexity). *J. Comput. Syst. Sci.* 28, 2 (1984), 244–259.

[39] Parag Singla and Pedro M. Domingos. 2006. Entity Resolution with Markov Logic. In *Proc. of the Sixth IEEE International Conference on Data Mining (ICDM 2006)*. 572–582.

[40] Larry J. Stockmeyer. 1976. The Polynomial-Time Hierarchy. *Theor. Comput. Sci.* 3, 1 (1976), 1–22.

[41] Moshe Y. Vardi. 1982. The Complexity of Relational Query Languages (Extended Abstract). In *Proc. of the Fourteenth ACM Symposium on Theory of Computing (STOC 1982)*. 137–146.

A PROOF DETAILS FOR SECTION 3

Proposition 1. *Let $\Sigma = \langle \Gamma_h \cup \Gamma_s, \Delta \rangle$ be an ER specification over $\mathcal{S}$, and let $\Sigma' = \langle \Gamma'_s, \Delta' \rangle$ be the ER specification with $\Gamma'_s = \Gamma_s \cup \{\sigma_\rho \mid \rho \in \Gamma_h\}$ and $\Delta' = \Delta \cup \{\delta_\rho \mid \rho \in \Gamma_h\}$. Then Σ and Σ' are equivalent in the following sense: $\mathsf{Sol}(D, \Sigma) = \mathsf{Sol}(D, \Sigma')$ for each $\mathcal{S}$-database D.*

Proof Sketch. To prove the claim, note that it is enough to show the following: given two ER specifications $\Sigma = \langle \Gamma, \Delta \rangle$ and $\Sigma' = \langle \Gamma', \Delta' \rangle$ such that $\Gamma'_h = \Gamma_h \setminus \{\rho\}$, $\Gamma_s = \Gamma'_s \cup \{\sigma_\rho\}$, and $\Delta' = \Delta \cup \{\delta_\rho\}$, where $\rho \in \Gamma_h$, we have that Σ and Σ' are equivalent.

Consider any database D. We shall show that $\mathsf{Sol}(D, \Sigma) = \mathsf{Sol}(D, \Sigma')$. First note that (D, Σ) and (D, Σ') have the same set of candidate solutions, since the notion of 'active' rule does not distinguish between hard rules and soft rules.

Let $E \in \mathsf{Sol}(D, \Sigma)$, i.e. E is a candidate solution for (D, Σ) satisfying $(D, E) \models \Gamma_h$ and $(D, E) \models \Delta$. From $(D, E) \models \rho$, we obtain $D_E \models \delta_\rho$. Since $D_E \models \delta$ for each $\delta \in \Delta$, this yields $(D, E) \models \Delta'$ as well. Therefore, E is a solution for (D, Σ') (condition $(D, E) \models \Gamma'_h$ immediately follows from the facts that $(D, E) \models \Gamma_h$ and $\Gamma'_h \subseteq \Gamma_h$). So, $\mathsf{Sol}(D, \Sigma) \subseteq \mathsf{Sol}(D, \Sigma')$.

Let $E \in \mathsf{Sol}(D, \Sigma')$, i.e. E is a candidate solution for (D, Σ') 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

(D, Σ) (condition $(D, E) \models \Delta$ immediately follows from the facts that $(D, E) \models \Delta'$ and $\Delta \subseteq \Delta'$). So, $\mathsf{Sol}(D, \Sigma') \subseteq \mathsf{Sol}(D, \Sigma)$. □

B PROOF DETAILS FOR SECTION 4

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

Proof Sketch. We outlined the upper bound argument in the main paper and now give a sketch of the lower bound.

It is well known that the satisfiability and entailment problems for propositional Horn 3CNF formulas are P-hard. For our reduction, it is more convenient to work with a variant, Horn-All, which takes as input a Horn formula $\phi = \lambda_1 \wedge \ldots \wedge \lambda_m$, with variables $V = \{v_1, \ldots, v_n\}$ and with each λ_i taking the form $v_j \wedge v_k \rightarrow v_h$ or $\top \wedge \top \rightarrow v_h$, and decides whether it is the case that $\phi \models v_1 \wedge \ldots \wedge v_n$. It can be shown that this variant remains P-hard.

We shall consider the schema $\mathcal{S} = \{R/4\}$ and use R to store the input Horn clauses, with each clause represented twice, using two copies of the head variable. The fixed specification $\Sigma_{\text{Horn-All}}$ will consist of a single hard rule:

$$\rho = \exists z_\ell, z_1, z_2. R(z_\ell, z_1, z_2, x) \wedge R(z_\ell, z_1, z_2, y) \Rightarrow \mathsf{EQ}(x, y),$$

which will be used to merge a propositional variable x with its copy y whenever both variables z_1 and z_2 have been already merged with their respective copies.

Now take some Horn-All instance $\phi = \lambda_1 \wedge \ldots \wedge \lambda_m$ of the form described earlier. We construct an $\mathcal{S}$-database D^ϕ as follows:

- for each $\lambda_i = \top \wedge \top \rightarrow v_h$, we include the facts $R(\ell_i, t, t, v_h)$ and $R(\ell_i, t, t, v'_h)$, which will force v_h and v'_h to be merged due to ρ;
- for each $\lambda_i = v_j \wedge v_k \rightarrow v_h$, we include the facts $R(\ell_i, v_j, v_k, v_h)$ and $R(\ell_i, v'_j, v'_k, v'_h)$, 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

$$\begin{aligned} E_V = &\{(t,t)\} \cup \{(l_i, l_i) \mid 1 \le i \le m\} \cup \\ &\{(v_j, v_j), (v'_j, v'_j), (v_j, v'_j), (v'_j, v_j) \mid 1 \le j \le n\} \end{aligned}$$

Clearly, D^ϕ and E_V can be constructed in LogSpace in ϕ. It can be shown that $\phi \models v_1 \wedge \ldots \wedge v_n$ iff $E_V \in \mathsf{Sol}(D^\phi, \Sigma_{\text{Horn-All}})$. □

Theorem 2. *Existence is NP-complete.*

Proof. We provide further details for the sketched reduction. Recall that the specification $\Sigma_{3\text{SAT}}$ contains three soft rules:

- $\sigma_1 = V(x) \wedge Q(y) \wedge FV(x) \dashrightarrow \mathsf{EQ}(x, y)$,
- $\sigma_2 = \exists x_p. V(x) \wedge Q(y) \wedge Prec(x_p, x) \wedge Q(x_p) \dashrightarrow \mathsf{EQ}(x, y)$,
- $\sigma_3 = \exists z. C_1(x) \wedge C_2(y) \wedge Q(z) \wedge LV(z) \dashrightarrow \mathsf{EQ}(x, y)$,

and its set Δ contains ten denial constraints, which are:

- $\delta_1 = \forall y_1, y_2. \neg(C_1(y_1) \wedge C_2(y_2) \wedge y_1 \neq y_2)$,
- $\delta_2 = \forall y. \neg(F(y) \wedge T(y))$,

and eight additional constraints $\delta_3, \ldots, \delta_{10}$, one for each relation $R_{s_{i,1}s_{i,2},s_{i,3}}$. For example, $\forall y_1, y_2, y_3. \neg(R_{tft}(y_1, y_2, y_3) \wedge F(y_1) \wedge T(y_2) \wedge F(y_3))$ is the constraint for R_{tft}, 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_{tft}-fact). To complete the proof, we need to show that ϕ is satisfiable iff $\mathsf{Sol}(D^\phi, \Sigma_{3\text{SAT}}) \neq \emptyset$.

First suppose that $\phi = c_1 \wedge \ldots \wedge c_m$ 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 ϕ. Set $E = \mathsf{EqRel}(\{(x_1, b_1), \ldots, (x_n, b_n), (c_1, c_2)\}, D^\phi)$. It is straightforward to verify that E is a candidate solution for $(D^\phi, \Sigma_{3\text{SAT}})$ and that $(D, E) \models \Delta$. Thus, $E \in \mathsf{Sol}(D^\phi, \Sigma_{3\text{SAT}})$.

Conversely, suppose that $E \in \mathsf{Sol}(D^\phi, \Sigma_{3\text{SAT}}) \neq \emptyset$. It follows that $D^\phi_E \models \delta_1$, which means that $(c_1, c_2) \in E$. By examining the available rules, only σ_3 can be used to add (c_1, c_2). In turn, σ_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 σ_2, 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, $D^\phi_E \models \delta_2$, 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 $D^\phi_E \models \delta_i$ for $i \in [3, 10]$, the truth assignment that assigns 0 to x_i if $(x_i, 0) \in E$ and 1 to x_i if $(x_i, 1) \in E$ cannot violate any of the clauses of ϕ, i.e. ϕ is satisfiable. □

As stated in the body of the paper, all of 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 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 $R_{s_{i,1}s_{i,2},s_{i,3}}$ 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 $\phi = c_1 \wedge \ldots \wedge c_m$ over variables $x_1, \ldots, x_n$, we consider the following database D^ϕ_{FD}:

$$\begin{aligned} &\{V(x_i) \mid 1 \le i \le n\} \cup \{FV(x_1), LV(x_n)\} \cup \{Prec(x_i, x_{i+1}) \mid 1 \le i < n\} \\ &\cup \{C_1(c, c_1), C_2(c, c_2), FT(0, c_F), FT(1, c_T), Q(0), Q(1)\} \\ &\cup \{R_{fff}(1,1,1,c'_R), R_{fft}(1,1,0,c'_R), R_{ftf}(1,0,1,c'_R), R_{ftt}(1,0,0,c'_R), \\ &\quad R_{tff}(0,1,1,c'_R), R_{tft}(0,1,0,c'_R), R_{ttf}(0,0,1,c'_R), R_{ttt}(0,0,0,c'_R)\} \\ &\cup \{R_{s_{i,1}s_{i,2}s_{i,3}}(x_{i,1}, x_{i,2}, x_{i,3}, c_R) \mid 1 \le i \le m\} \end{aligned}$$

where the polarities $s_{i,1}, s_{i,2}, s_{i,3}$ of literals in a clause are defined as in Theorem 2.

The specification $\Sigma^{FD}_{3\text{SAT}}$ will contain the soft rules σ_1 and σ_2, as well as the modified soft rule

$$\sigma'_3 = \exists z, z'. C(z, x) \wedge C(z, y) \wedge Q(z') \wedge LV(z') \dashrightarrow \mathsf{EQ}(x, y)$$

The denial constraints (all corresponding to FDs) are as follows:

- $\delta_C = \forall x, y_1, y_2. \neg(C(x, y_1) \wedge C(x, y_2) \wedge y_1 \neq y_2)$
- $\delta_{FT} = \forall x, y_1, y_2. \neg(FT(x, y_1) \wedge FT(x, y_2) \wedge y_1 \neq y_2)$
- $\delta_\tau = \forall x_1, x_2, x_3, y_1, y_2. \neg(R_I(x_1, x_2, x_3, y_1) \wedge R_I(x_1, x_2, x_3, y_2) \wedge y_1 \neq y_2)$, for each possible $\tau \in \{fff, fft, ftf, ftt, tff, tft, ttf, ttt\}$.

Intuitively, δ_C forces c_1 and c_2 to merge and thus replaces δ_1 from Theorem 2, whereas δ_{FT} replaces δ_2 and serves to forbid 0 and 1 merging (so that each variable receives a unique truth value). The constraints δ_τ 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 $R_{tft}(x_j, x_k, x_\ell, c_R)$ (representing the clause $x_j \vee \neg x_k \vee x_\ell$) and we merge x_j with 0, x_k with 1, and x_ℓ with 0 (thus falsifying the clause), then δ_{tft} will be violated due to fact $R_{tft}(0, 1, 0, c'_R)$.

Following an argument similar to the one used for Theorem 2, it can be shown that ϕ is satisfiable iff $\mathsf{Sol}(D^\phi_{FD}, \Sigma^{FD}_{3SAT}) \neq \emptyset$. ◻

Theorem 3. *MaxRec is coNP-complete.*

Proof Sketch. For the upper bound, we can decide $E \notin \mathsf{MaxSol}(D, \Sigma)$ in NP by first guessing E' and then checking that either (*i*) $E \notin \mathsf{Sol}(D, \Sigma)$ or (*ii*) $E \subsetneq E'$ and $E' \in \mathsf{Sol}(D, \Sigma)$ 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 3SAT instance ϕ can be merged with either 0 or 1 only if c and c' have been previously merged.

Recall the database D^ϕ encoding the 3SAT instance ϕ as in the proof of Theorem 2, and let $D^\phi_C = D^\phi \cup \{C(c), C'(c')\}$. The ER specification Σ'_{3SAT} is obtained from Σ_{3SAT} (from the proof of Theorem 2) by (*i*) replacing $V(x) \wedge Q(y) \wedge FV(x) \dashrightarrow \mathsf{EQ}(x, y)$ with $\exists z. V(x) \wedge Q(y) \wedge FV(x) \wedge C(z) \wedge C'(z) \dashrightarrow \mathsf{EQ}(x, y)$, (*ii*) including soft rule $C(x) \wedge C'(y) \dashrightarrow \mathsf{EQ}(x, y)$, and (*iii*) replacing $\forall y_1, y_2. \neg(C_1(y_1) \wedge C_2(y_2) \wedge y_1 \neq y_2)$ with $\forall y, y_1, y_2. \neg(C(y) \wedge C'(y) \wedge C_1(y_1) \wedge C_2(y_2) \wedge y_1 \neq y_2)$, which require c_1 and c_2 to be merged only if c and c' already merged. Finally, we let $E = \mathsf{EqRel}(\emptyset, D^\phi_C)$. With the correctness of the reduction provided in the proof of Theorem 2 at hand, it is not hard to see that ϕ is unsatisfiable if and only if $E \in \mathsf{MaxSol}(D^\phi_C, \Sigma'_{3SAT})$. ◻

Theorem 6. *CertAnswer is Π^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 $q = \exists z. C(z) \wedge C'(z)$ 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 $q = \exists z. C_1(z) \wedge C_2(z)$ 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 $E \in \mathsf{Sol}(D, \Sigma)$ using the technique illustrated in the proof sketch of Theorem 1. If $E \notin \mathsf{Sol}(D, \Sigma)$, then we return `false`; otherwise, we continue as follows. We collect in set S all those pairs of constants $\alpha = (c, c')$ such that α is active in (D, E) w.r.t. Γ_s and $\alpha \notin E$. For each $\alpha \in S$, we proceed as follows. Starting from $E' := \mathsf{EqRel}(E \cup \{\alpha\}, D)$, 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. Γ_h and $(c, c') \notin E'$, then set $E' := \mathsf{EqRel}(E' \cup \{(c, c')\}, D)$. 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 $E \in \mathsf{Sol}(D, \Sigma)$, to establish $E \notin \mathsf{MaxSol}(D, \Sigma)$ it is enough to 'minimally' extend E and see whether such a minimal extension leads to a solution for (D, Σ). This is because, if Δ 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 Σ: $E \in \mathsf{Sol}(D, \Sigma)$ iff $E = \{(a, b) \mid Eq(a, b) \in M\}$ for some stable model M of (Π_{Sol}, D). In particular, $\mathsf{Sol}(D, \Sigma) \neq \emptyset$ iff (Π_{Sol}, D) is coherent.*

Proof Sketch. Given $E \in \mathsf{Sol}(D, \Sigma)$, let M_E extend D with the following facts: $Adom(d)$ for every $d \in \mathsf{dom}(D)$, $Eq(a, b)$ for every $(a, b) \in E$, $Active(a, b)$ for every pair (a, b) that is active in (D, E) due to a soft rule, and $Neq(a, b)$ if $Active(a, b) \in M_E$ but $Eq(a, b) \notin M_E$. We claim that M_E is a stable model, i.e. the unique minimal model of $reduct(gr(\Pi_{Sol}, D))$. We describe a key part of the argument, which is to show that every fact in M_E is entailed from $reduct(gr(\Pi_{Sol}, D))$, focusing on Eq-facts. We fix a sequence $E_0, \alpha_0, E_1, \ldots, \alpha_n, E_n$ such that $E_0 = \mathsf{EqRel}(\emptyset, D)$, and for every $1 \leq i < n$, $E_{i+1} = \mathsf{EqRel}(E_i \cup \{\alpha_i\}, D)$ for some $\alpha_i \notin E_i$ that is active in (D, E_i). By suitably enumerating the pairs in E_0 and each $E_{i+1} \setminus (E_i \cup \{\alpha_i\})$, we obtain an enumeration of the Eq-facts in M_E:

$$Eq(\epsilon^0_1), \ldots, Eq(\epsilon^0_{\ell_0}), Eq(\alpha_1), Eq(\epsilon^1_1), \ldots, Eq(\epsilon^1_{\ell_1}), Eq(\alpha_2), \ldots, Eq(\alpha_n), Eq(\epsilon^n_1), \ldots, Eq(\epsilon^n_{\ell_n})$$

such that each fact can be derived from D and the preceding facts in the enumeration using the ground rules in $reduct(gr(\Pi_{Sol}, D))$. Intuitively, each $Eq(\epsilon^0_k)$ is obtained using groundings of $Eq(x, x) \leftarrow Adom(x)$ and the $Adom$ rules, each $Eq(\epsilon^j_k)$ (with $j > 0$) by applying the (instantiated) symmetry or transitivity rule, each $Eq(\alpha_i)$ with α_i added to E_i due to $\rho = q(x, y) \Rightarrow \mathsf{EQ}(x, y)$ by the (instantiation of the) rule $Eq(\alpha_i) \leftarrow q^+(\alpha_i)$, and each $Eq(\alpha_i)$ with α_i added due to $\sigma = q(x, y) \dashrightarrow \mathsf{EQ}(x, y)$ by the combination of (instantiations of) $Active(\alpha_i) \leftarrow q^+(\alpha_i)$ and $Eq(\alpha_i) \leftarrow Active(\alpha_i)$ (note that 'not $Neq(\alpha_i)$' is dropped in the reduct since $Neq(\alpha_i) \notin M_E$).

Conversely, suppose that M is a stable model of (Π_{Sol}, D), and let $E_M = \{(a, b) \mid Eq(a, b) \in M\}$. 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 $r_i \in reduct(gr(\Pi_{Sol}, D))$ whose head is β_i and whose body consists of facts from $D \cup \{\beta_1, \ldots, \beta_{i-1}\}$. We may assume w.l.o.g. that the enumeration respects the following strategy: (i) first apply all instantiations of rules of the form $Adom(x_i) \leftarrow P(x_1, \ldots, x_n)$ (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 Eq-facts are suitably ordered so as to be grouped into a sequence $E_0, \alpha_0, E_1, \ldots, \alpha_n, E_n$ such that $E_0 = \mathsf{EqRel}(\emptyset, D)$, and for every $1 \leq i < n$, $E_{i+1} = \mathsf{EqRel}(E_i \cup \{\alpha_i\}, D)$ for some α_i active in (D, E_i). As M is a model of (Π_{Sol}, D), and Π_{Sol} contains analogs of the hard rules and denial constraints from Σ, there cannot be any unsatisfied hard rule nor violated constraint in D_{E_M}, i.e. E_M is a solution. $\square$

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) $sg(x,x) \leftarrow V(x)$; (2) $sg(x,y) \leftarrow E(z,x) \wedge E(z',y) \wedge sg(z,z')$, and (3) $sg(x,y) \leftarrow sg(x,z) \wedge sg(z,y)$.

We first detail our claim that the ER specification $\Sigma_{sg} = \langle \{\exists z. E(z,x) \wedge E(z,y) \dashrightarrow \mathsf{EQ}(x,y)\}, \emptyset \rangle$ *expresses the sg property over digraphs*, i.e. $\mathsf{certMerge}(D_G, \Sigma_{sg}) = \{(v,v') \mid (v,v') \text{ is } sg \text{ in } G\}$ for every $\mathcal{S}_G$-database D_G representing a digraph G.

Proposition 2. *Σ_{sg} expresses the sg property over digraphs.*

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

We now formally define the class of *dgbc* graphs. Given $n, m \geq 0$, we define the digraph $G_n^m = (V, E)$ as follows:

- if $n = 0$: $V = \{u_1, \dots, u_m\}$ and $E = \emptyset$;
- if $n > 1$: $V = \{g, g', v_1, v_1', \dots, v_n, v_n', u_1, \dots, u_m\}$ and $E = \{(g,g'), (g',g)\} \cup C \cup C'$ with $C = \{(g,v_1), (v_1,v_2), \dots, (v_{n-1}, v_n)\}$ and $C' = \{(g,v_1'), (v_1',v_2'), \dots, (v_{n-1}', v_n')\}$.

Intuitively, G_n^m contains m isolated nodes and, if $n \geq 1$, two length-n chains originating from g and a g, g'-loop. By *directed bidirectional chain graph (dgbc)*, we shall mean a digraph G_n^m (for some $n, m \geq 0$).

It can be easily verified that the pairs of *sg* nodes in G_n^m are:

$$q_{sg}(D_{G_n^m}) = \{(v,v) \mid v \in V\} \cup \{(v_i, v_i'), (v_i', v_i) \mid i \leq n\}.$$

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

Now let us consider the EL framework. We shall say that an EL specification $\mathcal{H} = \langle \{\mathsf{L}\}, \mathcal{S}_G, \Omega \rangle$ *expresses the sg property over dgbc graphs*, if for every $\mathcal{S}_G$-database D_G representing a *dgbc* graph G, the extension of 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 *dgbc* graphs using the EL specification $\mathcal{H}^* = \langle \{\mathsf{L}\}, \mathcal{S}_G, \Omega \rangle$, where Ω 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:

$$\mathsf{L}(x,y) \rightarrow V(x) \wedge V(y) \wedge x = y \vee \exists z, z'. E(z,x) \wedge E(z',y) \wedge \mathsf{L}(z,z')$$

We can prove that this is not the case. Since there are no FDs over L, for every $\mathcal{S}_G$-database D, the set of certain links for D w.r.t. $\mathcal{H}^*$ coincide with the unique $\subseteq$-maximal $\{\mathsf{L}\}$-database J such that $(D, J) \models \Omega$. Consider the $\mathcal{S}_G$-database $D_{G_1^0} = \{V(g), V(g'), V(v_1), V(v_1'), E(g,g'), E(g',g), E(g,v_1), E(g,v_1')\}$ representing G_1^0, and let J be the $\subseteq$-maximal $\{\mathsf{L}\}$-database J such that $(D_{G_1^0}, J) \models \Omega$. We claim that $\mathsf{L}(g,g') \in J$ and $\mathsf{L}(g',g) \in J$. Since neither pair is *sg* in G_1^0, it follows that $\mathcal{H}^*$ does not express the *sg* property over *dgbc* graphs. Let us thus suppose for a contradiction that either $\mathsf{L}(g,g') \notin J$ or $\mathsf{L}(g',g) \notin J$, and consider $J' = J \cup \{\mathsf{L}(g,g'), \mathsf{L}(g',g)\}$. Observe that the instantiated MCs $\mathsf{L}(g,g') \rightarrow \exists z, z'. E(z,g) \wedge E(z',g') \wedge \mathsf{L}(z,z')$ and $\mathsf{L}(g',g) \rightarrow \exists z, z'. E(z,g') \wedge E(z',g) \wedge \mathsf{L}(z,z')$ are satisfied in $D_{G_1^0} \cup J'$. When combined with $(D_{G_1^0}, J) \models \Omega$, this yields $(D_{G_1^0}, J') \models \Omega$. Thus, J' is a solution for $D_{G_1^0}$ w.r.t. $\mathcal{H}^*$ with $J \subsetneq J'$, contradicting the maximality of J.

We now provide the proof sketch of Theorem 11.

Theorem 11. *There is no entity-linking specification $\mathcal{H} = \langle \{\mathsf{L}\}, \mathcal{S}_G, \Omega \rangle$ in $\mathcal{L}_2$ that expresses the sg property over dgbc graphs, i.e. such that, for every $\mathcal{S}_G$-database D_G representing a dgbc graph G, $\mathsf{L}(a,b)$ is a certain link iff (a,b) is a pair of sg nodes in G.*

Proof Sketch. Suppose that the EL specification $\mathcal{H} = \langle \{\mathsf{L}\}, \mathcal{S}_G, \Omega \rangle$ (in the most expressive $\mathcal{L}_2$ dialect of [13]) expresses the *sg* property over *dgbc* graphs, and its unique MC for L is

$$\omega = \mathsf{L}(x,y) \rightarrow \forall \vec{u}. (\psi(x,y,\vec{u}) \rightarrow \alpha_1 \vee \ldots \vee \alpha_k),$$

where $\psi(x,y,\vec{u})$ is a (possible empty) conjunction of relational atoms over $\mathcal{S}_G$, and each disjunct α_i is a conjunctive query over $\mathcal{S}_G \cup \{\mathsf{L}\}$, 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 $\mathsf{L}(X) \subseteq V(A)$ and $\mathsf{L}(Y) \subseteq V(A)$, with (X,Y) and A the attributes of L and V, respectively.
(3) We may assume w.l.o.g. that $\omega = \mathsf{L}(x,y) \rightarrow \alpha_1 \vee \ldots \vee \alpha_k$.
(4) No specification satisfying the preceding restrictions can express the *sg* property over *dgbc* graphs.

For points (1) and (2), consider the $\mathcal{S}_G$-database $D_{G_1^1} = \{V(g), V(g'), V(v_1), V(v_1'), V(u_1), E(g,g'), E(g',g), E(g,v_1), E(g,v_1')\}$. As (v_1,v_1) and (v_1,v_1') (resp., (v_1',v_1') and (v_1',v_1)) are pairs of *sg* nodes in G_1^1, they must belong to the extension of L, hence we cannot have the FD $\mathsf{L}\colon \{X\} \rightarrow Y$ (resp., $\mathsf{L}\colon \{Y\} \rightarrow X$). Further note that (u_1,u_1) is a pair of *sg* nodes in G_1^1. Thus, $\mathcal{H}$ must have the inclusion dependencies $\mathsf{L}(X) \subseteq V(A)$ and $\mathsf{L}(Y) \subseteq V(A)$, otherwise no solution for $D_{G_1^1}$ w.r.t. $\mathcal{H}$ can contain $\mathsf{L}(u_1,u_1)$, 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_n^m where m and n are not bounded by a fixed constant and depend instead on the number of variables occurring in ω. $\square$