Core Schema Mappings: Scalable Core Computations in Data Exchange*

Giansalvatore Mecca[1], Paolo Papotti[2], Salvatore Raunich[3]

[1] Università della Basilicata – Potenza, Italy
[2] Università Roma Tre – Roma, Italy
[3] University of Leipzig – Leipzig, Germany

Spicy Working Report # 01-2011 - Last update: February 28, 2011

Abstract

Research has investigated mappings among data sources under two perspectives. On one side, there are studies of practical tools for schema mapping generation; these focus on algorithms to generate mappings based on visual specifications provided by users. On the other side, we have theoretical researches about data exchange. These study how to generate a solution – i.e., a target instance – given a set of mappings usually specified as tuple generating dependencies. Since the notion of a core solution has been formally identified as an optimal solution, it is very important to efficiently support core computations in mapping systems. In this paper we introduce several new algorithms that contribute to bridge the gap between the practice of mapping generation and the theory of data exchange. We show how, given a mapping scenario, it is possible to generate an executable script that computes core solutions for the corresponding data exchange problem. The algorithms have been implemented and tested using common runtime engines to show that they guarantee very good performances, orders of magnitudes better than those of known algorithms that compute the core as a post-processing step.

1 Introduction

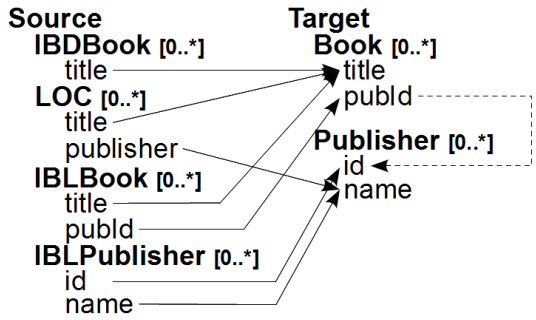
Integrating data coming from disparate sources is a crucial task in many applications. An essential requirement of any data integration task is that of manipulating *mappings* between sources. Mappings are executable transformations – say, SQL or XQuery scripts – that specify how an instance of the source repository should be translated into an instance of the target repository. We may identify two broad research lines in the literature.

On one side, we have studies on practical tools and algorithms for *schema mapping generation*. In this case, the focus is on the development of systems that take as input an abstract specification of the mapping, usually made of a bunch of correspondences between the two schemas, and generate the mappings and the executable scripts needed to perform the translation. This research topic was largely inspired by the seminal papers about the Clio system [26, 27]. The original algorithm has been subsequently extended in several ways [17, 5, 2, 29, 7] and various tools have been proposed to support users in the mapping generation process. More recently, a benchmark has been developed [1] to compare research mapping systems and commercial ones.

On the other side, we have theoretical studies about *data exchange*. Several years after the development of the initial Clio algorithm, researchers have realized that a more solid theoretical foundation was needed in order to consolidate the practical results obtained on schema mapping systems. This consideration has motivated a rich body of research in which

*Portions of this paper have appeared under the title *Core Schema Mappings* in the Proceedings of the ACM SIGMOD 2009 Conference.

the notion of a *data exchange problem* [12] was formalized, and a number of theoretical results were established. In this context, a *data exchange setting* is a collection of mappings – usually specified as *tuple generating dependencies* (*tgds*) [4] – that are given as part of the input; therefore, the focus is not on the generation of the mappings, but rather on the characterization of their properties. This has brought to an elegant formalization of the notion of a solution for a data exchange problem, and of operators that manipulate mappings in order, for example, to compose [14] or invert [11, 3] them.



IBDBook

title
The Hobbit
The Da Vinci Code
The Lord of the Rings

LOC

title	publisher
The Lord of the Rings	Houghton
The Catcher in the Rye	Lb Books

IBLBook

title	pubId
The Hobbit	245
The Catcher in the Rye	776

IBLPublisher

id	name
245	Ballantine
776	Lb Books

Target: Book

title	pubId
The Hobbit	N1
The Da Vinci Code	N2
The Lord of the Rings	N3
The Lord of the Rings	I1
The Catcher in the Rye	I2
The Hobbit	245
The Catcher in the Rye	776

Target: Publisher

id	name
245	Ballantine
776	Lb Books
I1	Houghton
I2	Lb Books

Figure 1: Mapping Bibliographic References

1.1 Motivation

There are many possible solutions for a data exchange problem. A natural question is the following: "which solution should be materialized by a mapping system?" A key contribution of data exchange research was the formalization of the notion of *core* [13] universal solution, which was identified as the "optimal" solution. Informally speaking, the core universal solution has a number of nice properties: it is "irredundant", since it is the smallest among the solutions that preserve the semantics of the exchange, and it represents a "good" instance for answering conjunctive queries over the target database. It can therefore be considered a natural requirement for a schema mapping system to generate executable scripts that materialize core solutions.

Example 1.1 Consider the mapping scenario informally described in Figure 1, where also a source instance is shown. The source database contains tables about books coming from three different data sources, namely the *Internet Book Database (IBD)*, the *Library of Congress* database *(LOC)*, and the *Internet Book List (IBL)*.

The desired mapping can be expressed using the following set of *tuple-generating dependencies (tgds)*:.

$$m_1.\ \forall t, p\colon LOC(t,p) \rightarrow \exists I\colon Book(t,I) \wedge Publisher(I,p)$$
$$m_2.\ \forall t, id\colon IBLBook(t,id) \rightarrow Book(t,id)$$
$$m_3.\ \forall id, p\colon IBLPublisher(id,p) \rightarrow Publisher(id,p)$$
$$m_4.\ \forall t\colon IBDBook(t) \rightarrow \exists N\colon Book(t,N)$$

It can be seen how each source has a slightly different organization wrt the others. In particular, the *IBD* source contains data about book titles only; mapping m_4 copies titles to the *Book* table in the target. The *LOC* source contains book titles and publisher names in a single table; these are copied to the target tables by mapping m_1, which also "invents" a value to correlate the key and the foreign key. Finally, the *IBL* source contains data about books and their publishers in separate tables; these data are copied to the target by mappings m_2, m_3; note that in this case we do not need to invent any values.

These expressions materialize the target instance in Figure 1, called a *canonical universal instance*. While this instance satisfies the tgds, still it contains many redundant tuples, those with a gray background. Consider for example the tuple t_1 = *(The Hobbit,* N_1*)*; it can be seen that the tuple is redundant since the target contains another tuple t_2 = *(The Hobbit, 245)* for the same book, which in addition to the title also gives information about the publisher. The fact that t_1 is redundant with respect to t_2 can be formalized by saying that there is an *homomorphism* from t_1 to t_2. A *homomorphism*, in this context, is a mapping of values that transforms t_1 into t_2. A similar argument holds for the tuple *(The Lord of the Rings,* N_3*)*, and for tuples *(The Catcher in the Rye, I2)* and *(I2, Lb Books)*, where *I2* is the value invented by executing tgd m_1. The presence of such homomorphisms means that the solution in Figure 1 has an *endomorphism*, i.e., a homomorphism into a sub-instance – the one obtained by removing all redundant tuples.

The fact that tgds produced by a schema mapping algorithm may generate redundancy in the target is well known and has motivated several practical proposals (e.g. [17]) towards the goal of removing such redundant data. Unfortunately, these proposals are applicable only in some cases and do not represent a general solution to the problem. In [13] the notion of *core universal solution* has been introduced as a "more desirable" solution than the one in Figure 1. The *core* is the smallest among the solutions for a given source instance that has homomorphisms into all other solutions. The core of the solution in Figure 1 is in fact the portion of the target tables with a white background.

◇

It can be seen how it is crucial to develop algorithms that natively produce executable scripts to compute the core. On the contrary, schema mapping systems typically generate canonical universal solutions, which may contain quite a lot of redundancy. This is partly due to the fact that computing cores is a challenging task.

A possible approach to the generation of core solutions for a relational data exchange problem is the following: (*i*) first, to generate a canonical solution by chasing the source-to-target tgds; to do this, a mapping system typically generates an SQL or XQuery script that performs this step very efficiently, even on large source instances; (*ii*) then, to apply a post-processing algorithm for core identification.

Several polynomial algorithms have been identified to this end [13, 18]. These algorithms provide a very general answer to the problem of computing core solutions for a data exchange setting. Also, an implementation of the core-computation algorithm in [18] has been developed [30] by using a combination of SQL for database access and a controlled form of recursive control-logic implemented in Java.

Although polynomial, experience with these algorithms shows that they hardly scale to large mapping scenarios. In fact, they exhaustively look for endomorphisms inside the canonical universal solution in order to identify which null values and which tuples can be removed. This kind of computation can take very high computing times, even on databases of a few thousand tuples, as shown in our experiments.

This paper makes several important contributions towards the goal of making the computation of core solutions a scalable functionality of mapping systems. More specifically:

- given a mapping scenario consisting of source-to-target tgds, we introduce a rewriting algorithm that generates a new set of dependencies that can be used to generate core solutions for the original tgds; these dependencies can be translated into an SQL script and ran inside any conventional database engine, thus achieving a very high degree of flexibility and performance;
- the algorithm has been implemented into the +Spicy mapping system; in the paper, we conduct an experimental evaluation on large mapping scenario that confirms the scalability of our solution;

- the rewriting algorithm is based upon a new characterization of the core, in terms of *expansions*; we introduce the notion of expansion and show how they represent a natural tool for the rewriting of the given tgds.

The algorithms developed in this paper concentrate on mapping scenarios made of source-to-target tgds only. However, how it will be discussed in Section 2.1, they represent an essential building block for more general algorithms that handle larger classes of constraints.

2 Overview

The main intuition behind our approach is that it is much more efficient to prevent the generation of redundant tuples in a solution, than trying to remove them after they have been generated by the chase. Following this intuition, given a set of s-t tgds Σ_{st}, our goal is to rewrite them as a new set of dependencies Σ'_{st}, such that, for any source instance I, chasing Σ'_{st} yields the core universal solution for Σ_{st} and I.

To do this, we need to analyze the given tgds in order to recognize when one of them may generate redundant tuples in the target. A key concept in order to do this is the notion of a *witness block*. Intuitively, a *witness block* is a set of facts that guarantee that a tgd is satisfied for some assignment of constants to the universal variables.

Example 2.1 Consider the tgds in Example 1.1. To simplify the notation, let us rename the source relations as A, B, C, D, and the target relations as S, T. In order to discuss our approach, as it will be detailed in the following, we find it useful to label atoms in tgd conclusions in order to properly distinguish them.

$$m_1.\ \forall x_1, x_2 : A(x_1, x_2) \rightarrow \exists Y_1 : S^1(x_1, Y_1) \wedge T^2(Y_1, x_2)$$
$$m_2.\ \forall x_3, x_4 : B(x_3, x_4) \rightarrow S^3(x_3, x_4)$$
$$m_3.\ \forall x_5, x_6 : C(x_5, x_6) \rightarrow T^4(x_5, x_6)$$
$$m_4.\ \forall x_7 : D(x_7) \rightarrow \exists Y_0 : S^5(x_7, Y_0)$$

Consider now the source instance $I = \{A(1,2), B(1,3), C(3,2), D(1)\}$, and the canonical universal solution: $J = \{S(1,N_0), T(N_0,2), S(1,3), T(3,2), S(1,N_1)\}$.

Let us restrict our attention to tgd m_1. The premise of m_1 is satisfied by the source atom $A(1,2)$ in I. We call the set of atoms $w = \{S(1,N_0), T(N_0,2)\}$ in J a *witness block* for m_1 and $A(1,2)$, since the atoms in w guarantee that I and J satisfy m_1.

However, for the same tgd and source atom there can be many alternative witness blocks in a solution. Consider for example atoms $w' = \{S(1,3), T(3,2)\}$: the two atoms taken together are also a witness block for m_1 and $A(1,2)$.

Notice how, for core computation purposes, w' is "preferable" with respect to w. In fact, w' contains less nulls than w. This can be formalized by saying that there is a homomorphism of w into w', as discussed above, and therefore that the atoms in w are made redundant by those in w'.

Witness blocks are at the foundations of our algorithm. We want to emphasize that other algorithms for core computation [13, 18, 31] rely on the contrary on the notion of *fact blocks*. Witness blocks are different from fact blocks, as it will be discussed in Section 5.

Notice however that witness blocks are sets of facts; in order to perform the rewriting, we need to reason about formulas; however, an important property of witness blocks is that they can be captured by a set of first-order queries on the target database called *expansions*.

Consider again tgd m_1 in our example. It has two expansions, as follows:

$$\epsilon_1 = S^1(x_1, Y_1) \wedge T^2(Y_1, x_2) \quad \textit{(the base expansion, i.e., the tgd conclusion)}$$
$$\epsilon_2 = S^3(x_3, x_4) \wedge T^4(x_5, x_6) \wedge x_4 = x_5 \wedge \exists Y_1 : (S^1(x_3, Y_1) \wedge T^2(Y_1, x_6))$$

Considered as a query over the target database, expansion ϵ_1 selects witness block w. Expansion ϵ_2 witness block w'.

Notice that, while expansions are formulas over the target database, they can be easily rewritten as formulas over the source. The source rewritings of ϵ_1, ϵ_2 are as follows:

$$\mathsf{sourceRew}\,(\epsilon_1) = A(x_1, x_2) \quad \textit{(the tgd premise)}$$
$$\mathsf{sourceRew}\,(\epsilon_2) = B(x_3, x_4) \wedge C(x_5, x_6) \wedge x_4 = x_5 \wedge (A(x_3, x_6))$$

◇

Based on these ideas, in the paper we undertake the following approach:

- we formalize the notion of a *witness block*, and introduce a partial order among witness blocks; based on this, we show that it is possible to provide a new characterization of the core of the universal solutions for a mapping scenario in terms of a set of "maximal" witness blocks;
- we formalize the notion of an *expansion* for a tgd, and provide algorithms for generating all expansions for a tgd;
- in order to generate core solutions, we need to select, among all witness blocks generated by an expansion, only the maximal ones; to do this, we introduce a notion of *formula homomorphism* among expansions; whenever, for a tgd m, we find that expansion e has a formula homomorphism into expansion e', we know that e generates witness blocks for m that might be made redundant by those generated by e'; in essence, in order to prevent the generation of homomorphisms among facts, we study homomorphisms among the formulas that generate them;
- based on formula homomorphisms, we rewrite expansions by introducing *negations*. In this way, we are able to formalize an alternative characterization of the core in terms of expansions, which represents the basis for the actual rewriting algorithm;
- given a set of s-t tgds, Σ_{st}, we rewrite the original tgds as a new set of dependencies; we call these dependencies *First-Order rules* (FO-rules), since they are strictly more expressive than ordinary tgds; more specifically, they allow for negations in the premise and *Skolem functions* [21, 27] in the conclusion. The rewritten set of rules is called a *core schema mapping*.

Consider again the scenario $\mathcal{M}$ in Example 2.1. The rewritten rules generated from the algorithm – i.e., the *core schema mapping* for $\mathcal{M}$ – are as follows:

$$
\begin{aligned}
&r_1.\ \forall x_1, x_2 : A(x_1, x_2) \wedge \neg(\exists x_4, x_5 : B(x_1, x_4) \wedge C(x_5, x_2) \wedge x_4 = x_5) \rightarrow \\
&\qquad\qquad S(x_1, f(x_1, x_2)) \wedge T(f(x_1, x_2), x_2).\\
&r_2.\ \forall x_3, x_4 : B(x_3, x_4) \rightarrow S(x_3, x_4).\\
&r_3.\ \forall x_5, x_6 : C(x_5, x_6) \rightarrow T(x_5, x_6).\\
&r_4.\ \forall x_7 : D(x_7) \wedge \neg(\exists x_2 : A(x_7, x_2)) \wedge \neg(\exists x_4 : B(x_7, x_4)) \rightarrow S(x_7, f(x_7)).
\end{aligned}
$$

If we go back to the original names in Example1.1, we obtain the following rules:

$$
\begin{aligned}
&r_1.\ \forall x_1, x_2 : \mathit{LOC}(x_1, x_2) \wedge \neg(\exists x_4, x_5 : \mathit{IBLBook}(x_1, x_4) \wedge \mathit{IBLPublisher}(x_5, x_2) \wedge x_4 = x_5)\\
&\qquad\qquad \rightarrow \mathit{Book}(x_1, f(x_1, x_2)) \wedge \mathit{Publisher}(f(x_1, x_2), x_2).\\
&r_2.\ \forall x_3, x_4 : \mathit{IBLBook}(x_3, x_4) \rightarrow \mathit{Book}(x_3, x_4).\\
&r_3.\ \forall x_5, x_6 : \mathit{IBLPublisher}(x_5, x_6) \rightarrow \mathit{Publisher}(x_5, x_6).\\
&r_4.\ \forall x_7 : \mathit{IBDBook}(x_7) \wedge \neg(\exists x_2 : \mathit{LOC}(x_7, x_2)) \wedge \neg(\exists x_4 : \mathit{IBLBook}(x_7, x_4))\\
&\qquad\qquad \rightarrow \mathit{Book}(x_7, f(x_7)).
\end{aligned}
$$

Once the original scenario $\mathcal{M}$ has been rewritten as a set of FO-rules, given a source instance I, in order to generate a core solution for $\mathcal{M}$ and I it is sufficient to chase the rules over I; this can be done very efficiently using common runtime languages like SQL (or XQuery) and guarantees very good performances, orders of magnitude better than those of previous core-computation algorithms.

In fact, we have implemented the algorithms developed in the paper as part of the +Spicy [25, 6] working prototype. Experimental results based on the use of the system show that our strategy scales up to large databases in practical scenarios.

In light of this, we believe that this paper makes a significant advancement towards the goal of bridging the gap between the practice of schema mapping systems and the theory of data exchange.

2.1 Target Constraints

Note that in this paper we restrict our attention to data exchange settings expressed as a set of source-to-target tgds only. We do not consider *target tgds* and *target egds* [4]. For this class of mappings we show that it is always possible to generate a core schema mapping.

This result is somehow optimal. In fact, with respect to target tgds, it was shown in [31] that it is in general not possible to rewrite a scenario with s-t tgds and target tgds into a *laconic mapping* [31]. The authors conjecture that the same also holds for target egds. Notice that, while very similar in spirit, the notion of a laconic mapping is not the same as the notion of a core schema-mapping that we use in this paper. In fact, a laconic mapping is required to be logically equivalent [16] to the given scenario, while we do not require this for core-schema mappings. However, the proof of [31] was given without the assumption of logical equivalence, and extends to core schema-mappings as well.

To see this, consider the following simple scenario $\mathcal{M}$ [31], with one s-t tgd and one full target tgd:

$$r_1.\ \forall x_1, x_2 : A(x_1, x_2) \rightarrow R(x_1, x_2).$$
$$r_2.\ \forall x, y, z : R(x, y) \wedge R(y, z) \rightarrow R(x, z).$$

It can be seen that, on a source instance I, the core solution J_0 contains in R the transitive closure of A. If there were a core-schema mapping for $\mathcal{M}$, i.e., a set of FO-rules to compute J_0, then it would be possible to compute the transitive closure of A using first-order logic, which is obviously a contradiction.

With respect to target egds, a similar result was recently proven in [23], where the authors show that it is not possible in general to rewrite a set of s-t tgds and target egds as an equivalent set of FO-rules.

Nevertheless, the techniques developed in this paper represent an important building block towards the goal of developing scalable core-computation techniques for large classes of scenarios that include target constraints, as follows.

Let us first consider target tgds. These are typically used in mapping applications to encode foreign-key constraints on the target schema. We may say that target tgds corresponding to foreign-key constraints have received quite a lot of attention, and are handled quite nicely by schema-mapping systems [26, 27]. The main idea is that, whenever the set of target tgds has an appropriate boundedness property, they can be rewritten into the source-to-target tgds [16]. In fact, the intuition of chasing foreign keys to generate source-to-target tgds is at the core of the original Clio mapping-generation algorithm. Once the target tgds corresponding to foreign keys have been rewritten under the form of s-t tgds, then the techniques developed in this paper can be used to generate core universal solutions.

Let us now consider target egds. Egds are typically used to encode key constraints and functional dependencies over the target. Handling key constraints is a delicate task, due to the particular form of processing that they require on the target instances – essentially equating values. However, in [23] it was shown that, for a very large fraction of cases, it is possible to rewrite a mapping scenario containing s-t tgds and target egds as a set of FO-rules, and then use these rules to compute core universal solutions for the original scenario. The results in [23] heavily rely on the algorithms developed in this paper in order to perform the rewriting, thus confirming the relevance of our contributions.

2.2 Paper Outline

The paper is organized as follows. Sections 3 and 4 provide some background. Section 5 introduces the notion of a *witness block*, and a characterization of the core in terms of witness blocks. Section 6 introduces the notion of an *expansion* and the notion of a *formula homomorphism*. Then, the alternative characterization of the core in terms of expansion is in Section 7. Section 8 shows how to rewrite expansions over the source database. Based on these results, the actual rewriting algorithm is introduced in Section 9. Skolemization strategies are discussed in Section 10. A complete example of application of the algorithm is in Section 11. A discussion on complexity is in Section 12. Experimental results are in Section 13. A discussion of related work is in Section 14.

3 Background

Data Model We fix two disjoint sets: a set of *constants*, CONSTS, a set of *labeled nulls*, VARS. We also fix a set of *labels* $\{A_0, A_1 \ldots\}$, and a set of *relation symbols* $\{R_0, R_1, \ldots\}$. With each relation symbol R we associate a *relation schema* $R(A_1, \ldots, A_k)$. A *schema* $\mathbf{S} = \{R_1, \ldots, R_n\}$ is a collection of relation schemas. An *instance* of a relation schema $R(A_1, \ldots, A_k)$ is a finite set of tuples of the form $R(A_1 : v_1, \ldots, A_k : v_k)$, where, for each i, v_i is either a constant or a labeled null. An *instance* of a schema $\mathbf{S}$ is a collection of instances, one for each relation schema in $\mathbf{S}$. In the following, we will interchangeably use the positional and non positional notation for tuples and facts; also, with an abuse of notation, we will often blur the distinction between a relation symbol and the corresponding instance.

Given an instance I, we shall denote by $consts(I)$ the set of constants occurring in I, and by $vars(I)$ the set of labeled nulls in I. Its *active domain*, denoted by $dom(I)$, is the set $consts(I) \cup vars(I)$. A *ground* instance is an instance I without labeled nulls (where $dom(I) = consts(I)$).

Given two disjoint schemas, $\mathbf{S}$ and $\mathbf{T}$, we shall denote by $\langle \mathbf{S}, \mathbf{T} \rangle$ the schema $\{S_1 \ldots S_n, T_1 \ldots T_m\}$. If I is an instance of $\mathbf{S}$ and J is an instance of $\mathbf{T}$, then the pair $\langle I, J \rangle$ is an instance of $\langle \mathbf{S}, \mathbf{T} \rangle$.

Dependencies Given two schemas, $\mathbf{S}$ and $\mathbf{T}$, an *embedded dependency* [4] is a first-order formula of the form $\forall \overline{x}(\phi(\overline{x}) \rightarrow \exists \overline{y}(\psi(\overline{x}, \overline{y}))$, where $\overline{x}$ and $\overline{y}$ are vectors of variables, $\phi(\overline{x})$ is a conjunction of atomic formulas such that all variables in $\overline{x}$ appear in it, and $\psi(\overline{x}, \overline{y})$ is a conjunction of atomic formulas. Formulas $\phi(\overline{x})$ and $\psi(\overline{x}, \overline{y})$ may contain equations of the form $v_i = v_j$, where v_i and v_j are variables.

An embedded dependency is a *tuple generating dependency* if $\phi(\overline{x})$ and $\psi(\overline{x}, \overline{y})$ only contain relational atoms. It is an *equality generating dependency (egd)* if $\psi(\overline{x}, \overline{y})$ contains only equations. A tgd is called a *source-to-target tgd* if $\phi(\overline{x})$ is a formula over $\mathbf{S}$ and $\psi(\overline{x}, \overline{y})$ over $\mathbf{T}$. It is a *target tgd* if both $\phi(\overline{x})$ and $\psi(\overline{x}, \overline{y})$ are formulas over $\mathbf{T}$.

Mapping Scenario A *mapping scenario* (also called a *data-exchange scenario* or a *schema mapping*) is a quadruple $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$, where $\mathbf{S}$ is a source schema, $\mathbf{T}$ is a target schema, Σ_{st} is a set of source-to-target tgds, and Σ_t is a set of target dependencies that may contain tgds and egds. In the case of interest for this paper, i.e., the case in which the set of target dependencies Σ_t is empty, we will use the notation $(\mathbf{S}, \mathbf{T}, \Sigma_{st})$.

Solutions A source instance for $\mathcal{M}$ is a ground instance I of the source schema $\mathbf{S}$. A target instance for $\mathcal{M}$ is an instance J of the target schema. A target instance J is a solution of $\mathcal{M}$ and a source instance I (denoted $J \in \mathsf{Sol}(\mathcal{M}, I)$) iff $\langle I, J \rangle \models \Sigma_{st} \cup \Sigma_t$.

Given two instances J, J' over a schema $\mathbf{T}$, a *homomorphism* $h : J \rightarrow J'$ is a mapping from $dom(J)$ to $dom(J')$ such that for each $c \in consts(J)$, $h(c) = c$, and for each tuple $t = R(A_1 : v_1, \ldots, A_k : v_k)$ in J it is the case that $h(t) = R(A_1 : h(v_1), \ldots, A_k : h(v_k))$ belongs to J'. Homomorphism h is called an *endomorphism* if $J' \subseteq J$; if $J' \subset J$ it is called a *proper endomorphism*. We say that two instances J, J' are *homomorphically equivalent* if there are homomorphisms $h : J \rightarrow J'$ and $h' : J' \rightarrow J$.

A solution J is *universal* [12] iff for every solution K there is a homomorphism from J to K. The set of universal solutions for $\mathcal{M}$ and I is denoted by $\mathsf{USol}(\mathcal{M}, I)$. Associated with scenario $\mathcal{M}$ is the following *data exchange problem*: given a source instance I, return *none* iff no solution exists, or return a universal solution $J \in \mathsf{USol}(\mathcal{M}, I)$.

4 Dependencies and the Chase

Traditionally, tgds are formulas of the form $\forall \overline{x}(\phi(\overline{x}) \rightarrow \exists \overline{y}(\psi(\overline{x}, \overline{y}))$, where both $\phi(\overline{x})$ and $\psi(\overline{x}, \overline{y})$ are restricted to be conjunctive formulas in which only relational atoms appear.

Tgds are executed using the classical chase procedure. There are several variants of the chase. In this paper, we concentrate on the *standard chase* and on the *naive chase*. In order to define these, we need to introduce the notion of an *assignment*.

Definition 1 ***[Assignments]*** *Given a formula $\varphi(\overline{x}, \overline{y})$, where $\overline{x}$ is a vector of universally quantified variables, and $\overline{y}$ is a vector of existentially quantified variables, an* assignment *for $\varphi(\overline{x}, \overline{y})$ is a mapping $a : \overline{x} \cup \overline{y} \rightarrow$ CONSTS $\cup$ VARS that*

associates with each variable $x_i \in \overline{x}$ a constant $a(x_i) \in$ CONSTS, and with each variable $y_i \in \overline{y}$ a value $a(y_i)$ that can be either a constant or a labeled null.

We say that an assignment a for $\varphi(\overline{x}, \overline{y})$ is canonical *if it injectively associates a labeled null with each existential variable $y_i \in \overline{y}$. The set of facts $a(\varphi(\overline{x}, \overline{y}))$ is called a* canonical block *if a is canonical.*

Given a formula $\varphi(\overline{x}, \overline{y})$, an *instance* of $\varphi(\overline{x}, \overline{y})$ is a set of facts of the form $\varphi(a(\overline{x}), a(\overline{y}))$, for some assignment a, obtained by replacing each variable v_i by $a(v_i)$. Consider for example $\varphi(\langle x_0, x_1, x_2\rangle, \langle y_0, y_1\rangle) = S(x_0, x_1, y_0) \wedge S(x_2, y_1, y_0)$. Following are two of its canonical blocks: $S(a, b, N_0) \wedge S(c, N_1, N_0)$, $S(a, b, N_2) \wedge S(a, N_3, N_2)$. Here are two other instances of the formula that are not canonical blocks: $S(a, b, N_4)$ (in this case $a(\langle x_0, x_1, x_2\rangle) = \langle a, b, a\rangle$, $a(\langle y_0, y_1\rangle) = \langle N_4, b\rangle$), and $S(a, b, N_5) \wedge S(c, d, N_5)$. Consider now the formula $\varphi(\langle x_0, x_1\rangle, \langle y_0\rangle) = S(x_0, y_0) \wedge S(x_1, y_0)$; the following blocks are canonical: $S(a, N_0) \wedge S(b, N_0)$, $S(a, N_1)$ (in this case $a(\langle x_0, x_1\rangle) = \langle a, a\rangle$).

Given a formula $\phi(\overline{x})$ with free variables $\overline{x}$, and an instance I, we say that I *satisfies* $\phi(\overline{x})$ with assignment $\overline{a}$ if $I \models \phi(a(\overline{x}))$.

Definition 2 ***[Standard Chase]*** *[12] Given an instance $\langle I, J\rangle$, during the* standard chase *a tgd $\forall \overline{x} : \phi(\overline{x}) \rightarrow \exists \overline{y}(\psi(\overline{x}, \overline{y}))$ is fired by a value assignment a if $I \models \phi(a(\overline{x}))$ and there is no vector of values $\overline{b}$ such that $J \models \psi(a(\overline{x}), \overline{b})$. To fire the tgd, a is extended to a canonical assignment a' by injectively assigning to each variable $y_i \in \overline{y}$ a fresh null, and then adding the facts in $\psi(a'(\overline{x}), a'(\overline{y}))$ to J.*

It can be seen how the standard chase, before actually firing a tgd on a value assignment, checks that the tgd conclusion is not already satisfied for that assignment. An interesting variant of the chase is the so-called *naive chase*, during which this check is not performed.

Definition 3 ***[Naive Chase]*** *[31] Given an instance $\langle I, J\rangle$, during the* naive chase *a tgd $\forall \overline{x} : \phi(\overline{x}) \rightarrow \exists \overline{y}(\psi(\overline{x}, \overline{y}))$ is fired for all value assignments a such that $I \models \phi(a(\overline{x}))$ by extending a to a canonical assignment a' by injectively assigning to each variable $y_i \in \overline{y}$ a fresh null, and then adding the facts in $\psi(a'(\overline{x}), a'(\overline{y}))$ to J.*

Given a scenario $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st})$, we call a *canonical solution* any solution obtained by chasing the dependencies in Σ_{st} (with either the standard or the naive chase). Any canonical solution is a universal solution [12]. Since all solutions obtained by using the naive chase are equal up to the renaming of nulls, we shall often speak of *the* canonical universal solution.

Example 4.1 Consider the following scenario $\mathcal{M}$:

$$m_1.\ A(x_1, x_2, x_3) \rightarrow \exists Y_1 : S(x_1, Y_1) \wedge S(Y_1, x_2) \wedge T(Y_1, x_3)$$
$$m_2.\ B(x_4, x_5) \rightarrow S(x_4, x_5)$$
$$m_3.\ C(x_6, x_7) \wedge D(x_7, x_8) \rightarrow S(x_6, x_7)$$

and the source instance: $I = \{A(1, 2, 3), B(1, 2), C(1, 2), D(2, 3)\}$. The naive chase generates the following canonical universal solution J for $\mathcal{M}$ over I:

$$J = \{S(1, N_0), S(N_0, 2), T(N_0, 3), S(1, 2)\}$$

◇

We find it useful to introduce a labeling system to identify the *provenance* [8] of tuples in the canonical solution. More specifically, for each tgd m in Σ_{st} and each atom $R(\ldots)$ in the conclusion of m, we associate with $R(\ldots)$ a unique integer *label*, i. Then, given a source instance I, for each tuple t in the canonical universal solution J, we keep track of its *provenance*, $\mathsf{provenance}(t)$, as a set of labeled relation symbols. More formally, whenever t' is generated during the chase by firing tgd m and instantiating atom $R(\ldots)$ in the conclusion of m, we add to the set $\mathsf{provenance}(t)$ the symbol R^i, where i is the label of R.

In Example 4.1, assume tgds are labeled as follows:

$$m_1.\ A(x_1, x_2, x_3) \rightarrow \exists Y_1 : S^1(x_1, Y_1) \wedge S^2(Y_1, x_2) \wedge T^3(Y_1, x_3)$$
$$m_2.\ B(x_4, x_5) \rightarrow S^4(x_4, x_5)$$
$$m_3.\ C(x_6, x_7) \wedge D(x_7, x_8) \rightarrow S^5(x_6, x_7)$$

The provenance of tuples in J would be as follows:

$$J \quad = \quad \{ \quad S(1, N_0)[\{S^1\}], S(N_0, 2)[\{S^2\}], T(N_0, 3)[\{T^3\}], S(1, 2)[\{S^4, S^5\}] \quad \}$$

Based on the labeling system that we have introduced, in the following we shall use labeled formulas of the form $R^i(\ldots)$ as queries that retrieve from an instance all tuples t in relation R such that $R^i \in \mathsf{provenance}(t)$. More specifically, given an atom $R^i(\overline{x}, \overline{y})$, where $\overline{x}$ is a set of universally quantified variables, and $\overline{y}$ a set of existentially quantified variables, an assignment a for $\overline{x}, \overline{y}$, and a canonical instance J, we say that $J \models a(R^i(\overline{x}, \overline{y}))$ if the following hold: (i) J contains a tuple $t = R(a(\overline{x}), a(\overline{y}))$; (ii) $R^i \in \mathsf{provenance}(t)$. Similarly for a conjunction of labeled atoms of the form $\varphi^l(\overline{x}, \overline{y})$.

The canonical solution has the nice property of being a universal solution [12]. Also, the naive chase of a set of s-t tgds can be implemented very efficiently using first-order languages as SQL as a set of queries on the source and insert statements into the target.[1] However, the canonical solution is not, in general, a core solution, and it is known that it may contain quite a lot of redundancy.

In the next Sections, we concentrate on the following problem: given a data-exchange scenario $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st})$, generate an executable script in a first-order language like SQL that, when run on a source instance I computes the core universal solution for $\mathcal{M}$ on I.

A central idea behind our approach is that the computation of core solutions can be implemented by properly rewriting the original scenario into a new set of source-to-target dependencies. However, in order to properly perform the rewriting, we resort to dependencies that are strictly more expressive than ordinary tgds. In particular, we will make extensive use of negation in the premise, and of *Skolem terms* in the conclusion. We call these more expressive dependencies *FO-rules*.

4.1 First-Order Rules

Before introducing the definition of what a FO-rule is, we need to formalize the notion of a Skolem term.

Definition 4 ***[Skolem Term]*** *Given a set of variables $\overline{x}$, a* Skolem term *over $\overline{x}$ is a term of the form $f(x_1, \ldots, x_k)$ where f is a function symbol of arity k and $x_1, \ldots, x_k$ are universal variables in $\overline{x}$.*

Skolem terms are used to create fresh labeled nulls on the target. Traditionally, Skolem functions are considered as *uninterpreted* functions. Given an assignment of values c for $\overline{x}$, with an uninterpreted Skolem term $f(\overline{x})$ we (injectively) associate a labeled null $N_{f(c(\overline{x}))}$. As an alternative, we may consider Skolem functions as being *interpreted*. In this second case, we associate with each function symbol f of arity k a function $f^i : \textsc{consts}^k \rightarrow \textsc{vars}$, and, for each value assignment c for $\overline{x}$, we compute the labeled null as the result of f^i over $c(\overline{x})$, $f^i(c(\overline{x}))$.

If we assume a linear order, $<$, over the underlying set of constants, CONSTS, then, a special class of interpreted Skolem functions are those that rely on the linear order. We shall refer to these functions as *linear-order dependent*. In this paper, we concentrate on a subset of these functions, obtained by composing two simple functions:

- *append*(v_1, v_2), where v_1 and v_2 may be either a constant string or a universally quantified variable, whose result is the string obtained by concatenating the two values;
- *sort*(x_1, x_2), where x_1 and x_2 are universally quantified variables; this function returns a string in which the values of the variables appear in increasing order according to the given linear order; for example, if $x_1 = d, x_2 = c$ and $c < d$, *sort*(x_1, x_2) is the string "$[c, d]$";

To give an example, consider the function: $f(x_1, x_2) = append(\text{“}f\text{”}, sort(x_1, x_2))$; on inputs $x_1 = 1, x_2 = 2, 1 < 2$, the Skolem function generates "$f[1, 2]$", while on inputs $x_1 = 4, x_2 = 3, 3 < 4$, generates "$f[3, 4]$".

Definition 5 ***[FO-Rule]*** *Given a source schema $\mathbf{S}$ and a target schema $\mathbf{T}$, an FO-rule is a dependency of the form $\forall \overline{x} : \phi(\overline{x}) \rightarrow \psi(\overline{x})$ where ϕ is a first-order formula over $\mathbf{S}$ and ψ is a conjunction of atoms of the form $R(t_1, \ldots, t_n)$, with $R \in \mathbf{T}$ and each term t_i is either a variable $t_i \in \{\overline{x}\}$ or a Skolem term over $\overline{x}$.*

[1] Notice that this is not the case if we also allow target constraints, i.e., target tgds or target egds.

Consider the scenario in Example 2.1. Following is a FO-rule from its rewriting:

$$r.\ \forall x_1, x_2 : A(x_1, x_2) \wedge \neg(\exists x_4, x_5 : B(x_1, x_4) \wedge C(x_5, x_2) \wedge x_4 = x_5) \rightarrow \\ S(x_1, f(x_1, x_2)) \wedge T(f(x_1, x_2), x_2)$$

Whenever a set of FO-rules $\{r_1, \ldots, r_n\}$ uses linear-order dependent Skolem functions, we shall say that $\{r_1, \ldots, r_n\}$ is also *linear-order dependent*, and denote it by $\mathcal{R}^< = \{r_1, \ldots, r_n\}$.

To execute a set of FO-rules, we now introduce an extension of the naive chase procedure.

Definition 6 ***[Chasing FO-Rules]*** *Given an FO-rule $\forall \overline{x} : \phi(\overline{x}) \rightarrow \psi(\overline{x})$, we call $Q_\phi(\overline{x})$ the first-order query over $\mathbf{S}$ obtained from $\phi(\overline{x})$ considering $\overline{x}$ as free variables. We denote by $Q_\phi(I)$ the set of tuples $\overline{c} \in dom(I)^{|\overline{x}|}$ such that $\overline{c}$ is an answer of Q_ϕ over I. Given $\overline{c} \in Q_\phi(I)$, we then denote by $\psi(\overline{c})$ the set of atoms obtained from ψ by replacing each variable $x_i \in \overline{x}$ by the corresponding $c_i \in \overline{c}$ and replacing each Skolem term by the corresponding labeled null.*

Given a set $\mathcal{R} = \{r_1, \ldots, r_n\}$ of FO-rules of the form above $r_i.\forall \overline{x} : \phi_i(\overline{x}) \rightarrow \psi_i(\overline{x})$ and a source instance I, we define the result of the chase of $\mathcal{R}$ over I as follows:

$$\mathcal{R}(I) = \bigcup_{i \in [1,n]} \Big(\bigcup_{\overline{c} \in Q_{\phi_i}(I)} (\psi_i(\overline{c})) \Big)$$

Based on this, it should be apparent how FO-rules lend themselves to a natural implementation as an SQL script. Consider for example rule r above from the rewriting of the tgds in Example 2.1. Based on the rule, we can materialize tuples in the S table by the following SQL statement (similarly for T). Notice how string manipulation functions are used to generate the needed Skolem terms:

```
INSERT into S
   SELECT A.a, append('f(', A.a, ',', A.b,')')
   FROM ( SELECT A.a, A.b FROM A
          EXCEPT
          SELECT B.a, C.b FROM B, C WHERE B.b = C.a )
```

4.2 Computing Core Solutions

Given a scenario $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st})$, and an instance I, the *core* [13] of a universal solution $J \in \mathsf{USol}_\mathcal{M}(I)$, $\mathbf{C}$, is a subinstance of J such that there is a homomorphism from J to $\mathbf{C}$, but there is no homomorphism from J to a proper subinstance of $\mathbf{C}$. It is known [13] that cores of the universal solutions for a scenario $\mathcal{M}$ and source instance I are all isomorphic to each other, and therefore it is possible to speak of *the core universal solution*.

We are now ready to introduce the notion of a *core schema-mapping*:

Definition 7 ***[Core Schema Mapping]*** *Given a scenario $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st})$, a set of FO-rules $\mathcal{R}$ is called a* core schema mapping *for $\mathcal{M}$ if, for any source instance I, the canonical target instance $\mathcal{R}(I)$ is the core universal solution for $\mathcal{M}$ over I.*

The main contribution of the paper is a set of algorithms that, given a mapping scenario, rewrite the given tgds as a set of FO-rules, $\mathcal{R}^<$, that represent a core schema mapping. Notice that – as it is common in practical applications – we shall assume a linear order on the underlying set of constants, CONSTS, and exploit the linear order in our Skolemization, as will be detailed in Section 10.

A very similar notion of *laconic schema mapping* was introduced in [31]. Notice, however, that a laconic schema mapping is required to be logically equivalent to the original scenario, i.e., to have the same set of solutions. We do not require the same.

As it was discussed earlier, we concentrate on mapping scenarios composed of a set of s-t tgds Σ_{st} and do not consider target constraints. Without loss of generality we require that the input tgds are in *normal form*, i.e., each tgd uses distinct

variables, and no tgd can be decomposed in two different tgds having the same left-hand side.[2] To formalize this notion, let us introduce the *Gaifman graph* of a formula as the undirected graph in which each variable in the formula is a node, and there is an edge between v_1 and v_2 if v_1 and v_2 occur in the same atom. The *dual Gaifman graph* of a formula is an undirected graph in which nodes are atoms, and there is an edge between atoms $R_i(\overline{x}_i, \overline{y}_i)$ and $R_j(\overline{x}_j, \overline{y}_j)$ if there is some existential variable y_k occurring in both atoms.

Definition 8 ***[Normal Form for Tgds]*** *A set of tgds Σ_{st} is in* normal form *if: (i) for each m_i, $m_j \in \Sigma_{st}$, $(\overline{x}_i \cup \overline{y}_i) \cap (\overline{x}_j \cup \overline{y}_j) = \emptyset$, i.e, the tgds use disjoint sets of variables; (ii) for each tgd m_i, the dual Gaifman graph of atoms in the conclusion of m_i is connected.*

If the input set of tgds is not in normal form, it is always possible to preliminarily rewrite them to obtain an input in normal form. In particular, we introduce a transformation, called normalize, that takes a set of dependencies, Σ_{st} (tgds or FO-rules), and generates a new set of dependencies, normalize(Σ_{st}), in normal form. To do that, it analyzes the dual Gaifman graph of a dependency conclusion. If the graph is not connected, it generates a set of new dependencies with the same premise, one for each connected component in the dual Gaifman graph.

5 A Characterization of the Core

This Section provides an important result upon which we shall build the rewriting algorithms reported in the remainder of the paper. It introduces the key concept of a *witness block*, and shows how it is possible to characterize the core of the universal solutions for a mapping scenario by means of witness blocks. In doing this, it outlines a core computation strategy that will be exploited in the next Sections.

Consider a scenario $\mathcal{M}$ with a set of s-t tgds Σ_{st}; given a source instance, I, each tgd in Σ_{st} represents a constraint that must be satisfied by any solution J for $\mathcal{M}$ over I. Informally speaking, a witness block is a set of facts in J that guarantees that a tgd in Σ_{st} is satisfied for some vector of constants $\overline{c}$. More formally:

Definition 9 ***[Witness Block]*** *Given a scenario $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st})$, a source instance I, and a universal solution $J \in \mathsf{USol}_{\mathcal{M}}(I)$, for each tgd $m. \forall \overline{x} : \phi(\overline{x}) \rightarrow \exists \overline{y}(\psi(\overline{x}, \overline{y})) \in \Sigma_{st}$ and any assignment c for $\overline{x}$ such that $I \models \phi(c(\overline{x}))$, a witness block for $\langle I, J \rangle$, m, and $\overline{c} = c(\overline{x})$ is a set of facts $w \subseteq J$ such that, for some assignment d for $\overline{y}$, it is the case that $w = \psi(c(\overline{x}), d(\overline{y}))$.*

In the following we shall use the following notation:

- $\mathcal{W}_{m,\overline{c}}^{<I,J>}$ will be used to denote the set of all witness blocks for $\langle I, J \rangle$, m, and $\overline{c}$;
- $\mathcal{W}_{m}^{<I,J>}$ the set of all witness blocks for $\langle I, J \rangle$ and m;
- $\mathcal{W}^{<I,J>}$ the set of all witness blocks of $\langle I, J \rangle$.

A key intuition is that there are usually multiple ways to satisfy a tgd m for some vector of constants $\overline{a}$, i.e., a solution usually contains multiple witness blocks for m and $\overline{a}$. The following examples show how the core can be characterized in terms of witness blocks.

Example 5.1 Reconsider the scenario in Example 2.1:

$$
\begin{aligned}
&m_1.\ \forall x_1, x_2 : A(x_1, x_2) \rightarrow \exists Y_1 : S(x_1, Y_1) \wedge T(Y_1, x_2)\\
&m_2.\ \forall x_3, x_4 : B(x_3, x_4) \rightarrow S(x_3, x_4)\\
&m_3.\ \forall x_5, x_6 : C(x_5, x_6) \rightarrow T(x_5, x_6)\\
&m_4.\ \forall x_7 : D(x_7) \rightarrow \exists Y_0 : S(x_7, Y_0)
\end{aligned}
$$

[2] This requirement is pretty common [13, 31, 19].

and the solution: $J = \{S(1, N_0), T(N_0, 2), S(1, 3), T(3, 2), S(1, N_1)\}$ for source instance: $I = \{A(1, 2), B(1, 3), C(3, 2), D(1)\}$. Following are the witness blocks for J:

$$\begin{array}{lcllcl} \mathcal{W}^{<I,J>}_{m_1,\langle 1,2\rangle} & = & \{\{S(1, N_0), T(N_0, 2)\}, \{S(1, 3), T(3, 2)\}\} & \mathcal{W}^{<I,J>}_{m_2,\langle 1,3\rangle} & = & \{\{S(1, 3)\}\} \\ \mathcal{W}^{<I,J>}_{m_4,\langle 1\rangle} & = & \{\{S(1, N_1)\}, \{S(1, N_0)\}, \{S(1, 3)\}\} & \mathcal{W}^{<I,J>}_{m_3,\langle 3,2\rangle} & = & \{\{T(3, 2)\}\} \end{array}$$

The core of J is as follows: $J_0 = \{S(1, 3), T(3, 2)\}$. The witness blocks for J_0 are as follows:

$$\begin{array}{lcllcl} \mathcal{W}^{<I,J_0>}_{m_1,\langle 1,2\rangle} & = & \{\{S(1, 3), T(3, 2)\}\} & \mathcal{W}^{<I,J_0>}_{m_2,\langle 1,3\rangle} & = & \{\{S(1, 3)\}\} \\ \mathcal{W}^{<I,J_0>}_{m_4,\langle 1\rangle} & = & \{\{S(1, 3)\}\} & \mathcal{W}^{<I,J_0>}_{m_3,\langle 3,2\rangle} & = & \{\{T(3, 2)\}\} \end{array}$$

◇

In Example 5.1 witness blocks are quite simple due to the absence of duplicate symbols in tgd conclusions. Whenever the conclusion $\phi(\overline{x})$ of a tgd m is such that the same relation symbol occurs more than once, we say that m contains *self-joins*. The following example shows the witness blocks of a scenario with self-joins in tgd conclusions.

Example 5.2 Consider now the following scenario $\mathcal{M}$:

$$\begin{array}{l} m_1.\ A(x_1, x_2, x_3) \rightarrow \exists Y_1, Y_2 : S(x_1, Y_1, Y_2) \wedge S(x_2, x_3, Y_2) \\ m_2.\ B(x_4, x_5) \rightarrow \exists Y_3, Y_4 : S(x_4, x_5, Y_3) \wedge S(Y_4, x_5, Y_3) \end{array}$$

and the source instance: $I = \{A(3, 1, 2), B(1, 2)\}$. The canonical universal solution is:

$$J = \{S(3, N_0, N_1), S(1, 2, N_1), S(1, 2, N_4), S(N_5, 2, N_4)\}$$

Following are some sets of witness blocks for J:

$$\begin{array}{lcll} \mathcal{W}^{<I,J>}_{m_1,\langle 3,1,2\rangle} & = & \{ & \{S(3, N_0, N_1), S(1, 2, N_1)\} \qquad\qquad\qquad\qquad\qquad\qquad\quad \} \\ \mathcal{W}^{<I,J>}_{m_2,\langle 1,2\rangle} & = & \{ & \{S(1, 2, N_4), S(N_5, 2, N_4)\}, \{S(1, 2, N_4)\}, \{S(1, 2, N_1)\} \quad \} \end{array}$$

The core of J is as follows: $J_0 = \{S(3, N_0, N_1), S(1, 2, N_1)\}$.

The witness blocks for J_0 are as follows:

$$\mathcal{W}^{<I,J_0>}_{m_1,\langle 3,1,2\rangle} = \{\{S(3, N_0, N_1), S(1, 2, N_1)\}\} \qquad \mathcal{W}^{<I,J_0>}_{m_2,\langle 1,2\rangle} = \{\{S(1, 2, N_1)\}\}$$

◇

An important observation is that other algorithms for core computation ([13, 18, 31]) have so far concentrated on a different notion of "blocks", namely *fact blocks*. Informally speaking, a fact block in an instance is a set of facts that are joined via labeled nulls. More formally, it is a connected component in the dual Gaifman graph of an instance, in which facts are the nodes, and there exists an edge between any two facts in which the same labeled null appears.

We want to emphasize that witness blocks are a different concept with respect to fact blocks, since they are essentially instances of tgd conclusions. In some cases the witness blocks of a tgd are also fact blocks. In other cases, they are unions of fact blocks. However, the following example shows that there may be witness blocks in an instance that are neither fact blocks nor unions of fact blocks.

Example 5.3 Consider now the following scenario $\mathcal{M}$:

$$\begin{array}{l} m_1.\ A(x_0, x_1, x_2, x_3) \wedge B(x_3, x_4) \rightarrow \exists Y_0, Y_1, Y_2, Y_3 : S(x_3, x_0, Y_0, x_1) \wedge \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad S(Y_1, x_0, Y_0, x_0) \wedge S(Y_1, x_2, Y_2, Y_3) \end{array}$$

and the source instance: $I = \{A(1, 1, 2, 1), A(2, 1, 2, 1), B(1, 4)\}$. The core universal solution for $\mathcal{M}$ over I is $J_0 = \{S(1, 1, N_0, 1), S(1, 2, N_5, 1), S(N_6, 2, N_5, 2)\}$.

The witness blocks for J_0 are as follows:

$$\begin{array}{lcll} \mathcal{W}^{<I,J_0>}_{m_1,\langle 1,1,2,1\rangle} & = & \{ & \{S(1, 1, N_0, 1), S(1, 2, N_5, 1)\} \quad \} \\ \mathcal{W}^{<I,J_0>}_{m_1,\langle 2,1,2,1\rangle} & = & \{ & \{S(1, 2, N_5, 1), S(N_6, 2, N_5, 2)\} \quad \} \end{array}$$

Notice how the witness block in $\mathcal{W}^{<I,J_0>}_{m_1,\langle 1,1,2,1\rangle}$ is not a fact block, nor the union of two fact blocks (it is rather the union of a fact block and a fragment of another fact block). ◇

Our goal is to find a characterization of the core in terms of its witness blocks. However, as it can be seen from the examples above, some of the witness blocks in the canonical solution are redundant, and therefore the corresponding tuples need to be removed to generate the core. As it is natural, we use the notion of a homomorphism to define what a "redundant" witness block is.

Recall that, given two instances J, J', a *homomorphism* $h : J \to J'$ is a mapping from $dom(J)$ to $dom(J')$ that maps constants to themselves (i.e. for each $c \in consts(J)$, $h(c) = c$) such that for each tuple $t = R(A_1 : v_1, \ldots, A_k : v_k)$ in J it is the case that $h(t) = R(A_1 : h(v_1), \ldots, A_k : h(v_k))$ belongs to J'. We say that h is *injective* if it maps distinct atoms in J into distinct atoms of J'. We say that h is *surjective*, or that it is a *surjection*, if every atom of J' is the image of some atom of J according to h.

There are several ways in which tuples can be made redundant in a solution. Generally speaking, tuples are redundant whenever they introduce unnecessary nulls. To formalize this notion, we introduce a classification of homomorphisms, as follows:

Definition 10 ***[Classification of Homomorphisms]*** *Given two instances J, J', and a homomorphism $h : J \to J'$:*

- *h is* compacting *if it is surjective, and $|vars(J')| < |vars(J)|$; we write $J \prec J'$ if there is a compacting homomorphism of J into J';*
- *h is* proper *if it is injective and not surjective, i.e., J' contains at least one atom that is not the image of an atom of J; in symbols, we write that $J < J'$;*
- *h is an* isomorphism *if it is surjective and injective and its inverse is also a homomorphism; in this case, we say that J and J' are* isomorphic*, in symbols $J \cong J'$.*

In terms of witness blocks, we can identify two main reasons according to which a witness block w can be made redundant by another witness block w' for the same tgd and assignment. The first one is if w' is *more compact* than w, i.e., if there exists a compacting homomorphism of w into w'.

To give an example, consider Example 5.2, and tgd m_2:

$$m_2.\ B(x_4, x_5) \to \exists Y_3, Y_4 : S(x_4, x_5, Y_3) \wedge S(Y_4, x_5, Y_3)$$

The tgd has the following witness blocks:

$$\mathcal{W}^{<I,J>}_{m_2,\langle 1,2\rangle} \quad = \quad \{ \quad \{S(1,2,N_4), S(N_5,2,N_4)\}, \{S(1,2,N_4)\}, \{S(1,2,N_1)\} \quad \}$$

Notice, however, that the witness block $w_1 = \{S(1,2,N_4), S(N_5,2,N_4)\}$ has a compacting homomorphism into $w_2 = \{S(1,2,N_4)\}$; in fact, w_2 contains a lower number of nulls than w_1. This means that w_1 is redundant for core computation properties.

It can be seen that the $\prec$ relation associated with compacting homomorphisms is antisymmetric and transitive, and therefore induces a partial order on witness blocks. A first intuition of our algorithm is that of selecting, among all possible witness blocks, only those that represent *maximal elements* with respect to this partial order, in order to minimize the null values in the final result.

However, even such maximal elements may still be redundant. In fact, other tgds and assignments may generate witness blocks that are "more informative". A witness block w' is said to be *more informative* than a witness block w if there exists a proper homomorphism of w into w'.

Consider again Example 5.2. We have learned that $w_2 = \{S(1,2,N_4)\}$ is a maximal element with respect to compacting homomorphisms for m_2 and $\langle 1,2\rangle$. However, it is still redundant in terms of core computation. In fact, among the witness blocks of m_1 we find $w_3 = \{S(3,N_0,N_1), S(1,2,\ N_1)\}$. It can be seen that w_2 has a proper homomorphism into w_3. In essence, the null N_1 carries "more information" than N_4. We need therefore to discard w_2 and consider w_3 only to generate the core.

Again, proper homomorphisms induce a partial order on the set of witness blocks. In light of this, our core computation procedure will proceed as follows:

- we shall first select the most compact witness blocks in any set $\mathcal{W}^{<I,J>}_{m,\overline{a}}$, i.e., all maximal elements with respect to the $\prec$ partial order;
- then, we will exclude all elements such that there are more informative witness blocks, i.e., we will select the maximal elements with respect to the $<$ partial order.

More formally, given a set of witness blocks $\mathcal{W}$, we define:

$$\begin{array}{lcl} \mathsf{mostCompact}(\mathcal{W}) & = & \{w \mid w \in \mathcal{W} \ \wedge \ \neg\exists\, w' \in \mathcal{W} : w \prec w'\} \\ \mathsf{mostInformative}(\mathcal{W}) & = & \{w \mid w \in \mathcal{W} \ \wedge \ \neg\exists\, w' \in \mathcal{W} : w < w'\} \end{array}$$

By doing this, we are able to remove most of the redundancy in the original solution. Unfortunately, not enough to generate the core. In fact, it may be the case that multiple isomorphic copies of a witness block survive in the result. To see this, consider the following example:

Example 5.4 Consider again the mapping scenario in Example 5.2:

$$\begin{array}{l} m_1.\ A(x_1,x_2,x_3) \rightarrow \exists Y_1, Y_2 : S(x_1,Y_1,Y_2) \wedge S(x_2,x_3,Y_2) \\ m_2.\ B(x_4,x_5) \rightarrow \exists Y_3, Y_4 : S(x_4,x_5,Y_3) \wedge S(Y_4,x_5,Y_3) \end{array}$$

and a different source instance: $I = \{A(1,1,2), B(1,2)\}$, for which the canonical universal solution is

$$J = \{S(1,N_0,N_1), S(1,2,N_1), S(1,2,N_4), S(N_5,2,N_4)\}$$

Following are some sets of witness blocks for J:

$$\begin{array}{lcl} \mathcal{W}^{<I,J>}_{m_1,\langle 1,1,2\rangle} & = & \{ \quad \{S(1,N_0,N_1), S(1,2,N_1)\}, \{S(1,2,N_4)\}, \{S(1,2,N_1)\} \quad \} \\ \mathcal{W}^{<I,J>}_{m_2,\langle 1,2\rangle} & = & \{ \quad \{S(1,2,N_4), S(N_5,2,N_4)\}, \{S(1,2,N_4)\}, \{S(1,2,N_1)\} \quad \} \end{array}$$

By taking the union of the set of maximal witness blocks, we obtain the following solution: $J^* = \{S(1,2,N_4), S(1,2,N_1)\}$ that is obviously not the core. In fact, the two maximal witness blocks are isomorphic to each other, and we need to consider only one of them. We may say that, by selecting maximal witness blocks, we are able to identify two alternative subsets of J that correspond to the core, so that we need to pick one of them. $\diamond$

After we have selected a set of maximal witness blocks, to generate an isomorphism-free solution we introduce a minimization algorithm, called reduce, that works as follows:

- given a set of witness blocks $\mathcal{W}$, it identifies all equivalence classes $\mathcal{E}_0, \ldots, \mathcal{E}_k$ of isomorphic witness blocks in $\mathcal{W}$;
- for each equivalence class, it (nondeterministically) selects exactly one representative, $w_{\mathcal{E}_i}$;
- then, it returns the subset of $\mathcal{W}$ obtained by taking the representative of each equivalence class, i.e., $\mathcal{W} = \{w_{\mathcal{E}_i} | i = 0, \ldots, k\}$.

Based on this intuition, we are ready to formalize our characterization of the core.

Theorem 1 *Given a scenario $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st})$, and a source instance I, suppose J is a universal solution for $\mathcal{M}$ over I. Consider the subset J_0 of J defined as follows:*

$$J_0 = \bigcup \mathsf{reduce}(\mathsf{mostInformative}(\mathsf{mostCompact}(\mathcal{W}^{<I,J>}))) \tag{1}$$

Then, J_0 is the core of J.

The proof is in the Appendix.

6 Expansions

Given a mapping scenario, $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st})$, our goal is to rewrite the given tgds under a set of FO-rules that represents a core schema mapping for $\mathcal{M}$, and then to generate an SQL script from them. In the following, we assume that the input scenario, $\mathcal{M}$, is fixed. To simplify the notation, from now on, we shall omit explicit references to $\mathcal{M}$ whenever this is clear from the context.

In order to perform the rewriting, we shall rely on the characterization of the core introduced in Section 5. A central intuition is that it is possible to select the needed witness blocks by using a set of first-order rules. In this Section we introduce the central notion of an *expansion* of a tgd conclusion, that we shall use in the next sections to perform the rewriting.

More specifically:

- in this Section we formalize the definition of an *expansion* for a tgd, and discuss the relationship among expansions and witness blocks;
- in order to do this, we introduce the notion of a *formula homomorphism*; we use formula homomorphisms to rewrite expansions in such a way to select only maximal witness blocks;
- in the next Section, we provide an alternative characterization of the core that mirrors the one in Section 5, but it is based on expansions and their homomorphisms;
- finally, in Section 8 we show that expansions, which are formulas over the target, can be fairly easily rewritten as formulas over the source database; in this way, we provide a solid foundation for the rewriting algorithm developed in the following sections.

Throughout this section, we shall mainly refer to the scenario in Example 5.2, of which we report the tgds here, complete of labels.

$$m_1.\ A(x_1, x_2, x_3) \rightarrow \exists Y_1, Y_2 : S^1(x_1, Y_1, Y_2) \wedge S^2(x_2, x_3, Y_2)$$
$$m_2.\ B(x_4, x_5) \rightarrow \exists Y_3, Y_4 : S^3(x_4, x_5, Y_3) \wedge S^4(Y_4, x_5, Y_3)$$

Recall that in our labeling system, atom S^i corresponds to all tuples in relation S whose provenance contains label i. In fact, in this Section we will systematically make use of labeled formulas. We use the notation $\varphi^l(\overline{x}, \overline{y})$ to denote a labeled conjunctive formula with universally quantified variables $\overline{x}$, and existentially quantified variables $\overline{y}$.

In the discussion we shall mainly make reference to the source instance

$$I = \{A(3,1,2), A(1,1,2), B(1,2)\}$$

and to the canonical universal solution J, which we report here along with the provenance of the various tuples, as discussed in Section 4:

$$\begin{aligned} J \ = \ \{ \ & S(3, N_0, N_1)[S^1],\ S(1,2,N_1)[S^2],\ S(1,N_2,N_3)[S^1],\ S(1,2,N_3)[S^2], \\ & S(1,2,N_4)[S^3],\ S(N_5,2,N_4)[S^4] \ \} \end{aligned}$$

The witness blocks in J are as follows:

$$\begin{aligned} \mathcal{W}^{<I,J>}_{m_1,\langle 3,1,2\rangle} &= \{\{S(3,N_0,N_1), S(1,2,N_1)\}\} \\ \mathcal{W}^{<I,J>}_{m_1,\langle 1,1,2\rangle} &= \{\{S(1,N_2,N_3), S(1,2,N_3)\}, \{S(1,2,N_1)\}, \{S(1,2,N_3)\}, \{S(1,2,N_4)\}\} \\ \mathcal{W}^{<I,J>}_{m_2,\langle 1,2\rangle} &= \{\{S(1,2,N_4), S(N_5,2,N_4)\}, \{S(1,2,N_1)\}, \{S(1,2,N_3)\}, \{S(1,2,N_4)\}\} \end{aligned}$$

6.1 Introducing Expansions

Once the canonical universal solution J for I has been generated by chasing the original tgds, our next step is to select the witness blocks that belong to the core. Notice that, since J is a finite instance, $\mathcal{W}^{<I,J>}$ is a finite set, and therefore the set of witness blocks for each tgd is finite. Our intuition is to generate a set of queries, called *expansions*, that capture the witness blocks in $\mathcal{W}^{<I,J>}$.

To give an example, consider mapping m_1 above. Tgd m_1 states that the target must contain a number of tuples in S that satisfy the tgd conclusion. The formula:

$$\epsilon_{11} = S^1(x_1, Y_1, Y_2) \wedge S^2(x_2, x_3, Y_2)$$

is called the *base expansion* of m_1, and by running the corresponding query over J we find a number of witness blocks for m_1. In our example, it selects the witness blocks $w_0 = \{S(3, N_0, N_1), S(1, 2, N_1)\}$ and $w_1 = \{S(1, N_2, N_3), S(1, 2, N_3)\}$ (notice that both of these blocks are canonical).

However, it does not capture all witness blocks for tgd m_1. In fact, tuples in J that satisfy the conclusion of m_1 (i) do not necessarily belong to the extent of S^1, S^2, since they may also come from S^3 or S^4; (ii) these tuples are not necessarily distinct, since there may be tuples that perform a self-join.

One alternative way to generate valid witness blocks for m_1 is to use only one tuple from S^2 in join with itself on the last attribute – i.e., S^2 is used to "cover" the S^1 atom. However, this may work as long as the two atoms generate tuples that do not conflict with the constants in the base expansion of m_1; in our example, the values generated by the S^2 atom must be consistent with those that would be generated by the S^1 atom in the base expansion, i.e., $x_2 = x_1$. Generally speaking, any expansion must be consistent with the base expansion, i.e., with the tgd conclusion. We write this second expansion as follows, first in its extended and more verbose form, to emphasize that an intersection with the base expansion is required, and then in its simplified form:

$$\begin{aligned} \epsilon_{12} &= S^2(x_2, x_3, Y_2) \wedge \exists x_1, Y_1 : (S^1(x_1, Y_1, Y_2) \wedge S^2(x_2, x_3, Y_2) \wedge x_1 = x_2) \\ &= S^2(x_2, x_3, Y_2) \wedge \exists Y_1 : (S^1(x_2, Y_1, Y_2)) \end{aligned}$$

Expansion ϵ_{12} captures witness blocks $w_2 = \{S(1, 2, N_1)\}$ and $w_3 = \{S(1, 2, N_3)\}$. To identify the last witness block for m_1, $w_4 = \{S(1, 2, N_4)\}$, we need one final expansion, that uses a single atom for S^3 (notice, in fact, that this witness block contains an atom whose provenance is S^3 in mapping m_2):

$$\begin{aligned} \epsilon_{13} &= S^3(x_4, x_5, Y_3) \wedge \exists x_1, x_2, x_3, Y_1, Y_2 : (S^1(x_1, Y_1, Y_2) \wedge S^2(x_2, x_3, Y_2) \wedge \\ &\qquad x_4 = x_1 \wedge x_4 = x_2 \wedge x_5 = x_3) \\ &= S^3(x_4, x_5, Y_3) \wedge \exists Y_1, Y_2 : (S^1(x_4, Y_1, Y_2) \wedge S^2(x_4, x_5, Y_2)) \end{aligned}$$

A similar approach can be used for tgd m_2 above. In this case, the algorithm first generates the base expansion:

$$\epsilon_{21} = \quad S^3(x_4, x_5, Y_3) \wedge S^4(Y_4, x_5, Y_3)$$

As it was noted [31, 19], the base expansion is hardly useful for core computation purposes. Consider the facts obtained from ϵ_{21} by considering each x_i as a constant and each Y_i as a labeled null. It is easy to see that this set is not a core. In fact, the second atom is useless with respect to the first one, since it has identical values on the second and third attribute, and a null instead of a constant on the first one. More interesting expansions are the following;

$$\begin{aligned} \epsilon_{22} &= S^3(x_4, x_5, Y_3) \\ \epsilon_{23} &= S^2(x_2, x_3, Y_2) \wedge \exists x_4, x_5, Y_3, Y_4 : (S^3(x_4, x_5, Y_3) \wedge S^4(Y_4, x_5, Y_3) \wedge \\ &\qquad x_2 = x_4 \wedge x_3 = x_5) \\ &= S^2(x_2, x_3, Y_2) \wedge \exists Y_3, Y_4 : (S^3(x_2, x_3, Y_3) \wedge S^4(Y_4, x_3, Y_3)) \end{aligned}$$

Notice that no intersection is present in ϵ_{22}, since we know that atom S^3 always covers S^4.

6.2 Formula Homomorphisms

In order to develop an algorithm that finds all expansions of a tgd conclusion, we introduce a notion of *formula homomorphism*, which is reminiscent of the notion of *containment mapping* used in [22]. We find it useful to define homomorphisms among variable occurrences, and not among variables.

Definition 11 ***[Variable Occurrence]*** *Given an atom $R^l(A_1 : v_1, \ldots, A_k : v_k)$ in a formula $\varphi^l(\overline{x}, \overline{y})$, a* variable occurrence *is a pair $R^l.A_j : v_i$. A variable occurrence $R^l.A_j : v_i$ in $\varphi^l(\overline{x}, \overline{y})$ is a* universal occurrence *if $v_i \in \overline{x}$; it is an* existential occurrence *if $v_i \in \overline{y}$.*

In the following, we denote by $occ(\varphi^l(\overline{x}, \overline{y}))$ the set of all variable occurrences in $\varphi^l(\overline{x}, \overline{y})$; $u\text{-}occ(\varphi^l(\overline{x}, \overline{y}))$, $e\text{-}occ(\varphi^l(\overline{x}, \overline{y}))$ will denote the set of universal and existential occurrences, respectively. Similarly, $occ(v)$, $u\text{-}occ(v)$, $e\text{-}occ(v)$ will denote the set of all (universal, existential) occurrences of a given variable v.

Definition 12 ***[Formula Homomorphism]*** *Given two conjunctive formulas, $\varphi^l(\overline{x}, \overline{y})$ and $\varphi^{'l}(\overline{x}', \overline{y}')$, a* formula homomorphism *is an injective mapping h^f from the set $occ(\varphi^l(\overline{x}, \overline{y}))$ to $occ(\varphi^{'l}(\overline{x}', \overline{y}'))$ such that:*

- *h^f maps universal occurrences into universal occurrences;*
- *for each atom $R^l(A_1 : v_1, \ldots, A_k : v_k) \in \varphi^l(\overline{x}, \overline{y})$, it is the case that $R(h^f(R^l.A_1 : v_1), \ldots, h^f(R^l.A_k : v_k)) \in \varphi^{'l}(\overline{x}', \overline{y}')$;*
- *for each pair of occurrences of an existential variable $y \in \overline{y}$, $R^l_i.A_j : y$, $R^l_n.A_m : y$ it is the case that either $h^f(R^l_i.A_j : y)$ and $h^f(R^l_n.A_m : y)$ are both universal, or they are occurrences of the same existential variable $y' \in \overline{y}'$.*

We say that a formula homomorphism h^f is *injective* if it maps distinct atoms of $\varphi^l(\overline{x}, \overline{y})$ into distinct atoms of $\varphi^{'l}(\overline{x}', \overline{y}')$. It is *surjective* if every atom in $\varphi^{'l}(\overline{x}', \overline{y}')$ is the image of some atom in $\varphi^l(\overline{x}, \overline{y})$ according to h^f.

Consider for example the two formulas over relations $R(A, B, C)$ and $T(A, B, C)$: $\varphi^l = R^1(x_1, x_2, Y_1) \wedge T^2(x_3, x_1, Y_1)$ and $\varphi^{'l} = R^3(x'_4, x'_5, x'_6) \wedge T^4(x'_9, x'_7, x'_8)$. There exists a formula homomorphism h^f of φ^l into $\varphi^{'l}$, based on the following mapping of variable occurrences:

$$h^f(R^1.A : x_1) \rightarrow R^3.A : x'_4 \qquad h^f(R^1.B : x_2) \rightarrow R^3.B : x'_5$$
$$h^f(R^1.C : Y_1) \rightarrow R^3.C : x'_6 \qquad h^f(T^2.A : x_3) \rightarrow T^4.A : x'_9$$
$$h^f(T^2.B : x_1) \rightarrow T^4.B : x'_7 \qquad h^f(T^2.C : Y_1) \rightarrow T^4.C : x'_8$$

Consider now the two formulas: $\varphi^l = R^1(x_1, x_2, Y_2)$ and $\varphi^{'l} = R^2(x'_4, x'_5, Y'_3) \wedge R^3(Y'_4, x'_5, Y'_3)$. There exists a formula homomorphism $h^{f'}$ of φ^l into $\varphi^{'l}$, based on the following mapping of variable occurrences:

$$h^{f'}(R^1.A : x_1) \rightarrow R^2.A : x'_4 \qquad h^{f'}(R^1.B : x_2) \rightarrow R^2.B : x'_5$$
$$h^{f'}(R^1.C : Y_2) \rightarrow R^2.C : Y'_3$$

It can be seen that, since formula homomorphisms map variable occurrences into variable occurrences, they may relate occurrences of the same variable on the left hand side to occurrences of different variables on the right hand side. In the following, we shall refer to the variable occurrence $h^f(R^l.A_j : v_i)$ by the syntax $A_j : {h^f}_{R^l.A_j}(v_i)$, so that ${h^f}_{R^l.A_j}(v_i)$ will be the variable whose occurrence is associated with occurrence $R^l.A_j$ of v_i. Consider for example h^f above. The two occurrences of variable x_1 in φ^l are mapped to occurrences of different universal variables in $\varphi^{'l}$; in fact, ${h^f}_{R^1.A}(x_1) = x'_4$, while ${h^f}_{T^2.B}(x_1) = x'_7$.

Parallel to the classification of homomorphisms among facts introduced in Section 5, it is useful to classify formula homomorphisms in several categories, as follows.

Definition 13 ***[Classification of Formula Homomorphisms]*** *Given two formulas, $\varphi^l(\overline{x}, \overline{y})$ and $\varphi^{'l}(\overline{x}', \overline{y}')$, and a formula homomorphism h^f from $occ(\varphi^l(\overline{x}, \overline{y}))$ to $occ(\varphi^{'l}(\overline{x}', \overline{y}'))$,*

- h^f *is* compacting *if it is surjective, and either* $|\varphi'^l(\overline{x}', \overline{y}')| < |\varphi^l(\overline{x}, \overline{y})|$ *or* $|\overline{y}'| < |\overline{y}|$*, i.e., either* $\varphi'^l(\overline{x}', \overline{y}')$ *is smaller than* $\varphi^l(\overline{x}, \overline{y})$ *or it contains less existential variables;*

- h^f *is said to be* proper *if it is injective but not surjective, i.e., there is at least one atom in* $\varphi'^l(\overline{x}', \overline{y}')$ *which is not the image of an atom of* $\varphi^l(\overline{x}, \overline{y})$*.*

In our examples above, formula homomorphism h^f is compacting, while $h^{f'}$ is proper.

It is very important to discuss the relationship between formula homomorphisms and the corresponding homomorphisms among facts. In essence, we study formula homomorphisms in order to detect possible homomorphisms among facts that are instances of the formulas. However, the presence of a formula homomorphism does not guarantee that actual homomorphisms arise among facts. Notice, in fact, that homomorphisms among formulas map occurrences of a universal variable into occurrences of other universal variables. However, these variables do not necessarily receive the same values when the formulas are instantiated. Therefore, the homomorphism may be "realized" or not among formula instances depending on values assumed by the universal variables.

Consider for example the formula homomorphism h^f of $\varphi^l = R^1(x_1, x_2, Y_1) \wedge T^2(x_3, x_1, Y_1)$ and $\varphi'^l = R^3(x'_4, x'_5, x'_6) \wedge T^4(x'_9, x'_7, x'_8)$, above. Consider now two instances of the formulas: $w = \{R(1, 2, N_0), T(3, 1, N_0)\}$ and $w' = \{R(1, 2, 4), T(3, 1, 4)\}$, respectively. Given this assignments of values to the variables, it can be seen that the set of facts w has in fact a (compacting) homomorphism into w'. This is in general not true if we change the assignments.

Consider now instances: $w'' = \{R(1, 2, N_0), T(3, 1, N_0)\}$ and $w''' = \{R(1, 4, 6),\ T(3, 5, 7)\}$, respectively. There is no homomorphism of w'' into w'''. The reason for this is twofold. (i) First, the formula homomorphism maps the occurrence of x_2 on the left into the occurrence of x'_5 on the right. By doing this, the formula homomorphism is imposing a restriction on the values of variables x_2, x'_5: in order to realize the homomorphism among formula instances, it must be the case that the two variables receive the same value. (ii) Second, h^f maps the two occurrences of N_0 into occurrences of x'_6, x'_8; this means that, in order for the homomorphism to be realized, it is necessary that the value of x'_6 equals that of x'_8.

More formally, given a formula homomorphism h^f, we may introduce several sets of equalities among universal variables that are associated with h^f:

- the set $\textsc{intersect}_{h^f}$ states the set of equalities among universal variables of $\varphi^l(\overline{x}, \overline{y})$ and universal variables of $\varphi'^l(\overline{x}', \overline{y}')$ that must hold to realize the homomorphism among instances of the two formulas:

$$\textsc{intersect}_{h^f}(\overline{x}, \overline{x}') = \{x_i = x'_j \mid h^f(R.A : x_i) = R.A : x'_j,\ x_i \in \overline{x},\ x'_j \in \overline{x}'\}$$

- the set $\textsc{joins}_{h^f}$ states the set of equalities among universal variables of $\varphi'^l(\overline{x}', \overline{y}')$ whose occurrences are images of occurrences of the same existential variable in $\varphi^l(\overline{x}, \overline{y})$:

$$\textsc{joins}_{h^f}(\overline{x}') = \{x'_h = x'_l \mid x'_h = h^f{}_{R_i.A_j}(y_k),\ x'_l = h^f{}_{R_n.A_m}(y_k),\ y_k \in \overline{y}\}$$

The set $\textsc{equal}_{h^f}$ will be the union of the two, as follows:

$$\textsc{equal}_{h^f}(\overline{x}, \overline{x}') = \textsc{intersect}_{h^f}(\overline{x}, \overline{x}') \cup \textsc{joins}_{h^f}(\overline{x}')$$

Intuitively, a formula homomorphism can be realized only by assignments that satisfy the equalities in $\textsc{equal}_{h^f}(\overline{x}, \overline{x}')$. In our example, we have that:

$$\textsc{equal}_{h^f}(\overline{x}, \overline{x}') = \{x_1 = x'_4, x_2 = x'_5, x_3 = x'_9, x_1 = x'_7\} \cup \{x'_6 = x'_8\}$$

This notion is made more precise in B.2, where we provide several results on the existence of formula and fact homomorphisms.

6.3 Defining and Finding Expansions

We are now ready to introduce the notion of an *expansion*. Given a scenario $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st})$, we shall denote by:

- $\mathcal{R} = \bigcup_i \psi_i(\overline{x}_i, \overline{y}_i)$ the union of all tgd conclusions in Σ_{st};
- $\mathcal{R}_k^{pow}$ the set of all multisets of atoms in $\mathcal{R}$ of size k or less; whenever multiple copies of the same atom appear in a multiset, we assume that they have been properly renamed to avoid variable collisions.

Definition 14 ***[Expansions of a Tgd]*** *Given a scenario $\mathcal{M}$ and a tgd $m : \phi(\overline{x}_2) \rightarrow \exists \overline{y}_2(\psi^l(\overline{x}_2, \overline{y}_2))$ in Σ_{st}, the set of expansions of m wrt $\mathcal{M}$, denoted by* $\mathsf{expansions}_{\mathcal{M}}(m)$*, is the set of logical formulas of the form:*

$$\epsilon = \chi^l(\overline{x}_1, \overline{y}_1) \wedge \exists \overline{x}_2, \overline{y}_2 : (\psi^l(\overline{x}_2, \overline{y}_2) \bigwedge \text{EQUAL}_{h^f{}_\epsilon}(\overline{x}_1, \overline{x}_2))$$

where $\chi^l(\overline{x}_1, \overline{y}_1)$ is a multiset of labeled atoms in $\mathcal{R}_k^{pow}$ (k is the size of $\psi^l(\overline{x}_2, \overline{y}_2)$) and there exists a surjection $h^f{}_\epsilon : \psi(\overline{x}_2, \overline{y}_2) \rightarrow \chi(\overline{x}_1, \overline{y}_1)$.

Without loss of generality, in the following we shall assume that expansions are such that $\overline{x}_1 \cap \overline{x}_2 = \emptyset$, $\overline{y}_1 \cap \overline{y}_2 = \emptyset$, i.e., $\overline{x}_1, \overline{x}_2$ ($\overline{y}_1, \overline{y}_2$, respectively) are disjoint.

We report here the tgds of Example 5.2, and the set of their expansions:

$$m_1.\ A(x_1, x_2, x_3) \rightarrow \exists Y_1, Y_2 : S^1(x_1, Y_1, Y_2) \wedge S^2(x_2, x_3, Y_2)$$
$$m_2.\ B(x_4, x_5) \rightarrow \exists Y_3, Y_4 : S^3(x_4, x_5, Y_3) \wedge S^4(Y_4, x_5, Y_3)$$

The set $\mathsf{expansions}_{\mathcal{M}}(m_1)$ contains the following three expansions (in simplified form):

$$\begin{aligned}
\epsilon_{11} &= S^1(x_1, Y_1, Y_2) \wedge S^2(x_2, x_3, Y_2) \\
\epsilon_{12} &= S^2(x_2, x_3, Y_2) \wedge \exists Y_1 : (S^1(x_2, Y_1, Y_2)) \\
\epsilon_{13} &= S^3(x_4, x_5, Y_3) \wedge \exists Y_1, Y_2 : (S^1(x_4, Y_1, Y_2) \wedge S^2(x_4, x_5, Y_2))
\end{aligned}$$

Similarly for $\mathsf{expansions}_{\mathcal{M}}(m_2)$:

$$\begin{aligned}
\epsilon_{21} &= S^3(x_4, x_5, Y_3) \wedge S^4(Y_4, x_5, Y_3) \\
\epsilon_{22} &= S^3(x_4, x_5, Y_3) \\
\epsilon_{23} &= S^2(x_2, x_3, Y_2) \wedge \exists Y_3, Y_4 : (S^3(x_2, x_3, Y_3) \wedge S^4(Y_4, x_3, Y_3))
\end{aligned}$$

Notice that an expansion ϵ can be also considered as a query $\epsilon(\overline{x}_1, \overline{y}_1)$ with free variables $\overline{x}_1, \overline{y}_1$. In fact, we will shortly prove that the result of evaluating such queries on a solution $J \in \mathsf{USol}_{\mathcal{M}}(I)$ returns exactly a set of witness blocks in $\mathcal{W}^{<I,J>}$.

More formally, given an instance J, and an assignment a_1 for $\overline{x}_1, \overline{y}_1$, we say that $J \models a_1(\epsilon(\overline{x}_1, \overline{y}_1))$ if the following holds:

- $J \models a_1(\chi^l(\overline{x}_1, \overline{y}_1))$;
- there exists an assignment a_2 such that $J \models a_2(\psi^l(\overline{x}_2, \overline{y}_2))$;
- a_1, a_2 are such that $\text{EQUAL}_{h^f{}_\epsilon}(a_1(\overline{x}_1), a_2(\overline{x}_2))$ evaluates to true.

Algorithm 1 describes how to generate the set $\mathsf{expansions}_{\mathcal{M}}(m)$.

It can be seen that the number of expansions of a tgd conclusion increases with the number of self-joins, and it is in general exponential in the size of the input tgds. We call $\mathsf{expansions}(\mathcal{M})$ the set of all expansions of the tgds in Σ_{st}.

Algorithm 1 FINDING EXPANSIONS

Input: a scenario $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st})$,
a tgd $m. \forall \overline{x} : \phi(\overline{x}) \rightarrow \exists \overline{y}(\psi(\overline{x}, \overline{y})) \in \Sigma_{st}$
Output: the set of expansions $\mathsf{expansions}_{\mathcal{M}}(m)$

Let $\mathsf{expansions}_{\mathcal{M}}(m) = \emptyset$
Let $\mathcal{R} = \bigcup\{\psi_i^l(\overline{x}_i, \overline{y}_i) \mid \forall \overline{x}_i : \phi_i(\overline{x}_i) \rightarrow \exists \overline{y}_i(\psi_i(\overline{x}_i, \overline{y}_i)) \in \Sigma_{st}\}$
Let $k = |\psi(\overline{x}, \overline{y})|$
Rename variables in $\psi^l(\overline{x}, \overline{y})$ as $\psi^l(\overline{x}_2, \overline{y}_2)$
Let $\mathcal{R}_k^{pow}$ be the set of all (renamed) multisets of atoms in $\mathcal{R}$ of size k or less
For each $\chi^l(\overline{x}_1, \overline{y}_1) \in \mathcal{R}_k^{pow}$
  If there exists a surjection $h^f : \psi^l(\overline{x}_2, \overline{y}_2) \rightarrow \chi(\overline{x}_1, \overline{y}_1)$
    Let $\text{INTERSECT}_{h^f}(\overline{x}_1, \overline{x}_2) = \emptyset$
    For each $R^l.A_i : x_{2j} \in u\text{-}occ(\psi^l(\overline{x}_2, \overline{y}_2))$
      $\text{INTERSECT}_{h^f}(\overline{x}_1, \overline{x}_2) = \text{INTERSECT}_{h^f}(\overline{x}_1, \overline{x}_2) \cup \{x_{2j} = h^f{}_{R^l.A_i}(x_{2j})\}$
    Let $\text{JOINS}_{h^f}(\overline{x}_1) = \emptyset$
    For each pair $R_i^l.A_j : y_{2k}, R_n^l.A_m : y_{2k}$ in $e\text{-}occ(\psi^l(\overline{x}_2, \overline{y}_2))$
    such that $h^f(R_i^l.A_j : y_{2k})$ and $h^f(R_n^l.A_m : y_{2k})$ are in $u\text{-}occ(\chi^l(\overline{x}_1, \overline{y}_1))$
      $\text{JOINS}_{h^f}(\overline{x}_1) = \text{JOINS}_{h^f}(\overline{x}_1) \cup \{h^f{}_{R_i^l.A_j}(y_{2k}) = h^f{}_{R_n^l.A_m}(y_{2k})\}$
    Let $\text{EQUAL}_{h^f}(\overline{x}_1, \overline{x}_2) = \text{INTERSECT}_{h^f}(\overline{x}_1, \overline{x}_2) \cup \text{JOINS}_{h^f}(\overline{x}_1)$
    $\mathsf{expansions}_{\mathcal{M}}(m) = \mathsf{expansions}_{\mathcal{M}}(m) \cup$
      $\{\chi^l(\overline{x}_1, \overline{y}_1) \wedge \exists \overline{x}_2, \overline{y}_2 : (\psi^l(\overline{x}_2, \overline{y}_2) \bigwedge \text{EQUAL}_{h^f}(\overline{x}_1, \overline{x}_2))\}$

7 Expansions and the Core

Our goal in this Section is to introduce an alternative characterization of the core that mimics the one given in Section 5 but relies on expansions. In order to do this, we build on the intuition that expansions of a tgd allow us to identify all witness blocks for that tgd. However, there are two main problems to be solved, as shown in Theorem 1:

- in order to generate core solutions, we need to select only maximal witness blocks, first the most compact ones, and then the most informative among these;
- then, we need to handle the possible isomorphisms among maximally informative witness blocks.

In the next subsections we discuss these two aspects.

7.1 More Compact and More Informative Expansions

Formula homomorphisms may help us to identify when an expansion generates witness blocks that are more compact or more informative than another. In fact, we introduce a parallel definition of a *more compact* and *more informative* expansion.

Definition 15 ***[More Compact and More Informative Expansions]*** *Given expansions:*

$$\epsilon = \chi^l(\overline{x}_1, \overline{y}_1) \wedge \exists \overline{x}_2, \overline{y}_2 : (\psi^l(\overline{x}_2, \overline{y}_2) \bigwedge \text{EQUAL}_{h^f{}_\epsilon}(\overline{x}_1, \overline{x}_2))$$
$$\epsilon' = \chi'^l(\overline{x}'_1, \overline{y}'_1) \wedge \exists \overline{x}'_2, \overline{y}'_2 : (\psi'^l(\overline{x}'_2, \overline{y}'_2) \bigwedge \text{EQUAL}_{h^f{}_{\epsilon'}}(\overline{x}'_1, \overline{x}'_2))$$

- ϵ' *is* more compact *than* ϵ *if there exists a compacting homomorphism* ${h^f}_c : \chi^l(\overline{x}_1, \overline{y}_1) \rightarrow \chi'^l(\overline{x}'_1, \overline{y}'_1)$*; in symbols:* $\epsilon \prec \epsilon'$*;*
- ϵ' *is* more informative *than* ϵ *if there exists a proper homomorphism* ${h^f}_p : \chi^l(\overline{x}_1, \overline{y}_1) \rightarrow \chi'^l(\overline{x}'_1, \overline{y}'_1)$*; in symbols:* $\epsilon < \epsilon'$*.*

Consider our running example. As we have discussed above, among the expansions of tgd m_1 we find:

$$\begin{array}{rcll} \epsilon_{11} & = & S^1(x_1, Y_1, Y_2) \wedge & S^2(x_2, x_3, Y_2) \\ \epsilon_{13} & = & S^3(x_4, x_5, Y_3) \wedge & \exists x_1, x_2, x_3, Y_1, Y_2 : (S^1(x_1, Y_1, Y_2) \wedge S^2(x_2, x_3, Y_2) \wedge \\ & & & x_4 = x_1 \wedge x_4 = x_2 \wedge x_5 = x_3) \end{array}$$

It can be seen that there is a compacting formula homomorphism ${h^f}_c$ of $\chi^l_{11} = S^1(x_1, Y_1, Y_2) \wedge S^2(x_2, x_3, Y_2)$ into $\chi^l_{13} = S^3(x_4, x_5, Y_3)$. This is a signal that ϵ_{13} may generate witness blocks that are more compact than those generated by ϵ_{11}. And in fact, this is the case in our example: witness block $w_{11} = \{S(1, N_2, N_3), S(1, 2, N_3)\}$ is made redundant by the more compact witness block $w_{13} = \{S(1, 2, N_4)\}$.

In order to remove witness block w_{11} from the extent of ϵ_{11}, we may think of rewriting ϵ_{11} by adding the *negation* of ϵ_{13}, with the appropriate equalities suggested by their compacting homomorphism, ${h^f}_c$, as follows:

$$\epsilon_{11}(\overline{x}_{11}, \overline{y}_{11}) \wedge \neg \exists \overline{x}_{13}, \overline{y}_{13}(\epsilon_{13}(\overline{x}_{13}, \overline{y}_{13}) \wedge \text{EQUAL}_{{h^f}_c}(\overline{x}_{11}, \overline{x}_{13}))$$

The actual formula is the following:

$$\begin{array}{l} S^1(x_1, Y_1, Y_2) \wedge S^2(x_2, x_3, Y_2) \wedge \\ \quad \neg \exists x'_4, x'_5, Y'_3 : (S^3(x'_4, x'_5, Y'_3) \wedge \\ \qquad \exists x''_1, x''_2, x''_3, Y''_1, Y''_2 : (S^1(x''_1, Y''_1, Y''_2) \wedge S^2(x''_2, x''_3, Y''_2) \wedge \\ \qquad\quad x'_4 = x''_1 \wedge x'_4 = x''_2 \wedge x'_5 = x''_3) \wedge \\ \qquad x_1 = x'_4 \wedge x_2 = x'_4 \wedge x_3 = x'_5) \end{array}$$

The important observation is that this formula is a new query over the canonical universal solution J; differently from the original expansion, ϵ_{11}, only witness blocks that belong to the core satisfy this new query.

Notice that in some cases, given expansions ϵ and ϵ', there can be different formula homomorphisms of $\chi^l(\overline{x}_1, \overline{y}_1)$ into $\chi'^l(\overline{x}'_1, \overline{y}'_1)$. Consider as an example $\chi^l = R^1(x_1, Y_1)$, $\chi'^l = R^2(x_2, Y_2) \wedge R^3(x_3, Y_2)$. It can be seen that a first proper formula homomorphism maps $R^1(x_1, Y_1)$ to $R^2(x_2, Y_2)$, while a second one maps $R^1(x_1, Y_1)$ to $R^3(x_3, Y_2)$. We write $\epsilon \prec_{h^f} \epsilon'$ ($\epsilon <_{h^f} \epsilon'$) when we need to make explicit that ϵ' is more compact (more informative, respectively) than ϵ according to formula homomorphism h^f.

Following this approach, given an expansion ϵ, we generate a new query based on ϵ, called $\mathsf{mostComp}(\epsilon)$, by adding to ϵ the negation of each expansion ϵ' that is more compact than ϵ, with the appropriate equalities, as follows.

Definition 16 ***[Selecting Most-Compact Witness Blocks:*** $\mathsf{mostComp}(\mathcal{M})$ *] Given a scenario* $\mathcal{M}$*, its set of expansions* $\mathsf{expansions}(\mathcal{M})$*, and an expansion*

$$\epsilon = \chi^l(\overline{x}_1, \overline{y}_1) \wedge \exists \overline{x}_2, \overline{y}_2 : (\psi^l(\overline{x}_2, \overline{y}_2) \bigwedge \text{EQUAL}_{{h^f}_\epsilon}(\overline{x}_1, \overline{x}_2))$$

in $\mathsf{expansions}(\mathcal{M})$ *the formula* $\mathsf{mostComp}(\epsilon)$ *is obtained as follows:*

- *initialize* $\mathsf{mostComp}(\epsilon) = \epsilon$*;*
- *for any* $\epsilon' = \chi'^l(\overline{x}'_1, \overline{y}'_1) \wedge \exists \overline{x}'_2, \overline{y}'_2 : (\psi'^l(\overline{x}'_2, \overline{y}'_2) \bigwedge \text{EQUAL}_{{h^f}_{\epsilon'}}(\overline{x}'_1, \overline{x}'_2))$ *in* $\mathsf{expansions}(\mathcal{M})$ *and any compacting formula homomorphism* ${h^f}_c$ *such that* $\epsilon \prec_{{h^f}_c} \epsilon'$*, i.e.,* ϵ' *is more compact than* ϵ *according to* ${h^f}_c$*, add to* $\mathsf{mostComp}(\epsilon)$ *a formula*

$$\wedge \neg \exists \overline{x}'_1, \overline{y}'_1 : (\epsilon' \bigwedge \text{EQUAL}_{{h^f}_c}(\overline{x}_1, \overline{x}'_1))$$

We shall denote by $\mathsf{mostComp}(\mathcal{M})$ *the set of all rewritings of the form* $\mathsf{mostComp}\,(\epsilon), \epsilon \in \mathsf{expansions}(\mathcal{M})$.

After this first rewriting, coherently with the strategy outlined in Theorem 1, we look among other expansions to favor those that generate more informative witness blocks in the target, and we further rewrite $\mathsf{mostComp}\,(\epsilon)$ accordingly. In doing this, we generate a further formula, called $\mathsf{mostInf}\,(\epsilon)$, as follows:

Definition 17 ***[Selecting Most-Informative Witness Blocks:*** $\mathsf{mostInf}(\mathcal{M})$*]* *Given a scenario* $\mathcal{M}$*, its set of expansions* $\mathsf{expansions}(\mathcal{M})$*, and an expansion*

$$\epsilon = \chi^{l}(\overline{x}_1, \overline{y}_1) \wedge \exists \overline{x}_2, \overline{y}_2 : (\psi^{l}(\overline{x}_2, \overline{y}_2) \bigwedge \mathrm{EQUAL}_{h^{f}{}_{\epsilon}}(\overline{x}_1, \overline{x}_2))$$

in $\mathsf{expansions}(\mathcal{M})$ *the formula* $\mathsf{mostInf}\,(\epsilon)$ *is obtained as follows:*

- *initialize* $\mathsf{mostInf}\,(\epsilon) = \mathsf{mostComp}\,(\epsilon)$*;*
- *for any* $\epsilon' = \chi^{'l}(\overline{x}'_1, \overline{y}'_1) \wedge \exists \overline{x}'_2, \overline{y}'_2 : (\psi^{'l}(\overline{x}'_2, \overline{y}'_2) \bigwedge \mathrm{EQUAL}_{h^{f}{}_{\epsilon'}}(\overline{x}'_1, \overline{x}'_2))$ *in* $\mathsf{expansions}(\mathcal{M})$ *and any proper formula homomorphism* $h^{f}{}_{i}$ *such that* $\epsilon <_{h^{f}{}_{i}} \epsilon'$*, i.e.,* ϵ' *is more informative than* ϵ *according to* $h^{f}{}_{i}$*, add to* $\mathsf{mostInf}\,(\epsilon)$ *a formula*

$$\wedge \neg \exists \overline{x}'_1, \overline{y}'_1 : (\mathsf{mostComp}\,(\epsilon') \bigwedge \mathrm{EQUAL}_{h^{f}{}_{i}}(\overline{x}_1, \overline{x}'_1))$$

We shall denote by $\mathsf{mostInf}(\mathcal{M})$ *the set of all rewritings of the form* $\mathsf{mostInf}\,(\epsilon)$*, for every* $\epsilon \in \mathsf{expansions}(\mathcal{M})$.

To summarize, in order to select maximal witness blocks, we consider each expansion ϵ of a tgd m; then:

- we first rewrite ϵ into a new formula $\mathsf{mostComp}\,(\epsilon)$ by adding the negation of all expansions ϵ_i such that ϵ_i is more compact than ϵ; we expect these new formulas to select the most-compact witness blocks among those associated with ϵ;
- then, we further rewrite $\mathsf{mostComp}\,(\epsilon)$ into a new formula $\mathsf{mostInf}\,(\epsilon)$ by adding the negation of $\mathsf{mostComp}\,(\epsilon_j)$, for all expansions ϵ_j such that ϵ_j is more informative than ϵ.

Similarly to expansions, also their rewritings can be considered as queries. More specifically, given an expansion ϵ, and the associated query $\epsilon(\overline{x}_1, \overline{y}_1)$, both formulas $\mathsf{mostComp}\,(\epsilon)$ and $\mathsf{mostInf}\,(\epsilon)$ can be seen as queries with free variables $\overline{x}_1, \overline{y}_1$. We write these queries as follows: $\mathsf{mostComp}\,(\epsilon)(\overline{x}_1, \overline{y}_1)$, and $\mathsf{mostInf}\,(\epsilon)(\overline{x}_1, \overline{y}_1)$. However, to simplify the notation, whenever it is possible we will omit to explicitly reference variables.

7.2 Isomorphisms

Once the $\mathsf{mostInf}\,()$ formula associated with each expansion has been derived, we are able to select all maximal witness blocks in the canonical universal solution. We know however that this is not sufficient. In fact, it is still possible that some of these witness blocks are isomorphic to each other.

Handling isomorphic witness blocks by means of expansions is a tricky issue. In fact, there are two possible sources of isomorphisms among witness blocks. The first one corresponds to isomorphic copies of a witness block that are generated by different expansions. This is the easiest one to capture. However, there is also the possibility that isomorphic witness blocks are generated by the same expansion. As it was noted in [31], this may happen if an expansion has non-trivial automorphisms.

Consider for example a tgd that has an expansion as the following:

$$\epsilon = R^1(x_1, Y_1) \wedge R^2(x_2, Y_1)$$

it can be seen that $\chi(\overline{x}, \overline{y})$ has a non-trivial automorphism that maps x_1 into x_2 and vice-versa. Therefore, the expansion might select pairs of isomorphic witness blocks in J of the form $\{R(1, N_0), R(2, N_0)\}$ and $\{R(2, N_1), R(1, N_1)\}$. These are difficult to detect and remove.

This problem does not arise if we make the assumption that J is *isomorphism–free*, i.e., it does not contain witness blocks that are isomorphic to each other.

Definition 18 ***[Isomorphism-Free Solution]*** *Given a scenario $\mathcal{M}$, a source instance I, and a solution $J \in \mathsf{Sol}_{\mathcal{M}}(I)$, we say that J is* isomorphism–free *if there exist no witness blocks $w, w' \in \mathcal{W}^{<I,J>}$ such that $w \cong w'$.*

Let us assume for now that, given a scenario $\mathcal{M}$ and a source instance I, we have generated an isomorphism-free solution. In the following sections, we shall discuss how it is possible to achieve this goal. In the meanwhile, we notice that one simple way to do this is to generate J by running the standard chase procedure instead of the naive one. The semantics of the standard chase, in fact, prevents the generation in the canonical solution of any form of isomorphisms.

Based on this assumption, we are ready to introduce our alternative characterization of the core based on expansions.

Theorem 2 *Given a scenario $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st})$, a source instance I, call J a canonical universal solution of Σ_{st} over I. If J is isomorphism–free, consider the set of expansions* $\mathsf{expansions}(\mathcal{M})$ *and, for each expansion*

$$\epsilon = \chi^l(\overline{x}_1, \overline{y}_1) \wedge \exists \overline{x}_2, \overline{y}_2 : (\psi^l(\overline{x}_2, \overline{y}_2) \bigwedge \mathrm{EQUAL}_{h^f{}_\epsilon}(\overline{x}_1, \overline{x}_2))$$

its rewriting, $\mathsf{mostInf}\,(\epsilon)$*. The following set:*

$$\mathcal{E}^J_{\mathsf{mostInf}} = \bigcup_{\epsilon \in \mathsf{expansions}(\mathcal{M})} \{a(\chi^l(\overline{x}_1, \overline{y}_1)) \mid a \text{ s.t. } J \models a(\mathsf{mostInf}\,(\epsilon)(\overline{x}_1, \overline{y}_1))\}$$

is such that:

$$\mathcal{E}^J_{\mathsf{mostInf}} = \mathsf{reduce}(\mathsf{mostInformative}(\mathsf{mostCompact}(\mathcal{W}^{<I,J>})))$$

The full proof is reported in the Appendix. From Theorem 1 it follows that $\mathcal{E}^J_{\mathsf{mostInf}}$ is exactly the core of J.

Theorem 2 suggests a natural core-computation strategy for a scenario $\mathcal{M}$, based on the following steps:

- given I, generate an isomorphism-free canonical solution J (for example by running the standard chase of the tgds in Σ_{st});
- for each expansion $\epsilon \in \mathsf{expansions}(\mathcal{M})$, generate the formula $\mathsf{mostInf}\,(\epsilon)$, and run the corresponding queries on J to select the maximal witness blocks;
- copy these witness blocks to a new instance, J_0, to generate the core universal solution for $\mathcal{M}$ and I.

Notice that the last two steps can be performed by chasing the following set of full tgds, one for each expansion $\epsilon \in \mathsf{expansions}(\mathcal{M})$, over J:

$$\forall \overline{x}_1, \overline{y}_1 : \mathsf{mostInf}\,(\epsilon)(\overline{x}_1, \overline{y}_1) \rightarrow \chi(\overline{x}_1, \overline{y}_1) \tag{2}$$

No new null value needs to be invented in this process, since we are only copying to J_0 witness blocks that are already in J. This is, in fact, the strategy proposed in [24].

A disadvantage of this strategy is that it requires a two-step process, i.e., it assumes that two different exchanges are executed: the first is needed to generate J, and the second to select inside J the witness blocks that belong to the core. In [31] it was shown that this two-step approach is not necessary, i.e., it is possible to produce a rewriting that achieves the same goal by running a single exchange. Our goal is therefore to refine the expansion-based strategy in order to compute the core within a single exchange.

This will be the subject of the next sections.

8 Rewriting Expansions over the Source

An expansion is a formula over the target schema. However, in this section we show that it is possible to rewrite it in terms of source relations. We call this the *source rewriting* of the expansion. While an expansion is a query to select atoms in the target instance, its source rewriting states a "precondition" over the source for the existence of these atoms.

8.1 Source Rewritings

The strategy to rewrite expansions over the source is quite straightforward, thanks to our labeling technique. In fact, an expansion is a conjunction of labeled atoms taken from the tgd conclusions. To each of these atoms we may associate a *premise*, i.e., the left-hand side of the respective tgd. By joining the premises of all of its atoms we obtain the source rewriting of an expansion. Notice how our provenance system plays a central role during this step: in fact, by looking at the label of each atom of an expansion, we immediately know the tgd it comes from, and therefore its premise.

Definition 19 ***[Premise of an Atom and of a Formula]*** *Given a tgd* $m. \forall \overline{x} : \phi(\overline{x}) \rightarrow \exists \overline{y}(\psi^l(\overline{x}, \overline{y}))$*, and an atom* $R^l(\overline{x}_i, \overline{y}_i) \in \psi^l(\overline{x}, \overline{y})$*, its* premise*,* $\mathsf{premise}\,(R^l(\overline{x}_i, \overline{y}_i))$*, is the formula* $\phi(\overline{x})$*.*

Given a conjunctive formula, $\chi^l(\overline{x}_1, \overline{y}_1)$*, its* premise *is the formula:*

$$\mathsf{premise}\,(\chi^l(\overline{x}_1, \overline{y}_1)) = \bigwedge \{\mathsf{premise}\,(R_i^l(\overline{x}_i, \overline{y}_i)) \mid R_i^l(\overline{x}_i, \overline{y}_i) \in \chi^l(\overline{x}_1, \overline{y}_1)\}$$

Definition 20 ***[Source Rewriting]*** *Given a tgd* $m. \phi(\overline{x}_2) \rightarrow \exists \overline{y}_2(\psi^l(\overline{x}_2, \overline{y}_2))$ *and an expansion* $\epsilon = \chi^l(\overline{x}_1, \overline{y}_1) \wedge \exists \overline{x}_2, \overline{y}_2 : (\psi^l(\overline{x}_2, \overline{y}_2) \bigwedge \text{EQUAL}_{hf_\epsilon}(\overline{x}_1, \overline{x}_2))$ *in* $\mathsf{expansions}_{\mathcal{M}}(m)$*, its* source rewriting*,* $\mathsf{sourceRew}\,(\epsilon)$*, is the following formula:*

$$\mathsf{sourceRew}\,(\epsilon) = \mathsf{premise}\,(\chi^l(\overline{x}_1, \overline{y}_1)) \wedge \exists \overline{x}_2 : (\phi(\overline{x}_2) \bigwedge \text{EQUAL}_{hf_\epsilon}(\overline{x}_1, \overline{x}_2))$$

Notice that, while an expansion ϵ of a tgd m can be seen as a query $\epsilon(\overline{x}_1, \overline{y}_1)$ whose free variables include both universally and existentially quantified variables of m, its source rewriting, $\mathsf{sourceRew}\,(\epsilon)$, is on the contrary a query $\mathsf{sourceRew}\,(\epsilon)(\overline{x}_1)$ in which all the free variables are universally quantified variables of m.

Consider the scenario in Example 5.1:

$$\begin{array}{l} m_1.\ \forall x_1, x_2 : A(x_1, x_2) \rightarrow \exists Y_1 : S^1(x_1, Y_1) \wedge T^2(Y_1, x_2) \\ m_2.\ \forall x_3, x_4 : B(x_3, x_4) \rightarrow S^3(x_3, x_4) \\ m_3.\ \forall x_5, x_6 : C(x_5, x_6) \rightarrow T^4(x_5, x_6) \\ m_4.\ \forall x_7 : D(x_7) \rightarrow \exists Y_0 : S^5(x_7, Y_0) \end{array}$$

Tgd m_1 has the following expansions:

$$\begin{array}{rcl} \epsilon_{11} & = & S^1(x_1, Y_1) \wedge T^2(Y_1, x_2) \ \ (\textit{the base expansion}) \\ \epsilon_{12} & = & S^3(x_3, x_4) \wedge T^4(x_5, x_6) \wedge x_4 = x_5 \wedge \exists Y_1 : (S^1(x_3, Y_1) \wedge T^2(Y_1, x_6)) \end{array}$$

Their source rewritings are as follows:

$$\begin{array}{rcl} \mathsf{sourceRew}\,(\epsilon_{11}) & = & A(x_1, x_2) \\ \mathsf{sourceRew}\,(\epsilon_{12}) & = & B(x_3, x_4) \wedge C(x_5, x_6) \wedge x_4 = x_5 \wedge \exists x_1, x_2 : \\ & & \quad (A(x_1, x_2) \wedge x_3 = x_1 \wedge x_6 = x_2) \\ & = & B(x_3, x_4) \wedge C(x_5, x_6) \wedge x_4 = x_5 \wedge (A(x_3, x_6)) \end{array}$$

Similarly for tgd m_4:

$$\begin{array}{rcl} \epsilon_{41} & = & S^5(x_7, Y_0) \ \ (\textit{the base expansion}) \\ \epsilon_{42} & = & S^1(x_1, Y_1) \wedge \exists Y_0 : (S^5(x_1, Y_0)) \\ \epsilon_{43} & = & S^3(x_3, x_4) \wedge \exists Y_0 : (S^5(x_3, Y_0)) \end{array} \qquad \begin{array}{rcl} \mathsf{sourceRew}\,(\epsilon_{41}) & = & D(x_7) \\ \mathsf{sourceRew}\,(\epsilon_{42}) & = & A(x_1, x_2) \wedge (D(x_1)) \\ \mathsf{sourceRew}\,(\epsilon_{43}) & = & B(x_3, x_4) \wedge (D(x_3)) \end{array}$$

Consider the scenario in Example 5.2:

$$\begin{array}{l} m_1.\ A(x_1, x_2, x_3) \rightarrow \exists Y_1, Y_2 : S^1(x_1, Y_1, Y_2) \wedge S^2(x_2, x_3, Y_2) \\ m_2.\ B(x_4, x_5) \rightarrow \exists Y_3, Y_4 : S^3(x_4, x_5, Y_3) \wedge S^4(Y_4, x_5, Y_3) \end{array}$$

Here are some expansions and their source rewritings:

$$\begin{array}{rcl} \epsilon_{11} & = & S^1(x_1, Y_1, Y_2) \wedge S^2(x_2, x_3, Y_2) \\ \epsilon_{12} & = & S^2(x_2, x_3, Y_2) \wedge \exists Y_1 : (S^1(x_2, Y_1, Y_2)) \\ \epsilon_{13} & = & S^3(x_4, x_5, Y_3) \wedge \exists Y_1, Y_2 : (S^1(x_4, Y_1, Y_2) \wedge S^2(x_4, x_5, Y_2)) \end{array}$$

$$\begin{array}{rcl} \mathsf{sourceRew}\,(\epsilon_{11}) & = & A(x_1, x_2, x_3) \\ \mathsf{sourceRew}\,(\epsilon_{12}) & = & A(x_1, x_2, x_3) \wedge x_1 = x_2 \\ \mathsf{sourceRew}\,(\epsilon_{13}) & = & B(x_4, x_5) \wedge (A(x_4, x_4, x_5)) \end{array}$$

8.2 Adding Negations to Source Rewritings

Given an expansion ϵ, its source rewriting, $\mathsf{sourceRew}(\epsilon)$ tells us a precondition for the generation of all of its witness blocks. However, it should be clear at this point that we are not interested in all of the witness blocks associated with an expansion. We rather want to select only the maximal ones with respect to the partial orders $\prec$ and $<$. We therefore need to generate new expressions that give us preconditions for the set of most compact and most informative witness blocks associated with an expansion, respectively.

In order to do this, given ϵ, we now introduce the formulas $\mathsf{sourceRew}(\mathsf{mostComp}(\epsilon))$ and $\mathsf{sourceRew}(\mathsf{mostInf}(\epsilon))$, that are analogous to $\mathsf{mostComp}(\epsilon)$ and $\mathsf{mostInf}(\epsilon)$, but are rewritten on the source. To generate $\mathsf{sourceRew}(\mathsf{mostComp}(\epsilon))$, the intuition is again that whenever expansion ϵ' is more compact than ϵ, we add to the source rewriting of ϵ the negation of $\mathsf{sourceRew}(\epsilon')$.

Definition 21 ***[Rewriting Source Expansions:*** $\mathsf{sourceRew}(\mathsf{mostComp}())$*] Given a scenario $\mathcal{M}$, its set of expansions* $\mathsf{expansions}(\mathcal{M})$*, their rewritings* $\mathsf{mostComp}(\mathcal{M})$*, for each expansion ϵ in* $\mathsf{expansions}(\mathcal{M})$*, the formula* $\mathsf{sourceRew}(\mathsf{mostComp}(\epsilon))$ *is obtained as follows:*

- *initialize* $\mathsf{sourceRew}(\mathsf{mostComp}(\epsilon)) = \mathsf{sourceRew}(\epsilon)$*;*
- *for any expansion ϵ' in* $\mathsf{expansions}(\mathcal{M})$ *such that ϵ' is more compact than ϵ, call $h^f{}_c$ the compacting homom. of $\chi^l(\overline{x}_1, \overline{y}_1)$ into $\chi'^l(\overline{x}'_1, \overline{y}'_1)$; add to* $\mathsf{sourceRew}(\mathsf{mostComp}(\epsilon))$ *a formula*

$$\wedge\neg\exists\overline{x}'_1 : (\mathsf{sourceRew}(\epsilon') \bigwedge \text{EQUAL}_{h^f{}_c}(\overline{x}_1, \overline{x}'_1))$$

Definition 22 ***[Rewriting Source Expansions:*** $\mathsf{sourceRew}(\mathsf{mostInf}())$*] Given a scenario $\mathcal{M}$, its set of expansions* $\mathsf{expansions}(\mathcal{M})$*, their rewritings* $\mathsf{mostInf}(\mathcal{M})$*, for each expansion ϵ in* $\mathsf{expansions}(\mathcal{M})$*, the formula* $\mathsf{sourceRew}(\mathsf{mostInf}(\epsilon))$ *is obtained as follows:*

- *initialize* $\mathsf{sourceRew}(\mathsf{mostInf}(\epsilon)) = \mathsf{sourceRew}(\mathsf{mostComp}(\epsilon))$*;*
- *for any expansion ϵ' in* $\mathsf{expansions}(\mathcal{M})$ *such that ϵ' is more informative than ϵ, call $h^f{}_i$ the proper homomorphism of $\chi^l(\overline{x}_1, \overline{y}_1)$ into $\chi'^l(\overline{x}'_1, \overline{y}'_1)$; add to* $\mathsf{sourceRew}(\mathsf{mostInf}(\epsilon))$ *a formula:*

$$\wedge\neg\exists\overline{x}'_1 : (\mathsf{sourceRew}(\mathsf{mostComp}(\epsilon)') \bigwedge \text{EQUAL}_{h^f{}_i}(\overline{x}_1, \overline{x}'_1))$$

Consider, again, Example 5.1 above. Since we know that ϵ_{12} is more compact than ϵ_{11}, the formula $\mathsf{sourceRew}(\mathsf{mostComp}(\epsilon_{11}))$ will be as follows:

$$\begin{aligned}\mathsf{sourceRew}(\mathsf{mostComp}(\epsilon_{11})) \quad &= A(x_1, x_2) \wedge \neg\exists x'_3, x'_4, x'_5, x'_6 : (B(x'_3, x'_4) \wedge C(x'_5, x'_6) \wedge \\ &\quad x'_4 = x'_5 \wedge (A(x'_3, x'_6) \wedge x_1 = x'_3 \wedge x_2 = x'_6)) \\ &= A(x_1, x_2) \wedge \neg\exists x'_4, x'_5 : (B(x_1, x'_4) \wedge C(x'_5, x_2) \wedge x'_4 = x'_5)\end{aligned}$$

In this specific case, $\mathsf{sourceRew}(\mathsf{mostInf}(\epsilon_{11})) = \mathsf{sourceRew}(\mathsf{mostComp}(\epsilon_{11}))$, since there is no expansion that is more informative than ϵ_{11}.

9 The Rewriting Algorithm

We are now ready to introduce the actual rewriting algorithm. The algorithm takes as input a set of s-t tgds, Σ_{st}, and generates a set of FO-rules that is a core schema-mapping for Σ_{st}.

The algorithm works through several steps, that we list here and discuss in the following subsections. A comprehensive example will be discussed in Section 11.

Step 1 *Generating Expansions*: given a mapping scenario $\mathcal{M} = \{\mathbf{S}, \mathbf{T}, \Sigma_{st}\}$, as a first preliminary step for each tgd $m \in \Sigma_{st}$, we generate the set of expansions $\mathsf{expansions}_{\mathcal{M}}(m)$ using Algorithm 1 in Section 6; the union of these sets of expansions gives us the set of all expansions of $\mathcal{M}$, $\mathsf{expansions}(\mathcal{M})$;

Step 2 *Partial Orders Among Expansions*: as a second preliminary step, we analyze expansions in $\mathsf{expansions}(\mathcal{M})$ and find their formula homomorphisms, and generate the more-compact partial order, $\prec$, and the more-informative partial order, $<$, among expansions, as discussed in Section 7;

Step 3 *Source Rewritings of Expansions*: then, we rewrite each expansion ϵ in $\mathsf{expansions}(\mathcal{M})$ as a formula over the source database by computing its *source rewriting*, $\mathsf{sourceRew}(\epsilon)$, as discussed in Section 8.1;

Step 4 *Adding Negations to Source Rewritings*: since we are interested in finding preconditions for the maximal witness blocks associated with an expansion, we use the partial orders among expansions to generate the formulas $\mathsf{sourceRew}(\mathsf{mostComp}(\epsilon))$ and $\mathsf{sourceRew}(\mathsf{mostInf}(\epsilon))$, as discussed in Section 8.2.

The set of $\mathsf{sourceRew}(\mathsf{mostInf}(\epsilon))$ will represent premises for the final set of FO-rules. In order to complete the generation of the rules, we need to perform a number of additional steps, namely:

Step 5 *Choosing a Skolemization Strategy*: during this phase, we pick up an appropriate Skolemization strategy in order to replace existentially quantified variables in rule conclusions by Skolem terms;

Step 6 *Generating Expansion Rules*: based on the results of the previous steps, this step generates a preliminary set of FO-rules, called *expansion rules*, one for each expansion;

Step 7 *Normalizing Expansion Rules*: as a final step, the set of expansion rules generated at Step 6 is further rewritten in order to guarantee that the final rules are normalized.

These steps are discussed in the next subsections.

9.1 [Step 5] Skolemization Strategy

The main intuition behind the generation of the final set of FO-rules is to generate a rule for each expansion, ϵ, in which $\mathsf{sourceRew}(\mathsf{mostInf}(\epsilon))$ represents the left-hand side, and $\chi(\overline{x}_1, \overline{y}_1)$ the right-hand side. In fact, $\mathsf{sourceRew}(\mathsf{mostInf}(\epsilon))$ tells us under which conditions the set of maximal witness blocks associated with ϵ belong to the core, while $\chi(\overline{x}_1, \overline{y}_1)$ actually generates the witness blocks into the target.

Before formalizing this idea in the next Subsection, we need to discuss our skolemization strategy. This is in fact crucial in order to properly handle the presence of multiple isomorphic witness blocks.

Notice, in fact, that there is a significant difference between the two–step strategy based on the full tgds in Formula 2, and the single exchange we are trying to develop here. While the two-step approach selects witness blocks that are already in the canonical solution, and therefore only need to be "copied" to generate the core, here we are performing a single exchange, through which we need to generate the witness blocks that belong to the core, by properly inventing labeled nulls.

In order to do that, we rely on a *skolemization strategy*, skol, i.e., a transformation that takes a formula with existential variables and replaces them by Skolem terms.

Definition 23 ***[Skolemization Strategy]*** *Given a formula $\varphi(\overline{x}, \overline{y})$ with universal variables $\overline{x}$, and existential variables $\overline{y}$, a* skolemization strategy *is a mapping* skol *that associates with each existential variable $y_i \in \overline{y}$ a Skolem term $f_{\varphi,y_i}(\overline{x})$.*

There are several possible skolemization strategies to pick from. The standard strategy, skol_{std}, would be that of associating a different, uninterpreted Skolem function f_{m,Y_i} with each existential variable Y_i of a tgd m, and taking as arguments all universal variables occurring in the conclusion. However, there are alternatives to this scheme.

Consider the issue of isomorphisms among witness blocks. Recall that there may be two different sources of isomorphic witness blocks inside a solution. The first one is due to different rules that have isomorphic conclusions. The second one to a single rule that has non-trivial automorphisms. The standard skolemization strategy can be used to handle isomorphisms of the first kind, but it is hopeless in front of automorphisms of a rule conclusion.

Our solution to the problem is to use *interpreted* Skolem functions rather than uninterpreted ones, and design them in such a way that all isomorphic copies of a witness block collapse into one.

More specifically, we introduce a notion of *isomorphism–invariant* skolemization strategy. In order to do this, we compare the result of a given skolemization with that of the standard strategy. More specifically, given a formula $\varphi(\overline{x}, \overline{y})$, and an assignment a to $\overline{x}$, we call $a(\mathsf{skol}(\varphi(\overline{x}, \overline{y})))$ the instance of $\varphi(\overline{x}, \overline{y})$ generated as follows: (a) first replace each existential variable $y_i \in \overline{y}$ by the corresponding Skolem term; this gives a new formula $\varphi'(\overline{x})$ that depends on $\overline{x}$ only; (b) then, generate the set of facts $a(\varphi'(\overline{x}))$. We call $a(\mathsf{skol}_{std}(\varphi(\overline{x}, \overline{y})))$ the *standard instance* of the formula.

Definition 24 ***[Isomorphism–Invariant Skolemization Strategy]*** *A skolemization strategy,* skol*, is* isomorphism–invariant *if:*

- *given a formula,* $\varphi(\overline{x}, \overline{y})$*, and an assignment* a*, then* $a(\mathsf{skol}(\varphi(\overline{x}, \overline{y})))$ *is isomorphic to* $a(\mathsf{skol}_{std}(\varphi(\overline{x}, \overline{y})))$*;*
- *given two formulas,* $\varphi(\overline{x}, \overline{y})$*,* $\varphi'(\overline{x}', \overline{y}')$*, and two assignments* a, a'*, if the standard instances* $a(\mathsf{skol}_{std}(\varphi(\overline{x}, \overline{y})))$*,* $a'(\mathsf{skol}_{std}(\varphi'(\overline{x}', \overline{y}')))$ *are isomorphic, then* $a(\mathsf{skol}(\varphi(\overline{x}, \overline{y}))) = a'(\mathsf{skol}(\varphi'(\overline{x}', \overline{y}')))$*.*

The definition above essentially states two requirements: (i) skol should be such that it generates formula instances, i.e., sets of facts, that are *compatible* with those that would be generated by the standard skolemization; (ii) at the same time, whenever the standard skolemization would generate distinct, isomorphic formula instances, skol on the contrary generates identical sets of facts.

In the following subsections, we shall assume that a suitable skolemization strategy, skol, has been chosen for expansion rules. Then, in Section 10, we discuss how this can be done in practice.

9.2 [Step 6] Generating Expansion Rules

We are now ready to introduce our final rewriting. In the following, we assume that a Skolemization strategy has been fixed. Based on that, we introduce the notion of *expansion rules* for a scenario $\mathcal{M}$. In essence, for each expansion ϵ we build a rule that uses the formula $\mathsf{sourceRew}\,(\mathsf{mostInf}\,(\epsilon))$ – in which only universally quantified variables appear – as a premise, and $\mathsf{skol}(\chi(\overline{x}_1, \overline{y}_1))$ as a conclusion:

Definition 25 ***[Expansion Rules]*** *Fixed a skolemization strategy,* skol*, and an expansion* $\epsilon = \chi^l(\overline{x}_1, \overline{y}_1) \wedge \exists \overline{x}_2, \overline{y}_2 : (\psi^l(\overline{x}_2, \overline{y}_2) \bigwedge \textsc{equal}_{h^f{}_\epsilon}(\overline{x}_1, \overline{x}_2))$ *in* $\mathsf{expansions}(\mathcal{M})$ *its* expansion rule*,* $\mathsf{expansionRule}\,(\epsilon)$*, is the following formula:*

$$\mathsf{expansionRule}\,(\epsilon) = \forall \overline{x}_1 : \mathsf{sourceRew}\,(\mathsf{mostInf}\,(\epsilon))(\overline{x}_1) \rightarrow \mathsf{skol}(\chi(\overline{x}_1, \overline{y}_1))$$

The set of expansion rules *of a scenario* $\mathcal{M}$ *is the set:*

$$\Sigma^{exp}_{\mathcal{M},\mathsf{skol}} = \{\mathsf{expansionRule}\,(\epsilon) \mid \epsilon \in \mathsf{expansions}(\mathcal{M})\}$$

As discussed in the previous sections, a straightforward optimization consists of generating expansion rules only for expansions whose set of atoms is a core.

Consider Example 5.1. Given expansion:

$$\epsilon_{11} \quad = \quad S^1(x_1, Y_1) \wedge T^2(Y_1, x_2)$$

i.e., the base expansion of tgd m_1, we have shown in Section 8 that:

$$\mathsf{sourceRew}\,(\mathsf{mostInf}\,(\epsilon_{11})) = \quad A(x_1, x_2) \wedge \neg \exists x'_4, x'_5 : (B(x_1, x'_4) \wedge C(x'_5, x_2) \wedge x'_4 = x'_5)$$

Then, we generate the rule:

$$r_1.\ \forall x_1, x_2 : A(x_1, x_2) \wedge \neg(\exists x'_4, x'_5 : B(x_1, x'_4) \wedge C(x'_5, x_2) \wedge x'_4 = x'_5) \rightarrow \\ S(x_1, f_{sk}(x_1, x_2)) \wedge T(f_{sk}(x_1, x_2), x_2).$$

Notice how we have skolemized the rule using simple, uninterpreted Skolem terms. In fact, the rewritten rules never generate isomorphic witness blocks.

9.3 [Step 7] Normalizing and Rewriting Expansion Rules

Expansion rules are very close to the final core schema-mapping we need to generate. However, there is a final issue we need to introduce. As we discussed in Section 5, witness blocks do not need to coincide with fact blocks. More specifically, a witness block may be in some cases a fact block of the core solution (i.e., a connected component of the dual Gaifman graph of the core), but it may also be the union of portions of fact blocks, whose facts are joined over constants instead of nulls.

This feature of witness blocks has a direct counterpart in their expansions. In fact, it is easy to see that expansions do not need to be normalized. Recall that we say that a formula is normalized if its dual Gaifman graph is connected. When used as rule conclusions, non-normalized expansions may generate unnecessary facts in the target, i.e., facts that have already been generated by other expansions. To prevent this, once the expansion rules have been generated, we need to normalize them.

More specifically, given $\Sigma^{exp}_{\mathcal{M},\mathsf{skol}}$, we generate a new set of rules:

$$\mathsf{normalize}(\Sigma^{exp}_{\mathcal{M},\mathsf{skol}})$$

We call any new rule obtained as a result of this normalization step, i.e., any rule in $\mathsf{normalize}(\Sigma^{exp}_{\mathcal{M},\mathsf{skol}})$ that was not in $\Sigma^{exp}_{\mathcal{M},\mathsf{skol}}$, a *decomposed rule*.

Consider the scenario in Example 5.3. Here are two expansions in $\mathsf{expansions}(\mathcal{M})$ (we report them partially to simplify the example):

$$\begin{aligned}\epsilon_a &= S^1(x_3,x_0,x_0,Y_0,x_1) \wedge S^2(Y_1,x_0,x_0,Y_0,x_0) \wedge \exists ...\\ \epsilon_b &= S^1(x_3,x_0,x_0,Y_0,x_1) \wedge S^1(x_3,x'_0,x'_0,Y'_0,x'_1) \wedge \exists ...\end{aligned}$$

Of these expansions, ϵ_b is not normalized (the two atoms do not join on an existential variable, but rather on x_3). Therefore, when we generate the final expansion rules, we end up with a set of non-normalized rules:

$$\begin{aligned}&r_a.\ \mathsf{sourceRew}\,(\mathsf{mostInf}\,(\epsilon_a)) \rightarrow \mathsf{skol}(S(x_3,x_0,x_0,Y_0,x_1) \wedge S(Y_1,x_0,x_0,Y_0,x_0))\\ &r_b.\ \mathsf{sourceRew}\,(\mathsf{mostInf}\,(\epsilon_b)) \rightarrow \mathsf{skol}(S(x_3,x_0,x_0,Y_0,x_1) \wedge S(x_3,x'_0,x'_0,Y'_0,x'_1))\end{aligned}$$

We therefore apply the normalize procedure introduced in Section 4 in order to normalize rule r_b. This generates the following set of rules, with two new decomposed rules:

$$\begin{aligned}&r_a.\ \mathsf{sourceRew}\,(\mathsf{mostInf}\,(\epsilon_a)) \rightarrow \mathsf{skol}(S(x_3,x_0,x_0,Y_0,x_1) \wedge S(Y_1,x_0,x_0,Y_0,x_0))\\ &r_{b1}.\ \mathsf{sourceRew}\,(\mathsf{mostInf}\,(\epsilon_b)) \rightarrow \mathsf{skol}(S(x_3,x_0,x_0,Y_0,x_1))\\ &r_{b2}.\ \mathsf{sourceRew}\,(\mathsf{mostInf}\,(\epsilon_b)) \rightarrow \mathsf{skol}(S(x_3,x'_0,x'_0,Y'_0,x'_1))\end{aligned}$$

By doing this, however, we may end up with different, unnecessary rules, that generate portions of witness blocks. As a consequence, once the normalization has been performed, some further processing is needed. In our example, rule r_a generates a witness block that is also a fact-block. On the contrary, r_{b1} and r_{b2} generate fragments of the same witness block. As a consequence, we need to fire r_{b1}, r_{b2} only as long as a more-informative atom is not generated by r_a.

In order to handle non-normalized blocks, we look for proper homomorphisms among rule conclusions. In order to do that, we extend the notion of formula homomorphism to skolemized formulas, by considering Skolem terms as existentially quantified variable. Then, we introduce a further rewriting, as follows:

Definition 26 ***[Final Rewriting:*** $\mathsf{finalRew}\,()$***]*** *For each rule* $r \in \mathsf{normalize}(\Sigma^{exp}_{\mathcal{M},\mathsf{skol}})$*, the rule* $\mathsf{finalRew}\,(r)$ *is obtained as follows:*

- *if* $r \in \Sigma^{exp}_{\mathcal{M},\mathsf{skol}}$*, i.e., r is not a decomposed rule, then* $\mathsf{finalRew}\,(r) = r$*;*
- *otherwise, if r is a decomposed rule of the form r.* $\phi(\overline{x}) \rightarrow \psi(\overline{x})$*:*
  - *initialize* $\mathsf{finalRew}\,(r) = r$*;*

– *for any rule* r'. $\phi'(\overline{x}') \rightarrow \psi'(\overline{x}')$ *in* $\mathsf{normalize}(\Sigma^{exp}_{\mathcal{M},\mathsf{skol}})$ *such that* $\psi'(\overline{x}')$ *is more informative than* $\psi(\overline{x})$ *according to homomorphism* $h^f{}_i$, *add to the premise of* $\mathsf{finalRew}\,(r)$ *a formula*

$$\wedge\neg\exists\overline{x}' : (\phi'(\overline{x}') \bigwedge \text{EQUAL}_{h^f{}_i}(\overline{x}, \overline{x}'))$$

This gives us a new set of FO-rules, obtained as follows:

$$\Sigma^{core}_{\mathcal{M},\mathsf{skol}} = \{\mathsf{finalRew}\,(r) \mid r \in \mathsf{normalize}(\Sigma^{exp}_{\mathcal{M},\mathsf{skol}})\}$$

9.4 Computing the Core

We are now ready to introduce our main result.

Theorem 3 *Given a scenario* $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st})$ *and an isomorphism-invariant skolemization strategy,* skol, $\Sigma^{core}_{\mathcal{M},\mathsf{skol}}$ *is a core schema mapping for* $\mathcal{M}$.

The full proof is in the Appendix.

Based on Theorem 3, our approach to the problem of generating core universal solutions in a scalable way is the following. Given a scenario $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st})$, we apply the 7-step rewriting algorithm introduced in the previous sections to generate the associated core schema-mapping, $\Sigma^{core}_{\mathcal{M},\mathsf{skol}}$. Then, from this set of FO-rules, we generates a runtime SQL script that can be executed on source instances to efficiently generate core solutions.

Section 11 develops a complete example that covers all the steps in the algorithm.

10 Skolemization

There are several ways in which it is possible to build an isomorphism–invariant strategy for a tgd or an FO-rule. One solution is to consider the dual Gaifman graph of the tgd conclusion. Recall that the dual Gaifman graph of a formula is an undirected graph in which nodes are atoms, and there is an edge between atoms $R_i(\overline{x}_i, \overline{y}_i)$ and $R_j(\overline{x}_j, \overline{y}_j)$ if there is some existential variable y_k occurring in both atoms. We may design an isomorphism–invariant Skolem function by returning strings that encode the structure of the graph. More specifically, we embed in the Skolem string a full description of the Gaifman graph, in terms of constants, in such a way that whenever two formulas generate isomorphic blocks the strings coincide.

Consider for example the following formula:

$$S(x_0, Y_0), T(x_1, Y_0, Y_1), W(Y_1, Y_2)$$

The Skolem functions for Y_0 and Y_1 will have three arguments:

- a description of the graph nodes – i.e., of the tuples generated by the formula – in terms of constants;
- an encoding of the graph edges, i.e., of the joins associated with common existential variables;
- a reference to the specific existential variable for which the function is used. Notice that each existential variable that appears more than once in the formula – like Y_0 or Y_1 – corresponds to one of the joins; on the contrary, existential variables that occur only once are not related to joins.

In the following we report the actual Skolem strings that are needed in order to properly replace the existential variables. We assume that these strings are generated by an interpreted Skolem function that appends constant strings and variable

values. Notice how we use symbols t_i, j_i, v to refer to tuples, i.e., nodes in the graph, joins, i.e., arcs in the graph, and variables respectively:

$$\begin{aligned}
\mathsf{skol}(Y_0) &= f_{sk}(t1 : S[A : x_0], t2 : T[A : x_1], t3 : W[\,], \\
&\qquad j1 : [t1.B = t2.B], j2 : [t2.C = t3.A], v : j1) \\
\mathsf{skol}(Y_1) &= f_{sk}(t1 : S[A : x_0], t2 : T[A : x_1], t3 : W[\,], \\
&\qquad j1 : [t1.B = t2.B], j2 : [t2.C = t3.A], v : j2) \\
\mathsf{skol}(Y_2) &= f_{sk}(t1 : S[A : x_0], t2 : T[A : x_1], t3 : W[\,], \\
&\qquad j1 : [t1.B = t2.B], j2 : [t2.C = t3.A], v : \mathit{noj}\text{-}t3.B)
\end{aligned}$$

Notice that, differently from standard Skolem terms, we are using a *global* function symbol, f_{sk}, rather than different symbols for each tgd. The main intuition behind this strategy is that tgd conclusions that may generate isomorphic blocks are essentially identical up to the renaming of nulls. Therefore, they have isomorphic dual Gaifman graphs, and the strings generated by such encodings are identical.

At the same time, the skolemization strategy is compatible with the standard one. In fact, for each tgd m and existential variable Y, it generates a Skolem term that is associated with m, since it encodes the structure of the relative graph, and Y, and that depends on the universal variables that appear in the conclusion of m.

An important point here is that set elements in the Skolem string must be encoded in lexicographic order, so that the functions generate appropriate values regardless of the order in which atoms appear in the rule conclusion. This last requirement introduces further subtleties in the way exchanges with self-joins are handled. In fact, note that in formulas like the one above – in which all relation symbols in the conclusion are distinct – the order of set elements can be established at script generation time (they depend on relation names). If, on the contrary, the same atom may appear more than once in the conclusion, then things change.

Consider for example the following formula, that has a non-trivial automorphism:

$$S(x_0, Y_0), S(x_1, Y_0)$$

In this case, for Y_0 a function of this form would be generated:

$$f_{sk}(t1 : S[A : x_0], t2 : S[A : x_1], j1 : [t1.B = t2.B], v : j1)$$

It can be seen that, by assignments $x_0 = 0, x_1 = 1$ and $x_0 = 1, x_1 = 0$ the function would generate two different strings, and therefore the two blocks would not be collapsed. To avoid this, we sort the tuples at execution time based on the actual assignment of values to the variables:

$$\mathsf{skol}(Y_0) = f_{sk}(\mathit{sort}(S[A : x_0], S[A : x_1]), j1 : [S.B = S.B], v : j1)$$

in this way, on both assignments the same Skolem string is generated:

$$f_{sk}(t1 : [S(A : 0)], t2 : [S(A : 1)], j1 : [S.B = S.B], v : j1)$$

and the two blocks are collapsed. To generate these strings, we therefore need a composition of append and sort functions.

To do this, we assume that there is a linear order on the constants. This is consistent with the linear–order requirement that was introduced in [31].

11 A Rewriting Example

This section is devoted to presenting an example of the rewriting algorithm. We will make reference to the following scenario:

$$m_1.\ R(x_0, x_1, x_2) \rightarrow \exists Y_0, Y_1, Y_2, Y_3 : S^1(Y_0, x_0, Y_1, Y_2) \wedge S^2(Y_0, x_1, x_2, Y_3)$$

While this scenario contains a single tgd, still it requires the application of the main steps of the rewriting algorithm. For an alternative, more complex example that covers all steps of the algorithm, see A.

[Step 1] Generating Expansions

As a first step, we generate all expansions of tgd m_1. In this case, our algorithm generates two expansions, as follows:

$$\begin{array}{rcl} \epsilon_1 & = & S^1(Y_0, x_0, Y_1, Y_2) \wedge S^2(Y_0, x_1, x_2, Y_3) \\ \epsilon_2 & = & S^2(Y_0, x_1, x_2, Y_3) \\ & & \exists Y_0' \ldots (S^1(Y_0', x_1, Y_1', Y_2') \wedge S^2(Y_0', x_1, x_2, Y_3')) \end{array}$$

Notice the difference between ϵ_1 and ϵ_2. While ϵ_2 generates witness blocks made of a single S^2 tuple that self-joins in order to cover the tuples that would be generated by S^1, ϵ_1 on the contrary uses two different tuples S^1 and S^2, that join on the first attribute.

[Step 2] Partial Orders Among Expansions

Once expansions have been derived, we analyze their formula homomorphisms, in order to build the more-compact and more-informative partial orders.

In this case, we discover that there are two formula homomorphisms among expansions. In brief:

$$\begin{array}{ll} \textit{more compact:} & \epsilon_1 \prec \epsilon_2 \\ \textit{more informative:} & \epsilon_2 < \epsilon_1 \end{array}$$

[Step 3] Source Rewritings of Expansions

We now need to rewrite expansions as formulas over the source database. The source rewritings are as follows:

$$\begin{array}{rcl} \mathsf{sourceRew}\,(\epsilon_1) & = & R(x_0, x_1, x_2) \\ \mathsf{sourceRew}\,(\epsilon_2) & = & R(x_0, x_1, x_2) \wedge R(x_1, x_1, x_2) \end{array}$$

[Step 4] Adding Negations to Source Rewritings

For each expansion ϵ_i, based on $\mathsf{sourceRew}\,(\epsilon_i)$, we need to generate the two expressions, $\mathsf{sourceRew}\,(\mathsf{mostComp}\,(\epsilon_i))$, and $\mathsf{sourceRew}\,(\mathsf{mostInf}\,(\epsilon_i))$. Let us first concentrate on $\mathsf{mostComp}\,()$. We consider the compacting homomorphisms found at Step 2, and introduce negations accordingly. We first notice that:

$$\mathsf{sourceRew}\,(\mathsf{mostComp}\,(\epsilon_2)) \quad = \quad \mathsf{sourceRew}\,(\epsilon_2)(\overline{x})$$

since there are no expansions that are more compact than ϵ_2. Consider now ϵ_1; there exists one formula homomorphism $h^f{}_{12}$ such that ϵ_2 is more compact than ϵ_1; as a consequence, $\mathsf{sourceRew}\,(\mathsf{mostComp}\,(\epsilon_1))$ has the following form:

$$\mathsf{sourceRew}\,(\mathsf{mostComp}\,(\epsilon_1)) \quad = \quad R(x_0, x_1, x_2) \wedge \neg \exists x_0' : R(x_0', x_1, x_2) \wedge R(x_1, x_1, x_2)$$

Let us now consider $\mathsf{mostInf}\,()$. There is only one proper formula homomorphism to be taken into account. We therefore have:

$$\begin{array}{rcl} \mathsf{sourceRew}\,(\mathsf{mostInf}\,(\epsilon_1)) & = & \mathsf{sourceRew}\,(\mathsf{mostComp}\,(\epsilon_1)) \\ \mathsf{sourceRew}\,(\mathsf{mostInf}\,(\epsilon_2)) & = & \mathsf{sourceRew}\,(\mathsf{mostComp}\,(\epsilon_2)) \wedge \\ & & \neg \exists \overline{x}' : (\mathsf{sourceRew}\,(\mathsf{mostComp}\,(\epsilon_1)) \wedge \textsc{equal}_{h^f{}_{21}}(\overline{x}, \overline{x}')) \\ & = & R(x_0, x_1, x_2) \wedge R(x_1, x_1, x_2) \wedge \\ & & \neg(\exists x_0' : R(x_0', x_1, x_2) \wedge \neg \exists x_0'' : R(x_0'', x_1, x_2) \wedge R(x_1, x_1, x_2)) \end{array}$$

[Step 5] Choosing a Skolemization Strategy

We now need to choose a proper skolemization strategy for this example, in order to guarantee that isomorphic witness blocks are properly collapsed by our rewritten rules. Since there are no expansions generating isomorphic witness blocks, our algorithm picks up the standard skolemization strategy that relies on uninterpreted Skolem functions.

[Step 6] Generating Expansion Rules

We are ready to generate our expansion rules. We generate one expansion rule for each expansion ϵ_i. The premise is represented by the formula $\mathsf{sourceRew}\,(\mathsf{mostInf}\,(\epsilon_i))$; the conclusion by χ_i, with the appropriate Skolem terms to replace existential variables, as follows:

$$\begin{array}{ll} r_1 : \mathsf{sourceRew}\,(\mathsf{mostInf}\,(\epsilon_1)) \rightarrow & \mathsf{skol}(S(Y_0, x_0, Y_1, Y_2) \wedge S(Y_0, x_1, x_2, Y_3)) \\ r_2 : \mathsf{sourceRew}\,(\mathsf{mostInf}\,(\epsilon_2)) \rightarrow & \mathsf{skol}(S(Y_0, x_1, x_2, Y_3)) \end{array}$$

Here we report some of the actual Skolem terms generated by skol.

$$\begin{array}{lcl} \mathsf{skol}(r_1, Y_0) & = & f(t1 : S[B : x_0], t2 : S[B : x_1, C : x_2], j1 : [t1.A = t2.A], v : j1) \\ \mathsf{skol}(r_1, Y_1) & = & f(t1 : S[B : x_0], t2 : S[B : x_1, C : x_2], j1 : [t1.A = t2.A], v : \textit{noj-t1.C}) \\ \mathsf{skol}(r_2, Y_0) & = & f(t1 : S[B : x_1, C : x_2], , v : \textit{noj-t1.A}) \end{array}$$

Since in this example all rules are normalized, Step 7 "Normalizing and Rewriting Expansion Rules" is not needed. This set of rules is the core schema mapping for the given scenario.

12 Complexity

A few comments are worth making here on the complexity of the rewriting algorithm. Recall that our goal is to execute the rewritten rules under the form of SQL scripts; in the scripts, source rewritings of expansions give rise to joins, and negated atoms give rise to difference operators. Generally speaking, joins and differences are executed very efficiently by the DBMS. However, the number of joins and differences needed to filter out redundant tuples largely depends on the nature of the scenario.

12.1 Scenarios with No Self-Joins

It is important to make a distinction between scenarios with self-joins in tgd conclusions and scenarios that do not have self-joins. Recall that we say that a scenario $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st})$ has *self-joins in tgd conclusions* (or simply self-joins) if the same relation symbol appears more than once in the conclusion of a tgd in Σ_{st}. The scenarios in Examples 5.2, 5.3, 5.4 have self-joins in tgd conclusions. The scenario in Example 5.1 does not have self-joins.

In [24], an alternative rewriting algorithm to generate core schema mappings for scenarios without self-joins was given. The algorithm can be seen as a special case of the one described in this paper, with two significant exceptions: (a) it considers a much smaller search space for expansions; (b) it generates a lower number of tgds in the final rewriting.

In this section, we want to justify that algorithm in the framework of the one introduced in this paper. In fact, the complexity of the final script generated by the algorithm described in Section 9.4 is strongly influenced by the number of expansions of a tgd conclusion that must be generated. Generally speaking, an expansion for a tgd whose conclusion has k atoms may be any multiset of atoms in $\bigcup\{\psi_i(x_i, y_i)\}$, of size k or less. This gives an upper bound on the number of expansions in $\mathsf{expansions}(\mathcal{M})$ that is clearly exponential with respect to the size of $\bigcup\{\psi_i(x_i, y_i)\}$. Also, among these expansions it is necessary to look for compacting homomorphisms first, and then for proper homomorphisms. Finally, each expansion generates a new dependency.

However, if we restrict our attention to scenarios that do not have self-joins, things improve significantly. In fact, we have the following result.

Theorem 4 *Given a scenario $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st})$, suppose Σ_{st} does not contain self-joins in tgd conclusions. Given a source instance I, call J a canonical universal solution for $\mathcal{M}$ over I, and J_0 the core universal solution for $\mathcal{M}$ over I. Then:*

- *for any fact block b_f in J, either all tuples in b_f belong also to J_0, or none of them does;*
- *for each tgd $m \in \Sigma_{st}$ whose conclusion has size k, all witness blocks in $\mathcal{W}_m^{<I,J>}$ have size exactly k.*

The proof is in the Appendix. Based on Theorem 4, we can infer several conclusions. First, if a tgd whose conclusion has size k does not have self-joins, then it may only have *fixed-size expansions*, i.e., sets of atoms of size exactly k. In fact, it is easy to see that it is not necessary to consider multisets of atoms in $\bigcup\{\psi_i(x_i, y_i)\}$ (no atom may appear more than once, since there are no self-joins), and that exactly one atom in the expansion is needed to "cover" each of the k distinct atoms in the tgd conclusion. This significantly reduces the number of expansions for a given tgd.

Second, since, according to Theorem 4, the core is the union of a subset of fact blocks in J, it is possible to see that we only need to discover which fact blocks to keep and which to discard. Since the tgds in Σ_{st} are normalized by hypothesis, fact blocks correspond to instances of the base expansions. Therefore, we adopt the following strategy:

- for each tgd, we generate only fixed-size expansions;
- then, we concentrate on the base expansion alone;
- we find compacting and proper homomorphisms of the base expansion into all other expansions, in order to remove unnecessary witness blocks from the result;
- finally, we generate one FO-rule for each base expansion in order to produce the result, and disregard rules for other expansions.

This significantly reduces the final number of rules.

Based on these ideas, we find it useful to classify the relevant homomorphisms among expansions in a scenario without self-join in two categories. We call a *subsumption* any compacting or proper homomorphism of a base expansion into a base expansion. We call a *coverage* any compacting or proper homomorphism of a base expansion into a fixed-size expansion that uses atoms of different tgd conclusions. Consider the tgds in Example 5.1. There is a subsumption of the base expansion of m_4, $S(x_7, Y_0)$ by the base expansion of m_2, $S(x_3, x_4)$. There is a coverage of the base expansion of m_1, $S(x_1, Y_1)$, $T(Y_1, x_2)$, by the union of atoms $S(x_3, x_4)$, $T(x_5, x_6)$. It should be apparent from the discussion above that, if a scenario does not have self-joins, then subsumptions and coverages are the only relevant forms of homomorphisms that must be taken into account in order to generate the core.

12.2 Complexity Bounds

As a first remark, let us note that subsumptions are handled more efficiently than coverages. Consider the graph of the subsumption relation among tgd conclusions, obtained by removing transitive edges. In the worst case – the graph is a path – there are $O(n^2)$ subsumptions. However, this is rather unlikely in real scenarios. Typically, the graph is broken into several smaller connected components, and the number of subsumptions is linear in the number of tgds. This means that only a linear number of differences will be introduced in the final SQL script.

The worst-case complexity of the rewriting is higher for coverages, for two reasons. First, coverages always require to perform additional joins before computing the actual difference, since they reuse atoms from different tgd conclusions. Second, and more important, if we call k the number of atoms in a tgd, and assume that each atom can be mapped into n other atoms via homomorphisms, then we need to generate n^k different coverages, and therefore n^k differences.

This exponential bound on the number of coverages is not surprising. In fact, Gottlob and Nash have shown that the problem of computing core solutions is *fixed-parameter intractable* [18] wrt the size of the tgds (in fact, wrt the size of blocks), and therefore it is very unlikely that the exponential bound can be removed. We want to emphasize however that we are talking about expression complexity and not data complexity (the data complexity remains polynomial).

Despite this important difference in complexity between subsumptions and coverages, coverages can usually be handled quite efficiently. In brief, the exponential bound is reached only under rather unlikely conditions; to see why, recall that coverages tend to follow the pattern shown above. Note that m_2 and m_3 write into the key–foreign key pair, while m_1 invents a labeled null. Complexity may become an issue, here, only if the set of tgds contains a significant number of other tgds like m_2 and m_3 which write into S and T separately. This may happen only in those scenarios in which a very large number of different data sources with a poor design of foreign key relationships must be merged into the same

target, which can hardly be considered as a frequent case. In fact, in our experiments with both real-life scenarios and large randomly generated schemas, coverages have never been an issue.

Computing times are usually higher for scenarios with self-joins in tgd conclusions. In fact, the exponential bound is more severe in these cases. Given a scenario $\mathcal{M} = \{\mathbf{S}, \mathbf{T}, \Sigma_{st}\}$, we call:

- n the total number of atoms in all tgd conclusions, i.e., the cardinality of the set $\mathcal{R} = \bigcup_i \psi_i(\overline{x}_i, \overline{y}_i)$;
- k the maximum number of atoms in a tgd conclusion, i.e., $k = max(\{|\psi_i(\overline{x}_i, \overline{y}_i)|\})$.

Recall that, in order to generate expansions for $\mathcal{M}$, we need to generate the set $\mathcal{R}_k^{pow}$. Recall that the number of multisets of size k or less of n elements is as follows:

$$\left(\binom{n}{k}\right) = \binom{n+k-1}{k} = \binom{n+k-1}{n-1}$$

Since, to generate all possible candidate expansions, we need to consider multisets of size k or less, we have that:

$$|\mathcal{R}_k^{pow}| = \Sigma_{i=1}^{k} \binom{n+k-1}{n-1}$$

In order to estimate the complexity of finding expansions for $\mathcal{M}$, we need to generate $\mathcal{R}_k^{pow}$, the set of all multisets of atoms in $\mathcal{R}$ of size k or less. This set may contain an exponentially large number of candidate expansions, and therefore force the algorithm to check an equally large number of formula homomorphisms among them. This, however, is done off-line, i.e., at script-generation time. Also, we are speaking about expression complexity, not data complexity, i.e., the data complexity of the final script remains polynomial.

Despite this, the exponential blowup in the number of expansions may also have direct impact at script-execution time. To see this, let's call m_k a tgd with k atoms in its conclusion. Consider the worst case in which each element in $\mathcal{R}_k^{pow}$ is a valid expansion for m_k, and therefore generates a negated subformula in the rewriting. It can be easily seen that the number of joins, intersections and differences in the final SQL script would be exponentially high with respect to k. In fact, it is not difficult to design synthetic scenarios that actually trigger such exponential explosion. How it is shown in our experiments, as the size of the source database increases, this may bring to very high computing times.

However, it is important to emphasize that, in more realistic scenarios containing self-joins, the overhead is usually much lower. To understand why, let us note that expansions tend to increase when tgds are designed in such a way that it is possible for a tuple to perform a join with itself. In practice, this happens very seldom. Consider for example a $Person(name, father)$ relation, in which children reference their father. It can be seen that no tuple in the *Person* table actually joins with itself. Similarly, in a $Gene(name, type, protein)$ table, in which "synonym" genes refer to their "primary" gene via the protein attribute, since no gene is at the same time a synonym and a primary gene.

In light of these ideas, we may say that, while it is true that the rewriting algorithm may generate expensive queries, this happens only in rather specific cases that hardly reflect practical scenarios. In practice, scalability is very good. In fact, we may say that the 90% of the complexity of the algorithm is needed to address a small minority of the cases. Our experiments confirm this intuition.

13 Experimental Results

The algorithms introduced in the paper have been implemented in the working prototype of the +Spicy [25] system, written in Java. In this section, we first study the performance of the SQL scripts generated by our rewriting algorithm on mapping scenarios of various kinds and sizes. We show that the rewriting algorithm efficiently computes the core, even for large databases and complex scenarios. Then, we study the scalability of the rewriting algorithm with respect to synthetic scenarios of increasing complexity. We show that the algorithm scales with respect to a large number of relations and joins. All experiments have been executed on a Intel Core 2 Duo machine with 2.4Ghz processor and 4 GB of RAM under Linux. The DBMS was PostgreSQL 8.3.

13.1 Execution Times for Core Computation

We selected a set of nine experiments to evaluate the execution times of the SQL scripts generated using our algorithms. The nine scenarios are divided in two groups. The first group includes two scenarios with subsumptions only (s_1, s_2) and two with subsumptions and coverages (c_1, c_2). The second group is composed of five scenarios with self-joins (sj_1–sj_5). The scenarios were taken from the literature (s_2 and sj_3 from [13], sj_2 from [33]), and from variants of the basic scenarios in STBenchmark [1]. Among the scenarios with self-joins, sj_4 and sj_5 are the ones with automorphisms in rule conclusions: sj_4 is a variant of Example 5.2, while sj_5 has been designed to artificially trigger the exponential complexity of the algorithm. In sj_5, ten tgds with the same target symbol repeated 25 times and eight three-way self-joins have been used. On average we had 7 tables, with a minimum of 2 and a maximum of 10.

Computing Times for Large Source Instances To study how the algorithm performs on databases of large sizes, we ran every scenario with five different source instances of 10k, 100k, 250k, 500k, and 1M tuples, respectively. We start by comparing our algorithm with an implementation [30] of the core computation algorithm developed in [18], made available to us by the authors. In the following we will refer to this implementation as the "post-processing approach".

A first evidence is that the post processing approach does not scale. We have been able to run experiments with 1k and 5k tuples, but starting at around 10k tuples the experiments took on average several hours. This result is not surprising, since these algorithms exhaustively look for endomorphisms in the canonical solution in order to remove variables (i.e, invented nulls). For instance, our first subsumption scenario with 5k tuples in the source generated 13500 variables in the target; the post-processing algorithm took on our machine running PostgreSQL around 7 hours to compute the final solution. It is interesting to note that in some cases the post processing algorithm finds the core after only one iteration (in the previous case, it took 3 hours), but the algorithm is not able to recognize this fact and stop the search. For all experiments, we fixed a timeout of 1 hour. If the experiment was not completed by that time, it was stopped. Since none of the scenarios we selected was executed in less than 1 hour we do not report computing times for the post-processing algorithm in our graphs.

To put these results in perspective we need to mention that the algorithms implemented by the core-computation engine we are considering are significantly more general than the ones in this paper. In fact, they allow to handle scenarios with arbitrary target dependencies, i.e., target tgds and egds. The poor performance registered in our tests is strictly related to the generality of the approach, which requires complex recursive computations.

Still, we believe there is some interesting evidence in this comparison. In fact, coherently with the approach followed in this paper, in our tests we have only considered mapping scenarios without target constraints. Our experimental results show that in this specific case our SQL-based approach performs much better. Since we were interested in

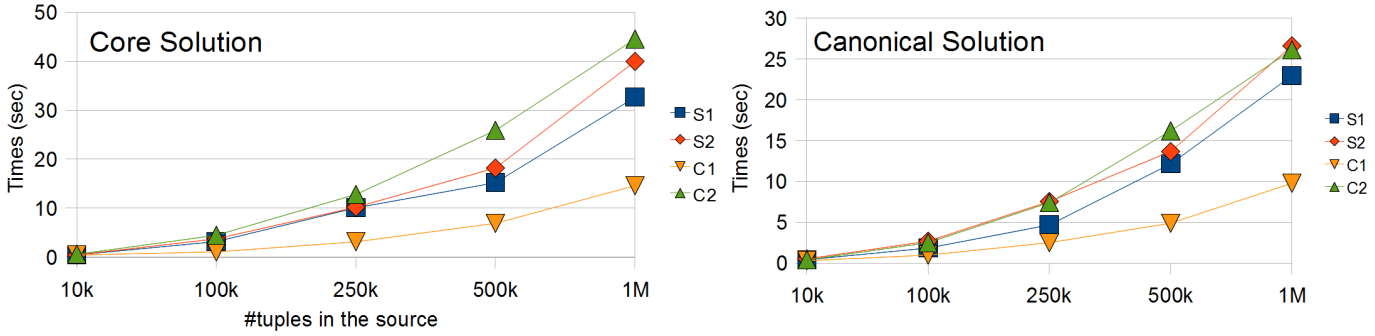


Figure 2: SQL Scripts: Execution Times for the First Group

comparing execution times of the scripts for core solutions to those of scripts for canonical solutions, we ran the two sets of scripts over the same scenarios and reported the results in Figure 2 and Figure 3. Figure 2 shows execution times

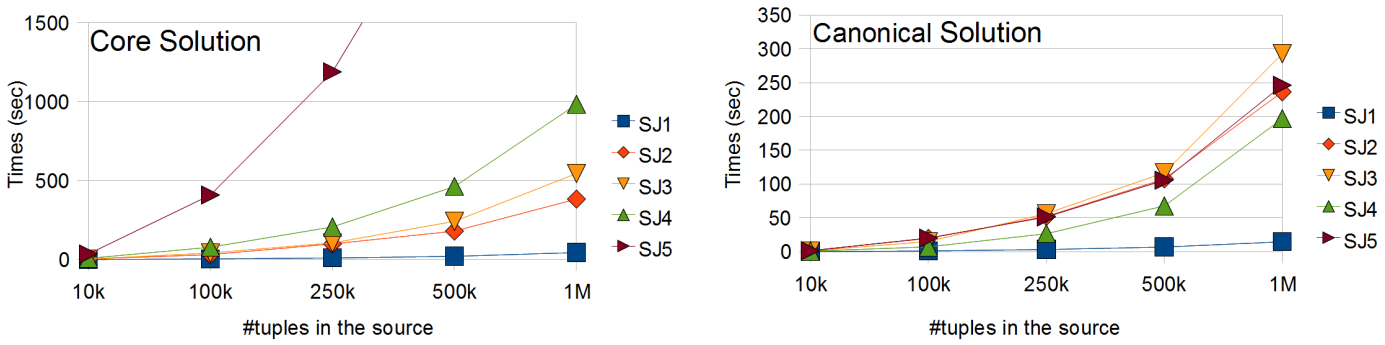


Figure 3: SQL Scripts: Execution Times for the Second Group

for the four scenarios that do not contain self-joins in the target. As it can be seen, execution times for all scenarios were extremely fast for both configurations. The overhead introduced by the rewriting of the FO-rules using negations is always acceptable, with a maximum of around 10 seconds for scenarios of one million tuples.

Figure 3 reports the results for the five scenarios with self-joins. It can be seen that the first three self-joins scenarios, sj_1 – sj_3, show times increasing linearly and did scale up to 1M tuples both in the core and in the canonical scripts executions. The difference is instead notable with sj_4 and sj_5, but is not surprising for two reasons. First, considering that many self-joins can trigger the exponential behavior discussed in the previous Section. Second, the running time to interpret the Skolem functions fills some of the overhead time. For these reasons, the core computation script for sj_4 took up to four times the canonical script execution time (21 minutes for the 1 million tuples source instance), while we stopped the execution for sj_5 on the biggest input (the core script took 41 minutes for the 500k tuples source instance).

Quality of Solutions We now want to study to which extent core universal solutions are more compact than canonical solutions. To do this, we consider source databases with different degrees of "redundancy". We dropped sj_5 from this comparison. For each of the remaining eight scenarios, we generated five synthetic source instances of fixed size (10K tuples) based on a pool of values of decreasing size. This process generated different levels of redundancy (from 0% to 40%) in the source databases and enabled a comparison of the quality of the two solutions. Figure 4 shows the percent

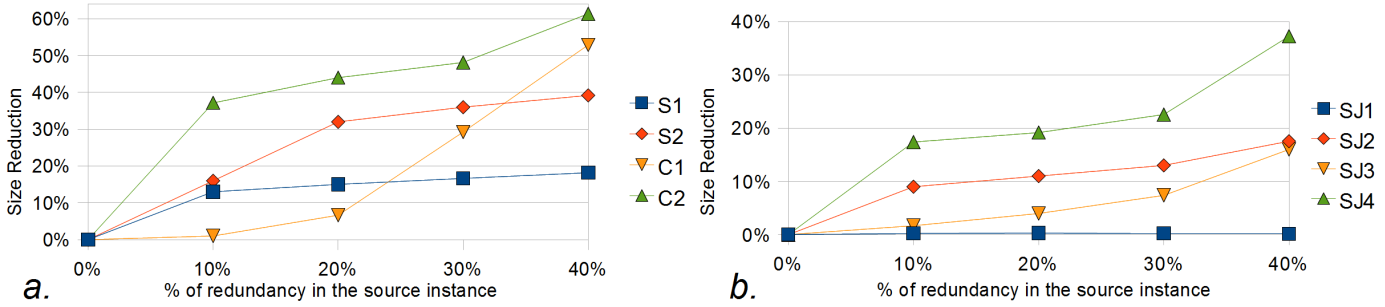


Figure 4: Core vs Canonical: Size Reduction in Solutions

reduction in the output size for core solutions compared to canonical solutions. As output size, we measured the number of tuples in the solutions. Figure 4.a shows results for the four scenarios that do not contain self-joins in the target. As expected, core solutions are more compact than canonical ones in all the scenarios and this behavior becomes more apparent with the increasing redundancy. The two subsumptions scenarios – s_1 and s_2 – follow the trend, but less

significantly than the two coverage scenarios c_1 and c_2. This is not surprising, since the design of the tgds in s_1 and s_2 tend to generate many duplicate tuples in the solutions, and those are removed both by the core script and the canonical one. Figure 4.b reports the percent reductions for the four self-join scenarios. Again, core solutions are more compact than canonical ones in all the scenarios except sj_1. This is also expected, since sj_1 is a full mapping and no Skolem nor null values are generated in the solution, i.e. canonical and core solution coincide.

13.2 Algorithm Scalability on Large Scenarios

To test the scalability of our algorithm on schemas of large size we generated a set of synthetic scenarios using the scenario generator developed for the STBenchmark. We generated four relational scenarios containing 20/50/75/100 tables, with an average join path length of 3, variance 1. Note that, to simulate real-application scenarios, we did not include self-joins. To generate complex schemas we used a composition of basic cases with an increasing number between 1 and 15, in particular we used: Vertical Partitioning (3/6/11/15 repetitions), Denormalization (3/6/12/15), and Copy (1 repetition). With such settings we got schemas varying between 11 relations with 3 joins and 52 relations with 29 joins. Figure 5

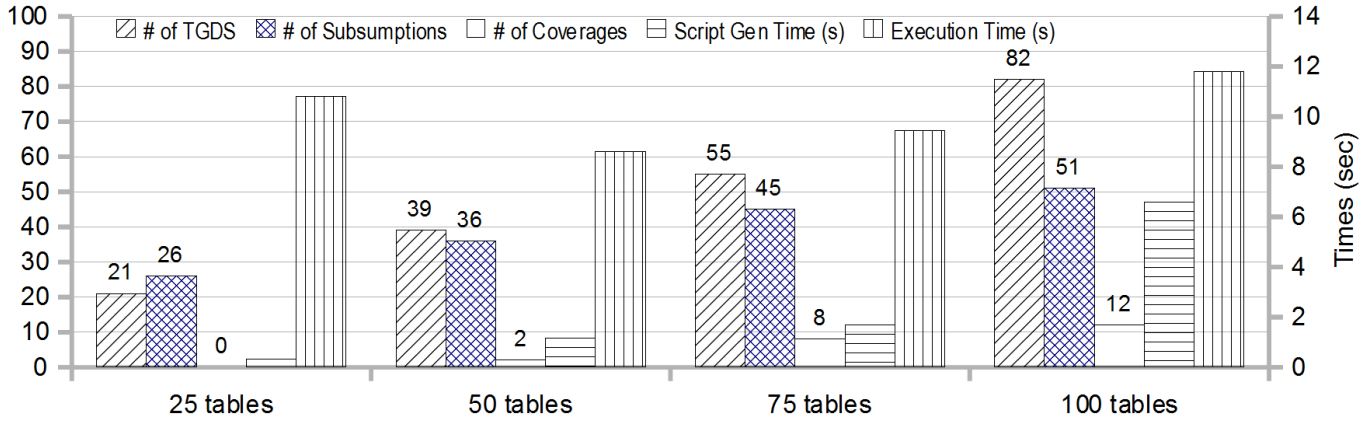


Figure 5: Algorithm scalability with large synthetic scenarios

summarizes the results. In the graph, we report several values. One is the number of tgds processed by the algorithm, with the number of subsumptions and coverages. Then, since we wanted to study how the tgd rewriting phase scales on large schemas, we measured the time needed to generate the SQL script. In all cases the algorithm was able to generate the SQL script in a few seconds. Finally, we report execution times in seconds for source databases of 100K tuples.

14 Related Work

In this section we review some related works in the fields of schema mappings and data exchange.

The original schema mapping generation algorithm was introduced in [26, 27] in the framework of the Clio project. The algorithm relies on a nested relational model to handle relational and XML data. The primary inputs are value correspondences and foreign key constraints on the two sources that are chased to build tableaux called logical relations; a tgd is produced for each source and target logical relations that cover at least one correspondence. The tgd generation algorithm we use in our system is a generalization of the basic mapping algorithm that captures a larger class of mappings, like self-joins [1] or those in [2]. Note that the need for explicit joins was first advocated in [29].

The amount of redundancy generated by basic mappings has motivated a revision of the algorithm known as *nested mappings* [17]. Intuitively, whenever a tgd m_1 writes into an external target set R and a tgd m_2 writes into a set nested

into R, it is possible to "merge" the two mappings by nesting m_2 into m_1. This reduces the amount of redundant tuples in the target. Another attempt to reduce the redundancy generated by basic mappings has been proposed by [7]. Unfortunately, these approaches are applicable only in some specific cases and do not represent a general solution to the problem of generating core universal solutions.

The notion of a core solution was first introduced in [13]; it represents a nice formalization of the notion of a "minimal" solution, since cores of finite structures arise in many areas of computer science (see, for example, [20]). Note that computing the core of an arbitrary instance is an intractable problem [13, 18]. However, we are not interested in computing cores for arbitrary instances, but rather for solutions of a data exchange problem; these show a number of regularities, so that polynomial-time algorithms exist.

In [13] the authors first introduce a polynomial greedy algorithm for computing the core of universal solutions, and then a *blocks* algorithm. A block is a connected component in the Gaifman graph of nulls. The block algorithm looks at the nulls in J and computes the core of J by successively finding and applying a sequence of small *useful* endomorphisms; here, *useful* means that at least one null disappears. Only egds are allowed as target constraints.

The bounds are improved in [18]. The authors introduce various polynomial algorithms to compute the core of universal solutions in the presence of weakly-acyclic target tgds and arbitrary egds, that is, a more general framework than the one discussed in this paper. The authors prove two complexity bounds. Using an exhaustive enumeration algorithm they get an upper bound of $O(vm|dom(J)|^b)$, where v is the number of variables in J, m is the size of J, and b is the block size of J. There exist cases where a better bound can be achieved by relying on hypertree decomposition techniques. In such cases, the upper bound is $O(vm^{[b/2]+2})$, with special benefits if the target constraints of the data exchange scenario are LAV tgds. One of the algorithms introduced [18] has been revised and implemented in a working prototype [30]. The prototype uses a relational DBMS to chase tgds and egds, and a specialized engine to find endomorphisms and minimize the universal solution. Unfortunately, as discussed in Section 13, the technique does not scale to large size databases.

The problem of computing the core for a set of s-t tgds using SQL queries has been recently studied in [9, 24, 31]. [9] represents an early approach at computing core solutions for schema mappings specified by the limited class of s-t tgds with single atomic formulas (without repetition of existential quantified variables) in the conclusions.

The first proposal of an algorithm for rewriting a set of s-t tgds in order to generate core solutions was introduced in [24]. Differences between that paper and this one have been discussed in the previous sections. Here we want to note that the algorithm presented in [24] is the two-step algorithm outlined at the beginning of Section 9, and then further discussed in Section 12. In this paper, we show that it is possible to reduce the two-step process to a single exchange. Notice how this produces a speed-up in computing times: in fact, our rewriting algorithm produces SQL scripts that are significantly faster than those reported in the experiments of [24].

After the appearance of the initial algorithm in [24], another algorithm was independently proposed in [31]. The authors develop an algorithm to rewrite a set of s-t tgds as a *laconic* mapping, that is, a new set of dependencies from which to generate an SQL script that computes core solutions for the original scenario. There are several differences with the approach we propose in this paper.

As a first difference, both approaches rely on a linear order over the underlying domain in order to perform the rewriting. The linear order is necessary in order to break automorphisms in tgd conclusions. However, there is difference in the way these cases are handled by the two algorithms. The algorithm in [31] uses *side-conditions*, i.e., inequalities among universally quantified variable, while in our approach we sort values inside Skolem terms.

Notice that the algorithm proposed in [31] is more general than the one proposed in this paper, since it can be applied to dependencies that make use of arbitrary first-order formulas in the premises, and not only conjunctive formulas. This is done by relying on a procedure called *certain()*, to rewrite the certain answers of a query on the target as a query on the source. In the paper, the authors introduce a practical version of the algorithm in which *certain()* is computed by adopting a variant of the MiniCon [28] algorithm, which works for conjunctive formulas. They also announce [32] a more general algorithm that, given a scenario $\mathcal{M}$ and an FO target query q, computes a source query q' defining $certain_{\mathcal{M},q}()$.[3]

Second, in terms of dependencies generated by the rewriting, a laconic mapping tends to contain a lower number of

[3]Provided that the active domain of the source database contains at least two elements.

dependencies with more complex premises with respect to a core schema mapping, which typically contains more rules. In fact, as discussed in Section 5, laconic mappings reason on fact-blocks and fact-block types at a "global" level, while core schema mappings reason on witness blocks at a "local" level, i.e., at the tgd level.

Finally, with respect to the complexity of the rewriting algorithm, we notice that laconic mappings require to compute *certain()* many times – actually, a combinatorial number of times with respect to the size of the existential variables – for each fact-block type. This may be expensive, since computing *certain()* requires to run a high-complexity algorithm (e.g. MiniCon). Our algorithm, albeit exponential, looks for formula homomorphisms, whose number is typically lower than those that must be computed to generate laconic mappings. Also, in the case in which a scenario does not contain self-joins, based on Theorem 4, our algorithm does a minimal rewriting that is typically faster to compute.

At the moment it is unclear how these differences affect performances. In fact, no implementation of the laconic mappings algorithm is currently available, so that it was not possible to compare the performances of the two algorithms.

Both algorithms can be used as building blocks for more complex rewritings, such as the one introduced in [23].

We note that the notion of provenance in a schema mappings framework has been studied in [10] under the notion of *routes*. In this paper we make a more restricted use of the notion of provenance, in order to identify atoms in a canonical solution.

Finally, tools that somewhat resemble our expansions have been independently developed in papers about inverting schema-mappings. In [15], a notion of a *minimal generator* is introduced as a conjunct that identifies a possible contribution to a target symbol. The purpose is to identify how different sets of fact-blocks should be treated when inverting the mapping. In [3], a notion of an *existential replacement* is introduced as a tool to rewrite queries posed over the target of a mapping scenario as tools over the source. A novel exponential-time algorithm is developed to this end. We notice that, in our approach, the corresponding step of rewriting expansions as queries over the source is greatly simplified by our provenance-based labeling system.

15 Conclusions

We have introduced new algorithms to compute solutions for data exchange scenarios that show how, despite their intrinsic complexity, core solutions can be computed very efficiently in practical, real-life scenarios by using relational database engines.

We believe that this represents a concrete advancement towards the goal of adopting data exchange techniques in real-life mapping scenarios.

In this respect, there are several interesting research problems that are worth studying in order to further bridge the gap between the practice of schema mappings and the theory of data exchange. A relevant one is the development of similar algorithms to handle target constraints. Another one is the extension of the notion of data exchange setting to nested data. A final one is the revision of existing schema mapping benchmarks in order to explicitly incorporate the notion of a core solution as the "optimal" solution.

Acknowledgements

We would like to thank Vadim Savenkov and Reinhard Pichler who made available to us an implementation of their post-processing core-computation algorithm. Many thanks also go to Bruno Marnette and Paolo Atzeni. Finally, we are very grateful to Donatello Santoro for his precious comments on early drafts of the paper.

References

[1] B. Alexe, W. Tan, and Y. Velegrakis. Comparing and Evaluating Mapping Systems with STBenchmark. *Proc. of the VLDB Endowment*, 1(2):1468–1471, 2008.

[2] Y. An, A. Borgida, R.J. Miller, and J. Mylopoulos. A Semantic Approach to Discovering Schema Mapping Expressions. In *Proc. of ICDE*, pages 206–215, 2007.

[3] M. Arenas, J. Pérez, and C. Riveros. The Recovery of a Schema Mapping: Bringing Exchanged Data Back. *ACM TODS*, 34(4):1–48, 2009.

[4] C. Beeri and M.Y. Vardi. A Proof Procedure for Data Dependencies. *J. of the ACM*, 31(4):718–741, 1984.

[5] P. Bohannon, E. Elnahrawy, W. Fan, and M. Flaster. Putting Context into Schema Matching. In *Proc. of VLDB*, pages 307–318. VLDB Endowment, 2006.

[6] A. Bonifati, G. Mecca, A. Pappalardo, S Raunich, and G. Summa. Schema Mapping Verification: The Spicy Way. In *Proc. of EDBT*, pages 85 – 96, 2008.

[7] L. Cabibbo. On Keys, Foreign Keys and Nullable Attributes in Relational Mapping Systems. In *Proc. of EDBT*, pages 263–274, 2009.

[8] J. Cheney, L. Chiticariu, and W. C. Tan. Provenance in Databases: Why, How, and Where. *Foundations and Trends in Databases*, 1(4):379–474, 2009.

[9] L. Chiticariu. Computing the Core in Data Exchange: Algorithmic Issues. MS Project Report, 2005. Unpublished manuscript.

[10] L. Chiticariu and W. C. Tan. Debugging Schema Mappings with Routes. In *Proc. of VLDB*, pages 79–90, 2006.

[11] R. Fagin. Inverting Schema Mappings. *ACM TODS*, 32(4), 2007.

[12] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: Semantics and query answering. *Theor. Comput. Sci.*, 336(1):89–124, 2005.

[13] R. Fagin, P. G. Kolaitis, and L. Popa. Data Exchange: Getting to the Core. *ACM TODS*, 30(1):174–210, 2005.

[14] R. Fagin, P. G. Kolaitis, L. Popa, and W. Tan. Composing Schema Mappings: Second-Order Dependencies to the Rescue. *ACM TODS*, 30(4):994–1055, 2005.

[15] R. Fagin, P. G. Kolaitis, L. Popa, and W. Tan. Quasi-Inverses of Schema Mappings. *ACM TODS*, 33(2):1–52, 2008.

[16] R. Fagin, P.G. Kolaitis, A. Nash, and L. Popa. Towards a Theory of Schema-Mapping Optimization. In *Proc. of ACM PODS*, pages 33–42, 2008.

[17] A. Fuxman, M. A. Hernández, C. T. Howard, R. J. Miller, P. Papotti, and L. Popa. Nested Mappings: Schema Mapping Reloaded. In *Proc. of VLDB*, pages 67–78, 2006.

[18] G. Gottlob and A. Nash. Efficient Core Computation in Data Exchange. *J. of the ACM*, 55(2):1–49, 2008.

[19] G. Gottlob, R. Pichler, and V. Savenkov. Normalization and Optimization of Schema Mappings. *Proc. of the VLDB Endowment*, 2(1):1102–1113, 2009.

[20] P. Hell and J. Nešetřil. The Core of a Graph. *Discrete Mathematics*, 109(1-3):117–126, 1992.

[21] R. Hull and M. Yoshikawa. ILOG: Declarative creation and manipulation of object identifiers. In *Proc. of VLDB*, pages 455–468, 1990.

[22] A. Y. Levy, A. O. Mendelzon, Y. Sagiv, and D. Srivastava. Answering queries using views. In *PODS*, pages 95–104, 1995.

[23] B. Marnette, G. Mecca, and P. Papotti. Scalable Data-Exchange with Functional Dependencies. *Proc. of the VLDB Endowment*, 3(1):105–116, 2010.

[24] G. Mecca, P. Papotti, and S. Raunich. Core Schema Mappings. In *Proc. of ACM SIGMOD*, pages 655–668, 2009.

[25] G. Mecca, P. Papotti, S. Raunich, and M. Buoncristiano. Concise and Expressive Mappings with +Spicy. *Proc. of the VLDB Endowment*, 2(2):1582–1585, 2009.

[26] R. J. Miller, L. M. Haas, and M. A. Hernandez. Schema Mapping as Query Discovery. In *Proc. of VLDB*, pages 77–99, 2000.

[27] L. Popa, Y. Velegrakis, R. J. Miller, M. A. Hernandez, and R. Fagin. Translating Web Data. In *Proc. of VLDB*, pages 598–609, 2002.

[28] R. Pottinger and A. Halevy. MiniCon: A Scalable Algorithm for Answering Queries using Views. *VLDB J.*, 10(2-3):182–198, 2001.

[29] A. Raffio, D. Braga, S. Ceri, P. Papotti, and M. A. Hernández. Clip: a Visual Language for Explicit Schema Mappings. In *Proc. of ICDE*, pages 30–39, 2008.

[30] V. Savenkov and R. Pichler. Towards practical feasibility of core computation in data exchange. In *Proc. of LPAR*, pages 62–78, 2008.

[31] B. ten Cate, L. Chiticariu, P. Kolaitis, and W. C. Tan. Laconic Schema Mappings: Computing Core Universal Solutions by Means of SQL Queries. *Proc. of the VLDB Endowment*, 2(1):1006–1017, 2009.

[32] B. ten Cate and P. Kolaitis. Structural Characterizations of Schema-Mapping Languages. In *Proc. of ICDT*, pages 63–72, 2009.

[33] L. L. Yan, R. J. Miller, L. M. Haas, and R. Fagin. Data Driven Understanding and Refinement of Schema Mappings. In *Proc. of ACM SIGMOD*, pages 485–496, 2001.

A A Complex Rewriting Example

This section is devoted to presenting a complete example of application of the rewriting algorithm. We will make reference to the scenario in Example 5.3:

$$m_1.\ A(x_0,x_1,x_2,x_3) \wedge B(x_3,x_4) \rightarrow \exists Y_0,Y_1,Y_2,Y_3 : S^1(x_3,x_0,Y_0,x_1) \wedge \\ S^2(Y_1,x_0,Y_0,x_0) \wedge S^3(Y_1,x_2,Y_2,Y_3)$$

While this scenario contains a single tgd, still it requires the application of all steps of the rewriting algorithm.

[Step 1] Generating Expansions

As a first step, we generate all expansions of tgd m_1. In this case, our algorithm generates four expansions, as follows:

$$\begin{array}{rcl}
\epsilon_1 & = & S^1(x_3,x_0,Y_0,x_1) \wedge S^2(Y_1,x_0,Y_0,x_0) \wedge S^3(Y_1,x_2,Y_2,Y_3) \\
\epsilon_2 & = & S^1(x_3,x_0,Y_0,x_1) \wedge S^2(Y_1,x_0,Y_0,x_0) \wedge \\
 & & \exists Y_0'' \ldots (S^1(x_3,x_0,Y_0'',x_1) \wedge S^2(Y_1'',x_0,Y_0'',x_0) \wedge S^3(Y_1'',x_0,Y_2'',Y_3'')) \\
\epsilon_3 & = & S^1(x_3,x_0,Y_0,x_0) \wedge S^1(x_3,x_0',Y_0',x_1') \wedge \\
 & & \exists Y_0'' \ldots (S^1(x_3,x_0,Y_0'',x_0) \wedge S^2(Y_1'',x_0,Y_0'',x_0) \wedge S^3(Y_1'',x_0',Y_2'',Y_3'')) \\
\epsilon_4 & = & S^1(x_3,x_0,Y_0,x_0) \wedge \\
 & & \exists Y_0'' \ldots (S^1(x_3,x_0,Y_0'',x_0) \wedge S^2(Y_1'',x_0,Y_0'',x_0) \wedge S^3(Y_1'',x_0,Y_2'',Y_3''))
\end{array}$$

Notice the difference between ϵ_3 and ϵ_4. While ϵ_4 generates witness blocks made of a single S^1 tuple that self-joins twice in order to cover the tuples that would be generated by S^2 and S^3, ϵ_3 on the contrary uses two different S^1 tuples, that join on the first attribute. It is also possible to see, however, that ϵ_4 can be considered as a special case of ϵ_3, when $x_0 = x_0'$ and $x_0 = x_1'$, i.e., when the two tuples coincide. Therefore, all witness blocks generated by ϵ_4 are also generated by ϵ_3. We therefore discard ϵ_4 and in the following steps only consider $\epsilon_1, \epsilon_2, \epsilon_3$.

[Step 2] Partial Orders Among Expansions

Once expansions have been derived, we analyze their formula homomorphisms, in order to build the more-compact and more-informative partial orders.

In this case, we discover that there are several compacting formula homomorphisms among expansions. In brief:

$$\begin{array}{lllll}
\textit{more compact:} & \epsilon_1 \prec \epsilon_2 & \epsilon_1 \prec \epsilon_3 & \epsilon_2 \prec \epsilon_3 \\
\textit{more informative:} & \epsilon_2 < \epsilon_1
\end{array}$$

To be more specific, there are 2 different compacting homomorphisms of ϵ_2 into ϵ_3. To see this, notice that we can map the atoms of χ_2^l into those of χ_3^l as follows (for the sake of brevity we omit the mappings of variable occurrences): S^1 into the first occurrence of S^1 in χ_2^l, and S^2 into the second occurrence of S^1. But we also have an alternative compacting formula homomorphism by the opposite mapping: S^1 into the second occurrence of S^1, S^2 into the first occurrence. We need to consider both homomorphisms in our rewritings.

[Step 3] Source Rewritings of Expansions

We now need to rewrite expansions as formulas over the source database. The source rewritings are as follows:

$$\begin{array}{rcl}
\mathsf{sourceRew}\,(\epsilon_1) & = & A(x_0,x_1,x_2,x_3) \wedge B(x_3,x_4) \\
\mathsf{sourceRew}\,(\epsilon_2) & = & A(x_0,x_1,x_0,x_3) \wedge B(x_3,x_4) \wedge \exists x_4' : (A(x_0,x_1,x_0,x_3) \wedge B(x_3,x_4')) \\
 & = & A(x_0,x_1,x_0,x_3) \wedge B(x_3,x_4) \\
\mathsf{sourceRew}\,(\epsilon_3) & = & A(x_0,x_0,x_2,x_3) \wedge B(x_3,x_4) \wedge A(x_0',x_1',x_0',x_3) \wedge B(x_3,x_4') \wedge \\
 & & \exists x_4'' : (A(x_0,x_0,x_0',x_3) \wedge B(x_3,x_4''))
\end{array}$$

[Step 4] Adding Negations to Source Rewritings

For each expansion ϵ_i, based on $\mathsf{sourceRew}\,(\epsilon_i)$, we need to generate the two expressions, $\mathsf{sourceRew}\,(\mathsf{mostComp}\,(\epsilon_i))$, and $\mathsf{sourceRew}\,(\mathsf{mostInf}\,(\epsilon_i))$.

Let us first concentrate on $\mathsf{sourceRew}\,(\mathsf{mostComp}\,(\epsilon_i))$. We consider the compacting homomorphisms found at Step 2, and introduce negations accordingly. We first notice that:

$$\mathsf{sourceRew}\,(\mathsf{mostComp}\,(\epsilon_3)) \quad = \quad \mathsf{sourceRew}\,(\epsilon_3)$$

since there are no expansions that are more compact than ϵ_3. Consider now ϵ_1; there exists one formula homomorphism ${h^f}_{12}$ such that ϵ_2 is more compact than ϵ_1; as a consequence, $\mathsf{sourceRew}\,(\mathsf{mostComp}\,(\epsilon_1))$ has the following form:

$$\begin{array}{rcl} \mathsf{sourceRew}\,(\mathsf{mostComp}\,(\epsilon_1)) & = & A(x_0,x_1,x_2,x_3) \wedge B(x_3,x_4) \wedge \\ & & \neg\exists x'_4 : A(x_0,x_1,x_0,x_3) \wedge B(x_3,x'_4) \wedge \\ & & \neg\exists \ldots \textit{ negations of } \mathsf{sourceRew}\,(\epsilon_3) \end{array}$$

Similarly for ϵ_2. The complete formulas are too complex to be reported in full. In fact, from now on we shall discuss only fragments of the rewritten formulas.

Let us now consider $\mathsf{sourceRew}\,(\mathsf{mostInf}\,(\epsilon_i))$. There is only one proper formula homomorphism to be taken into account. We therefore have:

$$\begin{array}{rcl} \mathsf{sourceRew}\,(\mathsf{mostInf}\,(\epsilon_1)) & = & \mathsf{sourceRew}\,(\mathsf{mostComp}\,(\epsilon_1)) \\ \mathsf{sourceRew}\,(\mathsf{mostInf}\,(\epsilon_2)) & = & \mathsf{sourceRew}\,(\mathsf{mostComp}\,(\epsilon_2)) \wedge \\ & & \neg\exists \overline{x}' : (\mathsf{sourceRew}\,(\mathsf{mostComp}\,(\epsilon_1)) \wedge \textsc{equal}_{{h^f}_{21}}(\overline{x},\overline{x}')) \\ \mathsf{sourceRew}\,(\mathsf{mostInf}\,(\epsilon_3)) & = & \mathsf{sourceRew}\,(\mathsf{mostComp}\,(\epsilon_3)) \end{array}$$

[Step 5] Choosing a Skolemization Strategy

We now need to choose a proper skolemization strategy for this example, in order to guarantee that isomorphic witness blocks are properly collapsed by our rewritten rules. Since we notice that expansion ϵ_3 does generate isomorphic witness blocks, our algorithm picks up the most general, isomorphism–invariant skolemization strategy, that relies on interpreted Skolem functions.

[Step 6] Generating Expansion Rules

We are ready to generate our expansion rules. We generate one expansion rule for each expansion ϵ_i. The premise is represented by the formula $\mathsf{sourceRew}\,(\mathsf{mostInf}\,(\epsilon_i))$; the conclusion by χ_i, with the appropriate Skolem terms to replace existential variables, as follows:

$$\begin{array}{rcl} r_1 : \mathsf{sourceRew}\,(\mathsf{mostInf}\,(\epsilon_1)) \rightarrow & \mathsf{skol}(S(x_3,x_0,Y_0,x_1) \wedge S(Y_1,x_0,Y_0,x_0) \wedge S(Y_1,x_2,Y_2,Y_3)) \\ r_2 : \mathsf{sourceRew}\,(\mathsf{mostInf}\,(\epsilon_2)) \rightarrow & \mathsf{skol}(S(x_3,x_0,Y_0,x_1) \wedge S(Y_1,x_0,Y_0,x_0)) \\ r_3 : \mathsf{sourceRew}\,(\mathsf{mostInf}\,(\epsilon_3)) \rightarrow & \mathsf{skol}(S(x_3,x_0,Y_0,x_0) \wedge S(x_3,x'_0,Y'_0,x'_1)) \end{array}$$

Here we report some of the actual Skolem terms generated by skol.

Let us consider the Skolem terms for r_2:

$$\begin{array}{rcl} \mathsf{skol}(r_2,Y_0) & = & f(t1 : S[A : x_3, B : x_0, D : x_1]t2 : S[B : x_0, D : x_0], \\ & & \quad j1 : [t1.C = t2.C], v = j1) \\ \mathsf{skol}(r_2,Y_1) & = & f(t1 : S[A : x_3, B : x_0, D : x_1]t2 : S[B : x_0, D : x_0], \\ & & \quad j1 : [t1.C = t2.C], v = \textit{noj-}t2.A) \end{array}$$

The Skolem terms for r_1 are as follows:

$$\begin{aligned}
\mathsf{skol}(r_1, Y_0) &= f(t1 : [S.A : x_3, S.B : x_0, S.D : x_1]t2 : [S.B : x_0, S.D : x_0] \\
&\qquad t3 : [S.B : x_2], j1 : [t1.C = t2.C], j2 : [t2.A = t3.A], v = j1) \\
\mathsf{skol}(r_1, Y_1) &= f(t1 : [S.A : x_3, S.B : x_0, S.D : x_1]t2 : [S.B : x_0, S.D : x_0] \\
&\qquad t3 : [S.B : x_2], j1 : [t1.C = t2.C], j2 : [t2.A = t3.A], v = j2) \\
\mathsf{skol}(r_1, Y_2) &= f(t1 : [S.A : x_3, S.B : x_0, S.D : x_1]t2 : [S.B : x_0, S.D : x_0] \\
&\qquad t3 : [S.B : x_2], j1 : [t1.C = t2.C], j2 : [t2.A = t3.A], v = \mathit{noj}\text{-}t3.C) \\
\mathsf{skol}(r_1, Y_3) &= f(t1 : [S.A : x_3, S.B : x_0, S.D : x_1]t2 : [S.B : x_0, S.D : x_0] \\
&\qquad t3 : [S.B : x_2], j1 : [t1.C = t2.C], j2 : [t2.A = t3.A], v = \mathit{noj}\text{-}t3.D)
\end{aligned}$$

Notice how, in this case, it is not necessary to resort to the *sort*() function, since the rule conclusions do not contain nontrivial automorphisms, and therefore it is possible to distinguish the tuple terms at script compilation time.

[Step 7] Normalizing and Rewriting Expansion Rules

However, we notice that ϵ_3 is not normalized. As a consequence, we normalize it into two expansion rules ϵ_{3a} and ϵ_{3b}:

$$\begin{aligned}
r_{3a} : \mathsf{sourceRew}\,(\mathsf{mostInf}\,(\epsilon_3)) \rightarrow &\quad S(x_3, x_0, \mathsf{skol}(r_3, Y_0), x_0) \\
r_{3b} : \mathsf{sourceRew}\,(\mathsf{mostInf}\,(\epsilon_3)) \rightarrow &\quad S(x_3, x_0', \mathsf{skol}(r_3, Y_0'), x_1')
\end{aligned}$$

We now look for proper homomorphisms among the four rule conclusions. We notice that r_1 and r_2 subsume r_{3a} with homomorphisms h_1 and h_2, we therefore rewrite r_{3a} as $\mathsf{finalRew}\,(r_{3a})$. A similar rewriting applies for r_{2b}.

$$\begin{aligned}
\mathsf{finalRew}\,(r_1): &\quad \mathsf{sourceRew}\,(\mathsf{mostInf}\,(\epsilon_1)) \rightarrow S(x_3, x_0, \mathsf{skol}(r_1, Y_0), x_1) \wedge \\
&\qquad S(\mathsf{skol}(r_1, Y_1), x_0, \mathsf{skol}(r_1, Y_0), x_0) \wedge \\
&\qquad S(\mathsf{skol}(r_1, Y_1), x_2, \mathsf{skol}(r_1, Y_2), \mathsf{skol}(r_1, Y_3)) \\
\mathsf{finalRew}\,(r_2): &\quad \mathsf{sourceRew}\,(\mathsf{mostInf}\,(\epsilon_2)) \rightarrow S(x_3, x_0, \mathsf{skol}(r_2, Y_0), x_1) \wedge \\
&\qquad S(\mathsf{skol}(r_2, Y_1), x_0, \mathsf{skol}(r_2, Y_0), x_0) \\
\mathsf{finalRew}\,(r_{3a}): &\quad \mathsf{sourceRew}\,(\mathsf{mostInf}\,(\epsilon_{3a})) \wedge \\
&\qquad \neg(\exists \overline{x}' : \mathsf{sourceRew}\,(\mathsf{mostInf}\,(\epsilon_1)) \wedge \textsc{equal}_{h_1}(\overline{x}, \overline{x}')) \wedge \\
&\qquad \neg(\exists \overline{x}'' : \mathsf{sourceRew}\,(\mathsf{mostInf}\,(\epsilon_2)) \wedge \textsc{equal}_{h_2}(\overline{x}, \overline{x}'')) \\
&\qquad\qquad \rightarrow S(x_3, x_0, \mathsf{skol}(r_3, Y_0), x_0) \\
\mathsf{finalRew}\,(r_{3b}): &\quad \mathsf{sourceRew}\,(\mathsf{mostInf}\,(\epsilon_{3b})) \wedge \\
&\qquad \neg(\exists \overline{x}' : \mathsf{sourceRew}\,(\mathsf{mostInf}\,(\epsilon_1)) \wedge \textsc{equal}_{h_1}(\overline{x}, \overline{x}')) \wedge \\
&\qquad \neg(\exists \overline{x}'' : \mathsf{sourceRew}\,(\mathsf{mostInf}\,(\epsilon_2)) \wedge \textsc{equal}_{h_2}(\overline{x}, \overline{x}'')) \\
&\qquad\qquad \rightarrow S(x_3, x_0', \mathsf{skol}(r_3, Y_0'), x_1')
\end{aligned}$$

This set of rules is the core schema mapping for the given scenario.

B Proofs of the Theorems

B.1 Proof of Theorem 1

THEOREM 1 *Given a scenario $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st})$, and a source instance I, suppose J is a universal solution for $\mathcal{M}$ over I. Consider the subset J_0 of J defined as follows:*

$$J_0 = \bigcup \mathsf{reduce}(\mathsf{mostInformative}(\mathsf{mostCompact}(\mathcal{W}^{<I,J>}))) \tag{3}$$

Then, J_0 is the core of J.

Proof: Before getting to the actual proof, let us introduce two preliminary results about witness blocks.

Proposition 5 *Given a solution $J \in \mathsf{USol}_{\mathcal{M}}(I)$, for any tgd m and vector of constants $\overline{a}$, the set of witness blocks $\mathcal{W}_{m,\overline{a}}^{<I,J>}$ is closed under isomorphisms.*

Proof: Consider a witness block $w \in \mathcal{W}_{m,\overline{a}}^{<I,J>}$, and suppose there exists $w' \in \mathcal{W}^{<I,J>}$ such that $w \cong w'$, i.e., there exists an isomorphism $h : w' \to w$. We need to show that $w' \in \mathcal{W}_{m,\overline{a}}^{<I,J>}$. Since $w \in \mathcal{W}_{m,\overline{a}}^{<I,J>}$, we know that $w = \psi(\overline{a}, \overline{b})$, for some vector $\overline{b}$ of values in $dom(J)$. To prove the claim it is sufficient to show that also $w' = \psi(\overline{a}, \overline{b}')$, for some $\overline{b}'$. But we know that $w' = h^{-1}(w) = h^{-1}(\psi(\overline{a}, \overline{b})) = \psi(\overline{a}, h^{-1}(\overline{b}))$. This proves the claim. ◇

Consider instance J_0 defined according to Equation 3. It can be seen that the witness blocks of J_0 fall in two categories: beside "principal" witness blocks, there may be *induced* witness blocks. A witness block w in $\mathcal{W}^{<I,J_0>}$ is said to be *induced* if it is a proper subset of another witness block w' in $\mathcal{W}^{<I,J_0>}$. We shall call *principal* any witness block that is not induced.

Proposition 6 *Consider the set of witness blocks $\mathcal{W}^{<I,J_0>}$. There cannot be two witness blocks $w, w' \in \mathcal{W}^{<I,J_0>}$ such that w is a principal witness block, w' is an induced witness block and there exists an injective homomorphism $h : w \to w'$.*

Proof: We shall prove the claim by contradiction. Suppose there exists $h : w \to w'$. Since w' is induced, there exists w'' in $\mathcal{W}^{<I,J_0>}$ such that $w' \subset w''$ and w'' is a principal witness block. But this means that h is also a homomorphism of w into w''; moreover, h is proper, since it is injective by hypothesis and there are atoms in $w'' - w'$ that do not belong to $h(w)$. However, this is not possible by construction of J_0, since w is a maximal witness block with respect to $<$, and therefore cannot have proper homomorphisms into other witness blocks of J_0. ◇

We are now ready to prove the main claim. We need to prove the following:

Part 1. J_0 is a universal solution for $\mathcal{M}$ over I, i.e., $J_0 \in \mathsf{USol}_{\mathcal{M}}(I)$;

Part 2. J_0 does not contain any smaller endomorphic image that is also a solution.

Part 1. – J_0 is a universal solution for $\mathcal{M}$ over I – To prove that $J_0 \in \mathsf{USol}_{\mathcal{M}}(I)$, we shall first prove that J_0 is a solution, and then that it is universal.

To prove that J_0 is a solution, i.e., $J_0 \in \mathsf{Sol}_{\mathcal{M}}(I)$, it is sufficient to show that, for any tgd $m. \forall \overline{x} : \phi(\overline{x}) \to \exists \overline{y}(\psi(\overline{x}, \overline{y}))$ in Σ_{st}, and any vector of constants $\overline{a}$ such that $I \models \phi(\overline{a})$, the set of witness blocks corresponding to m and $\overline{a}$ in J_0, $\mathcal{W}_{m,\overline{a}}^{<I,J_0>}$, is not empty.

Consider now a tgd $m. \forall \overline{x} : \phi(\overline{x}) \to \exists \overline{y}(\psi(\overline{x}, \overline{y}))$ in Σ_{st}, and a vector of constants $\overline{a}$ such that $I \models \phi(\overline{a})$. We know that the set of witness blocks $\mathcal{W}_{m,\overline{a}}^{<I,J>}$ is not empty, since J is a solution; also, it is a finite set, since J is finite. Consider a maximal element w in $\mathcal{W}_{m,\overline{a}}^{<I,J_0>}$ with respect to the $\prec$ relation. We need to distinguish two cases.

(*a*) There is no other $w' \in \mathcal{W}^{<I,J>}$ such that $w < w'$, i.e., w is also maximal with respect to $<$. In this case, since witness blocks are closed under isomorphisms by Proposition 5, the equivalence class of witness blocks isomorphic to w, $\mathcal{E}_w$, is included in $\mathcal{W}_{m,\overline{a}}^{<I,J>}$. Call $w_{\mathcal{E}_w}$ the representative selected for $\mathcal{E}_w$ by reduce; by construction of J_0, $w_{\mathcal{E}_w}$ belongs to $\mathcal{W}_{m,\overline{a}}^{<I,J_0>}$, which cannot be empty.

(*b*) There exists some $w' \in \mathcal{W}^{<I,J>}$ such that $w < w'$; in this case, consider the set of witness blocks $\mathcal{W} = \{w_i | w_i \in \mathcal{W}^{<I,J>} \text{ and } w < w_i\}$, and a maximal element w^* in $\mathcal{W}$. Consider the equivalence class of witness blocks isomorphic to w^*, $\mathcal{E}_{w^*}$, and call $w_{\mathcal{E}_{w^*}}$ the representative selected for $\mathcal{E}_{w^*}$ by reduce; by construction of J_0, $w_{\mathcal{E}_{w^*}} \subseteq J_0$. We know that the following homomorphisms exist:

$$h : w \to w^* \qquad\qquad h' : w^* \to w_{\mathcal{E}_{w^*}}$$

It can be seen that the set of tuples $h'(h(w))$ is a subset of $w_{\mathcal{E}_{w^*}}$ and therefore is contained in J_0. We now show that $h'(h(w)) \in \mathcal{W}_{m,\overline{a}}^{<I,J_0>}$. In fact, we know that $w = \psi(\overline{a}, \overline{b})$; therefore, $h(w) = \psi(\overline{a}, h(\overline{b}))$, and $h'(h(w)) = \psi(\overline{a}, h'(h(\overline{b})))$. As a consequence, $h'(h(w)) \in \mathcal{W}_{m,\overline{a}}^{<I,J_0>}$, and $\mathcal{W}_{m,\overline{a}}^{<I,J_0>}$ is not empty.

This proves that J_0 is a solution. We now need to prove that J_0 is a universal solution for $\mathcal{M}$ over I, i.e., that for any other solution $J' \in \mathsf{Sol}_{\mathcal{M}}(I)$ there exists a homomorphism $h' : J_0 \to J'$. Since we know that J is universal, we also know that there exists a homomorphism $h : J \to J'$. As a consequence, in order to show that h' of J_0 into J' exists, it is sufficient to show that there exists a homomorphism of $h_0 : J_0 \to J$. But we know that h_0 exists and it is the identity mapping, since J_0 is a subset of J. Therefore, we have:

$$h_0 : J_0 \to J \qquad\qquad h : J \to J'$$

By composing the two homomorphisms, we obtain a homomorphism h' of J_0 into J'. As a consequence, $J_0 \in \mathsf{USol}_{\mathcal{M}}(I)$.

Part 2.: J_0 does not contain any smaller endomorphic image that is also a universal solution – We shall prove the claim by contradiction. Suppose there exists a smaller universal solution than J_0, i.e., a solution $J_\# \subset J_0$. Since $J_\#$ is properly contained in J_0, there is at least one tuple t in $J_0 - J_\#$; by removing t from J_0, we are also removing any witness block $w \in \mathcal{W}^{<I,J_0>}$ that contains t.

Since $J_\#$ is a universal solution, it must be homomorphically equivalent to J_0. Therefore, we know there exists a homomorphism $h_\# : J_0 \to J_\#$. This is obviously true also for any subset of J_0. Let us consider one of the principal witness blocks w that belong to J_0 but not to $J_\#$, i.e., $w \in \mathcal{W}^{<I,J_0>} - \mathcal{W}^{<I,J_\#>}$. Let us call h_w the restriction of $h_\#$ to w, i.e., $h_w : w \to J_\#$. We shall now prove that such a homomorphism cannot exist.

Let us consider the image of w in $J_\#$, $h_w(w)$. Note that $w \neq h_w(w)$, since w is not contained in $J_\#$. On the contrary, since $h_w(w)$ is contained in $J_\#$, it is also contained in J_0. Call $\mathcal{W}^{<I,J_0>}_{m,\overline{a}}$ one of the witness-block sets to which w belongs. It is easy to see that, since w is of the form $\psi(\overline{a},\overline{b})$, $h_w(w)$ is of the form $\psi(\overline{a}, h_w(\overline{b}))$. Therefore, also $h_w(w)$ is a witness block in $\mathcal{W}^{<I,J_0>}_{m,\overline{a}}$. As a consequence, we know that $\mathcal{W}^{<I,J_0>}_{m,\overline{a}}$ contains two distinct witness blocks, w and $h_w(w)$.

We notice that $h_w : w \to h_w(w)$ is a surjective mapping. Let us consider the number of nulls in w and $h_w(w)$. There are three possible cases.

(*a*) $|vars(h_w(w))| = |vars(w)|$ – in this case $h_w(w)$ is simply a renaming of the labeled nulls of w and h_w must be one to one; this means that w and $h_w(w)$ are isomorphic; therefore, $h_w(w)$ cannot be a principal witness block, by construction of J_0; however, it cannot be an induced witness block, either, because of Proposition 6;

(*b*) $|vars(h_w(w))| > |vars(w)|$ – in this case, since h_w is surjective, any labeled null in $h_w(w)$ is the image of some value in w; but since there are more nulls $h_w(w)$ than in w, it must be the case that some of the nulls in $h_w(w)$ are image of a constant in w, and this contradicts the definition of homomorphism; therefore, this cannot be the case;

(*c*) $|vars(h_w(w))| < |vars(w)|$ – in this case, since h_w is surjective, h_w would be a compacting homomorhpism; this is clearly not possible, since by hypothesis w is maximal with respect to $\prec$.

Therefore, we have shown that h_w cannot exist. This proves Part 2. of the claim and concludes the proof. ◇

B.2 Proof of Theorem 2 - Preliminary Notions

Before getting to the actual proof of Theorem 2, we shall introduce several preliminary definitions and lemmas.

Images of a Variable Since formula homomorphisms map variable occurrences to variable occurrences, they are not mappings among variables. In fact, they typically relate occurrences of a variable with occurrences of several different variables. To formalize this notion, we need to introduce the notion of an *image* of a variable according to a formula homomorphism.

Definition 27 ***[Image of a Variable]*** *Given a formula homomorphism $h^f : \varphi(\overline{x},\overline{y}) \to \varphi'(\overline{x}',\overline{y}')$; for every variable v_i in $\varphi(\overline{x},\overline{y})$, the* image *of v_i according to h^f is the set of variables $\mathcal{V}_{h^f}(v_i)$ whose occurrences are images of occurrences of v_i via h^f, defined as follows:*

$$\mathcal{V}_{h^f}(v_i) \quad = \quad \{v'_k \mid \quad R.A : v_i \in occ(\varphi(\overline{x},\overline{y})) \text{ and } h^f(R.A : v_i) = R.A : v'_k\}$$

Let us first establish an important property of variable images.

Proposition 7 *Given an instance J and two formulas $\varphi(\overline{x},\overline{y})$, $\varphi'(\overline{x}',\overline{y}')$ such that there exists a formula homomorphism $h^f : \varphi(\overline{x},\overline{y}) \rightarrow \varphi'(\overline{x}',\overline{y}')$, suppose there are assignments a, a' such that: $J \models a(\varphi(\overline{x},\overline{y}))$, $J \models a'(\varphi'(\overline{x}',\overline{y}'))$ and a, a' are such that $\text{EQUAL}_{h^f}(a(\overline{x}), a'(\overline{x}'))$ evaluates to true. Then, for every variable $v \in \overline{x} \cup \overline{y}$, it is the case that:*

- *a' has the same value on all variables $v' \in \mathcal{V}_{h^f}(v)$;*
- *if v is universal, then it is the case that $a(v) = a'(v')$, for every variable $v' \in \mathcal{V}_{h^f}(v)$.*

Proof: We shall first prove the claim in the case in which the variable is universal, and then existential.

Consider a universal variable $x \in \overline{x}$. Consider $\text{INTERSECT}_{h^f}(\overline{x},\overline{x}')$. We know that it contains an equality of the form $x = x'$ for every variable occurrence in $\varphi'(\overline{x}',\overline{y}')$ such that $R.A : x' = h^f(R.A : x)$. This means that, for any two variables x'_i, x'_j in $\mathcal{V}_{h^f}(x)$, $\text{INTERSECT}_{h^f}(\overline{x},\overline{x}')$ contains equalities of the form $x = x'_i$, $x = x'_j$. Since we know that a, a' are such that $\text{EQUAL}_{h^f}(a(\overline{x}), a'(\overline{x}'))$ evaluates to true, it must be the case that $a(x) = a'(x'_i) = a'(x'_j)$, which proves the claim.

Consider now an existentially quantified variable $y \in \overline{y}$. We need to prove that all variables in $v' \in \mathcal{V}_{h^f}(v)$ receive by a' the same value. In this case, for every pair of occurrences $R_i.A_j : y, R_n.A_m : y$, by definition of a formula homomorphism, we have two possible cases: (i) either both occurrences are mapped to occurrences of the same existential variable y'; in this case, $\mathcal{V}_{h^f}(y)$ is a singleton set containing y', and the claim is obviously true; (ii) or they are mapped to universal occurrences of variables x'_h, x'_k; but in this case, $\text{JOINS}_{h^f}(\overline{x},\overline{x}')$ contains an equality of the form $x'_h = x'_k$, and it must be the case that $a'(x'_h) = a'(x'_k)$; therefore all variables in $\mathcal{V}_{h^f}(y)$ receive the same value by a'. This proves the claim. $\diamond$

Homomorphisms and Formula Homomorphisms We now want to establish two important lemmas, that show the dual nature of homomorphisms among facts and homomorphisms among formulas. More specifically, under appropriate conditions, whenever one exists there exists also the other. In order to do this, we need to introduce a notion of *compatibility* among these two kinds of homomorphisms, as follows.

Definition 28 ***[Compatible Formula Homomorphisms]*** *Given a formula homomorphism between conjunctive formulas, $h^f : \varphi(\overline{x},\overline{y}) \rightarrow \varphi'(\overline{x}',\overline{y}')$, and assignments a, a' such that there is a homomorphism $h : a(\varphi(\overline{x},\overline{y})) \rightarrow a'(\varphi'(\overline{x}',\overline{y}'))$, we say that h^f is* compatible *with h, a, a' if, for every variable $v \in \overline{x} \cup \overline{y}$, and for every variable $v' \in \mathcal{V}_{h^f}(v)$, it is the case that $h(a(v)) = a'(v')$.*

In essence, we are requiring that h maps the value $a(v)$ of a variable v to the value $a'(v')$ of every variable that is in the image of v according to h^f. We also need to introduce the notion of an *invertible assignment*. Recall that an assignment a for $\varphi(\overline{x},\overline{y})$ is *canonical* if it injectively associates a labeled null with each existential variable $y_i \in \overline{y}$. The set of facts $a(\varphi(\overline{x},\overline{y}))$ is called a *canonical block* if a is canonical.

Definition 29 ***[Invertible Assignment]*** *A canonical assignment is called* invertible *if $|a(\varphi(\overline{x},\overline{y}))| = |\varphi(\overline{x},\overline{y})|$, i.e., each atom in $\varphi(\overline{x},\overline{y})$ generates a different fact.*

We are now ready to state our result about the dual nature of fact and formula homomorphisms.

Lemma 8 *Given an instance J, and two conjunctive formulas $\varphi(\overline{x},\overline{y})$ and $\varphi'(\overline{x}', \overline{y}')$ such that there exists a formula homomorphism $h^f : \varphi(\overline{x},\overline{y}) \rightarrow \varphi'(\overline{x}',\overline{y}')$, suppose there exist canonical assignments a, a' such that $J \models a(\varphi(\overline{x},\overline{y}))$, $J \models a'(\varphi'(\overline{x}',\overline{y}'))$, and a, a' are such that $\text{EQUAL}_{h^f}(a(\overline{x}), a'(\overline{x}'))$ evaluates to true. Then, there exists a homomorphism $h : a(\varphi(\overline{x},\overline{y})) \rightarrow a'(\varphi'(\overline{x}',\overline{y}'))$, and h^f is compatible with h, a, a'. Moreover:*

- *if h^f is a surjection, then h is a surjection;*
- *if h^f is proper, and a' is invertible, then h is proper.*

Proof: We shall construct h, and then show that it is a valid homomorphism. For any variable $v \in \overline{x} \cup \overline{y}$, consider the value $a(v)$, and let us define $h(a(v))$ in such a way that $h(a(v)) = a'(v')$, where v' is any variable in $\mathcal{V}_{h^f}(v)$. By Proposition 7, we know that this is a well defined mapping. Note also that, by construction, if h is indeed a homomorphism, h^f is compatible with h, a, a'.

We shall now prove that h is a homomorphism of $a(\varphi(\overline{x}, \overline{y}))$ into $a'(\varphi'(\overline{x}', \overline{y}'))$. To see this, consider any atom $R_i(\ldots A_j : v_k \ldots)$ in $\varphi(\overline{x}, \overline{y})$. Recall that, since h^f is a valid formula homomorphism, we know that $R_i(\ldots A_j : {h^f}_{R_i.A_j}(v_k) \ldots) \in \varphi'(\overline{x}', \overline{y}')$. By construction of h, we also know that, for any occurrence $R_i.A_j : v_k$, we have that $h(a(v_k)) = a'({h^f}_{R_i.A_j}(v_k))$. It follows that:

$$\begin{array}{lcll} h(a(R_i(\ldots A_j : v_k \ldots))) & = & R_i(\ldots A_j : h(a(v_k)) \ldots) & = \\ R_i(\ldots A_j : a'({h^f}_{R_i.A_j}(v_k)) \ldots) & = & a'(R_i(\ldots A_j : {h^f}_{R_i.A_j}(v_k) \ldots)) & \in \quad a'(\varphi'(\overline{x}', \overline{y}')) \end{array}$$

This proves that h is a valid homomorphism, and also that h^f is indeed compatible with h, a, a'. To complete the proof, we need to show that:

- if h^f is a surjection, then h is a surjection;
- if h^f is proper, and a' is invertible, then h is proper.

Suppose h^f is a surjection. Then, every atom $R_i(\overline{x}_i', \overline{y}_i') \in \varphi'(\overline{x}', \overline{y}')$ is the image of some atom $R_i(\overline{x}_i, \overline{y}_i) \in \varphi(\overline{x}, \overline{y})$. By construction of h, it is the case that $a'(R_i(\overline{x}_i', \overline{y}_i'))$ is the image of $a(R_i(\overline{x}_i, \overline{y}_i))$ according to h, and therefore h is a surjection.

Similarly, assume h^f is proper. We want to prove that h is proper. We shall first prove that (a) h is injective, and then that (b) h^f is not surjective.

(a) We shall prove that h is injective by contradiction. Assume h is not injective. This means that there are two distinct facts $R(t_0), R(t_1)$ in $a(\varphi(\overline{x}, \overline{y}))$ such that $h(R(t_0)) = h(R(t_1))$. Call $R^l(\overline{x}, \overline{y}), R'^l(\overline{x}, \overline{y})$ the atoms in $\varphi(\overline{x}, \overline{y})$ such that $R(t_0) = a(R^l(\overline{x}, \overline{y})), R(t_1) = a(R'^l(\overline{x}, \overline{y}))$. We know that $R^l(\overline{x}, \overline{y}), R'^l(\overline{x}, \overline{y})$ are distinct atoms, since $R(t_0) \neq R(t_1)$. We have that:

$$\begin{array}{lcll} h(R(t_0)) = h(a(R^l(\ldots A_j : v_k \ldots))) & = & R(\ldots A_j : h(a(v_k)) \ldots) & = \\ R(\ldots A_j : a'({h^f}_{R^l.A_j}(v_k)) \ldots) & = & a'(h^f(R^l(\ldots A_j : v_k \ldots))) & \\ h(R(t_1)) = h(a(R'^l(\ldots A_j : v_k \ldots))) & = & R(\ldots A_j : h(a(v_k)) \ldots) & = \\ R(\ldots A_j : a'({h^f}_{R'^l.A_j}(v_k)) \ldots) & = & a'(h^f(R'^l(\ldots A_j : v_k) \ldots))) & \end{array}$$

We therefore have that:

$$a'(h^f(R^l(\ldots A_j : v_k \ldots))) = a'(h^f(R'^l(\ldots A_j : v_k) \ldots)))$$

Notice, however, that h^f is injective by hypothesis, and therefore:

$$h^f(R^l(\ldots A_j : v_k \ldots)) \neq h^f(R'^l(\ldots A_j : v_k) \ldots))$$

But this contradicts the hypothesis that a' is invertible, since we are concluding that a' maps two different atoms in $\varphi'(\overline{x}', \overline{y}')$ to the same fact. Therefore, h^f must be injective.

(b) We now want to prove that h is not surjective. Since h^f is proper, there exists an atom $R_i(\overline{x}_i', \overline{y}_i') \in \varphi'(\overline{x}', \overline{y}')$ that is not the image of an atom in $\varphi(\overline{x}, \overline{y})$. Since a' is invertible, the fact $a'(R_i(\overline{x}_i', \overline{y}_i'))$ is such that it can only be generated by $R_i(\overline{x}_i', \overline{y}_i')$. Since there is no atom in $\varphi(\overline{x}, \overline{y})$ that maps into $R_i(\overline{x}_i', \overline{y}_i')$, by construction of h, $a'(R_i(\overline{x}_i', \overline{y}_i'))$ cannot belong to the image of h. This proves that h is proper. $\diamond$

The relationship among fact homomorphisms and formula homomorphisms stated in Lemma 8 has a dual aspect, as stated in Lemma 9. Before stating the lemma, we need to introduce a tool that plays an important role in the proof. More specifically, given a formula and one of its instances, we introduce a way to map each tuple in the formula instance to an atom in the formula, as follows:

Definition 30 ***[Mapping Facts to Atoms]*** *Given a formula $\varphi(\overline{x},\overline{y})$ and a canonical assignment a for it, we introduce a mapping, called $atom_a$, of each fact in $a(\varphi(\overline{x},\overline{y}))$ to an atom in $\varphi(\overline{x},\overline{y})$. More specifically, given a fact $t \in a(\varphi(\overline{x},\overline{y}))$, we define $atom_a(t)$ as one atom $R_i(\overline{x}_i,\overline{y}_i)$ of $\varphi(\overline{x},\overline{y})$ such that $t = a(R_i(\overline{x}_i,\overline{y}_i))$. Notice that, if a is invertible, there is exactly one atom of this kind for each fact. This is not true in general if a is not invertible; in this case, we nondeterministically pick one of the atoms associated with each fact.*

Lemma 9 *Given two conjunctive formulas $\varphi(\overline{x},\overline{y})$, $\varphi'(\overline{x}',\overline{y}')$, suppose there are assignments a, a' such that there exists a homomorphism $h : a(\varphi(\overline{x},\overline{y})) \to a'(\varphi'(\overline{x}',\overline{y}'))$ and a' is a canonical assignment for $\varphi'(\overline{x}',\overline{y}')$. Then, there exists a formula homomorphism: $h^f : \varphi(\overline{x},\overline{y}) \to \varphi'(\overline{x}',\overline{y}')$ and h^f is compatible with h, a, a'. Moreover:*

- *if h is a surjection, and a' is invertible, then h^f is a surjection;*
- *if h is proper, and a, a' are invertible, then h^f is proper.*

Proof: Let us call $w = a(\varphi(\overline{x},\overline{y}))$, $w' = a'(\varphi'(\overline{x}',\overline{y}'))$. We want to build a formula homomorphism h^f of $\varphi(\overline{x},\overline{y})$ into $\varphi'(\overline{x}',\overline{y}')$. As a first step, we introduce a mapping of each atom $R_i(\overline{x}_i,\overline{y}_i) \in \varphi(\overline{x},\overline{y})$ to an atom in $\varphi'(\overline{x}',\overline{y}')$, called $match_{h,a,a'}(R_i(\overline{x}_i,\overline{y}_i))$, according to the following strategy:

- we first map $R_i(\overline{x}_i,\overline{y}_i)$ to $a(R_i(\overline{x}_i,\overline{y}_i)) \in w$;
- then, we map $a(R_i(\overline{x}_i,\overline{y}_i))$ to $h(a(R_i(\overline{x}_i,\overline{y}_i))) \in w'$;
- finally, we map $h(a(R_i(\overline{x}_i,\overline{y}_i)))$ to $atom_{a'}(h(a(R_i(\overline{x}_i,\overline{y}_i)))) \in \varphi'(\overline{x}',\overline{y}')$;

i.e., we have that: $match_{h,a,a'}(R_i(\overline{x}_i,\overline{y}_i)) = atom_{a'}(h(a(R_i(\overline{x}_i,\overline{y}_i))))$.

To build h^f, we consider each variable occurrence $R_i.A_j : v_k$ in $\varphi(\overline{x},\overline{y})$, and choose $h^f(R_i.A_j : v_k)$ to be the corresponding variable occurrence in $match_{h,a,a'}(R_i(\overline{x}_i,\overline{y}_i))$, called $occ_{A_j}(match_{h,a,a'}(R_i(\overline{x}_i,\overline{y}_i)))$, i.e.:

$$h^f(R_i.A_j : v_k) = occ_{A_j}(match_{h,a,a'}(R_i(\overline{x}_i,\overline{y}_i))) \tag{4}$$

To prove that h^f is a formula homomorphism of $\varphi(\overline{x},\overline{y})$ into $\varphi'(\overline{x}',\overline{y}')$, according to the definition, we need to show that:

- h^f maps each atom in $\varphi(\overline{x},\overline{y})$ to an atom in $\varphi'(\overline{x}',\overline{y}')$;
- h^f maps universal occurrences in $\varphi(\overline{x},\overline{y})$ to universal occurrences in $\varphi'(\overline{x}',\overline{y}')$;
- h^f is such that two different occurrences of the same existential variable in $\varphi(\overline{x},\overline{y})$ are either mapped to universal occurrences, or to occurrences of the same existential variable in $\varphi'(\overline{x}',\overline{y}')$.

It can be seen immediately that, by construction, h^f maps each atom $R_i(\overline{x}_i,\overline{y}_i)$ in $\varphi(\overline{x},\overline{y})$ to an atom $match_{h,a,a'}(R_i(\overline{x}_i,\overline{y}_i))$ in $\varphi'(\overline{x}',\overline{y}')$. In fact, by construction h^f maps each occurrence $R_i.A : v_i$ in $R_i(\overline{x}_i,\overline{y}_i)$ to the corresponding occurrence in $match_{h,a,a'}(R_i(\overline{x}_i,\overline{y}_i))$.

We note that h^f maps universal occurrences into universal occurrences. In fact, each universal occurrence $R_i.A_j : v_k$ in $\varphi(\overline{x},\overline{y})$ is first mapped to a constant in $a(R_i(\ldots A_j : v_k \ldots)) \in w$, and then to a constant in $h(a(R_i(\ldots A_j : v_k \ldots))) \in w'$; then, since we know that w' is a canonical block for $\varphi'(\overline{x}',\overline{y}')$, only universal variables are mapped by a' to constants; therefore, $occ_{A_j}(match_{h,a,a'}(R_i(\overline{x}_i,\overline{y}_i)))$ must be a universal occurrence.

Finally, consider two occurrences $R_i.A_j : y_k$, $R_n.A_m : y_k$ of the same existential variable y_k in $\varphi(\overline{x},\overline{y})$. There are two possible cases:

(i) $a(y_k)$ is a constant; in this case, since w' is a canonical block, we know that both $R_i.A_j : y_k$, $R_n.A_m : y_k$ will be mapped to universal variable occurrences in $\varphi'(\overline{x}',\overline{y}')$;

(ii) $a(y_k)$ is a labeled null; in this case, both occurrences in $\varphi(\overline{x},\overline{y})$ will be mapped to the same labeled null $a(y_k)$ in w; this, in turn, can be either mapped to a constant or a labeled null by h; if $h(a(y_k))$ is a constant, then, by the same reasoning as (i) above, we know that both occurrences of y_k will be mapped to universal occurrences; if, on the contrary,

$h(a(y_k))$ is a labeled null, since we know that w' is a canonical block, i.e., a' maps existential variables injectively to labeled nulls, both occurrences will be mapped to occurrences of the same existential variable in $\varphi'(\overline{x}', \overline{y}')$.

This proves that h^f is a valid formula homomorphism.

To show that h^f is compatible with h, a, a', we need to prove that, for every variable $v \in \overline{x} \cup \overline{y}$, and for every variable $v' \in \mathcal{V}_{h^f}(v)$, it is the case that $h(a(v)) = a'(v')$. Call $R_i.A_j : v$ the occurrence of v such that $h^f(R_i.A_j : v) = R_i.A_j : v'$. By construction of h^f, we know that:

$$\begin{array}{lcll} R_i(\ldots A_j : v' \ldots) & = & h^f(R_i(\ldots A_j : v \ldots)) & = \\ match_{h,a,a'}(R_i(\ldots A_j : v \ldots))) & = & atom_{a'}(h(a(R_i(\ldots A_j : v \ldots)))) & = \\ & = & atom_{a'}(R_i(\ldots A_j : h(a(v)) \ldots)) & \end{array}$$

Recall now that, by the definition of $atom_{a'}$, $a'(atom_{a'}(t)) = t$. If we apply a' to both the first and the last atom in the equation above, we therefore have:

$$\begin{array}{lcll} a'(R_i(\ldots A_j : v' \ldots)) & = & R_i(\ldots A_j : a'(v') \ldots)) & = \\ a'(atom_{a'}(R_i(\ldots A_j : h(a(v)) \ldots))) & = & R_i(\ldots A_j : h(a(v)) \ldots) & \end{array}$$

Based on this, we can conclude that $a'(v') = h(a(v))$. This proves that h^f is compatible with h, a, a'. To complete the proof, we need to prove that:

- if h is a surjection, and a' is invertible, then h^f is a surjection;
- if h is proper, and a, a' are invertible, then h^f is proper.

These follows immediately from the definition of $atom_{a'}$. In fact, assume h is a surjection and a' is invertible. In this case, by definition of invertible assignment, $atom_{a'}$ is both injective and surjective. Therefore $match_{h,a,a'}$ is obtained by the composition of three surjective mappings – a, h, and $atom_{a'}$, and therefore it is itself surjective. As a consequence, h^f is surjective.

Similarly, assume h is proper, and a, a' are invertible. We need to prove that (a) h^f is injective, and (b) there is an atom in $\varphi'(\overline{x}', \overline{y}')$ that is not the image of an atom in $\varphi(\overline{x}, \overline{y})$ according to h^f.

To prove part (a), i.e, that h^f is injective, we notice that $match_{h,a,a'}$ is obtained by the composition of three injective mappings – a, h, and $atom_{a'}$, and therefore h^f is injective. To prove part (b), we notice that, since h is proper, $h(a(\varphi(\overline{x}, \overline{y}))$ does not coincide with $a'(\varphi'(\overline{x}', \overline{y}'))$. Call $a'(R_i(\overline{x}'_i, \overline{y}'_i))$ the atom in $a'(\varphi'(\overline{x}', \overline{y}'))$ that is not image of an atom in $a(\varphi(\overline{x}, \overline{y}))$. Since each atom in a formula generates a single fact, it must be the case that $R_i(\overline{x}'_i, \overline{y}'_i)$ does not belong to the image of $\varphi(\overline{x}, \overline{y})$ according to $match_{h,a,a'}$. Therefore, h^f is a proper formula homomorphism.

This proves the claim. ◇

Lemma 9 has a direct impact on the way in which our rewritings are evaluated, as stated by the following Lemma.

Lemma 10 *Given two conjunctive formulas $\varphi(\overline{x}, \overline{y})$, $\varphi'(\overline{x}', \overline{y}')$, suppose there are canonical assignments a, a' such that $J \models a(\varphi(\overline{x}, \overline{y}))$ and $J \models a'(\varphi'(\overline{x}', \overline{y}'))$ and there exists a homomorphism: $h : a(\varphi(\overline{x}, \overline{y})) \to a'(\varphi'(\overline{x}', \overline{y}'))$. Call h^f the formula homomorphism of $\varphi(\overline{x}, \overline{y})$ into $\varphi'(\overline{x}', \overline{y}')$ compatible with h, a, a'. Then, $\textsc{equal}_{h^f}(a(\overline{x}), a'(\overline{x}'))$ evaluates to true.*

Proof: Consider $\textsc{equal}_{h^f}(\overline{x}, \overline{x}')$; it contains equalities of two forms:

$$\begin{array}{lcl} \textsc{intersect}_{h^f}(\overline{x}, \overline{x}') & = & \{x_k = x'_k \mid x_k \in \overline{x},\ R_i.A_j : x'_k = {h^f}_{R_i.A_j}(x_k)\} \\ \textsc{joins}_{h^f}(\overline{x}') & = & \{x'_h = x'_l \mid\ y_k \in \overline{y},\ x'_h = {h^f}_{R_i.A_j}(y_k), x'_l = {h^f}_{R_n.A_m}(y_k)\} \\ \textsc{equal}_{h^f}(\overline{x}, \overline{x}') & = & \textsc{intersect}_{h^f}(\overline{x}, \overline{x}') \cup \textsc{joins}_{h^f}(\overline{x}') \end{array}$$

Let us first consider $\textsc{intersect}_{h^f}(\overline{x}, \overline{x}')$. To prove the claim we need to show that $a(x_k) = a'(x'_k)$, whenever $R_i.A_j : x'_k = {h^f}_{R_i.A_j}(x_k)$. But it is easily seen that, since $x'_k \in \mathcal{V}_{h^f}(x_k)$, and h^f is compatible with h, a, a', by definition of compatible homomorphism, it is the case that $h(a(x_k)) = a'(x'_k)$. Since both x_k and x'_k are universal, $a(x_k)$ is a constant, and therefore $h(a(x_k)) = a(x_k) = a'(x'_k)$. This proves that $\textsc{intersect}_{h^f}(a(\overline{x}), a'(\overline{x}'))$ evaluates to true.

Let us now consider $\text{JOINS}_{h^f}(\overline{x}')$. To prove the claim, we need to show that, $a'(x'_h) = a'(x'_l)$, for every pair of universal variables in the image of some $y_k \in \overline{y}$. But since both x'_h and x'_l belong to $\mathcal{V}_{h^f}(y_k)$, by definition of compatible homomorphism it must be the case that $h(a(y_k)) = a'(x'_h) = a'(x'_l)$. Therefore also $\text{JOINS}_{h^f}(a'(\overline{x}'))$ evaluates to true. This proves the claim. ◇

B.3 Proof of Theorem 2

THEOREM 2 *Given a scenario $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st})$, a source instance I, call J a canonical universal solution of Σ_{st} over I. If J is isomorphism–free, consider the set of expansions* $\mathsf{expansions}(\mathcal{M})$ *and, for each expansion*

$$\epsilon = \chi^l(\overline{x}_1, \overline{y}_1) \wedge \exists \overline{x}_2, \overline{y}_2 : (\psi^l(\overline{x}_2, \overline{y}_2) \bigwedge \text{EQUAL}_{h^f{}_\epsilon}(\overline{x}_1, \overline{x}_2))$$

its rewriting, $\mathsf{mostInf}\,(\epsilon)$*. The following set:*

$$\mathcal{E}^J_{\mathsf{mostInf}} = \bigcup_{\epsilon \in \mathsf{expansions}(\mathcal{M})} \{a(\chi^l(\overline{x}_1, \overline{y}_1)) \mid a \text{ s.t. } J \models a(\mathsf{mostInf}\,(\epsilon))\}$$

is such that:

$$\mathcal{E}^J_{\mathsf{mostInf}} = \mathsf{reduce}(\mathsf{mostInformative}(\mathsf{mostCompact}(\mathcal{W}^{<I,J>})))$$

Proof: In order to prove the claim, we introduce the following additional sets:

$$\begin{array}{lcl} \mathcal{E}^J & = & \bigcup_{\epsilon \in \mathsf{expansions}(\mathcal{M})} \{a(\chi^l(\overline{x}_1, \overline{y}_1)) \mid a \text{ s.t. } J \models a(\epsilon)\} \\ \mathcal{E}^J_{\mathsf{mostComp}} & = & \bigcup_{\epsilon \in \mathsf{expansions}(\mathcal{M})} \{a(\chi^l(\overline{x}_1, \overline{y}_1)) \mid a \text{ s.t. } J \models a(\mathsf{mostComp}\,(\epsilon))\} \end{array}$$

The proof is organized in three parts. We shall prove the following claims:

Part 1. $\mathcal{E}^J = \mathcal{W}^{<I,J>}$

Part 2. $\mathcal{E}^J_{\mathsf{mostComp}} = \mathsf{mostCompact}(\mathcal{W}^{<I,J>})$

Part 3. $\mathcal{E}^J_{\mathsf{mostInf}} = \mathsf{mostInformative}(\mathsf{mostCompact}(\mathcal{W}^{<I,J>}))$

Notice, in fact, that, once we have proven Part 3., the thesis follows immediately from the hypothesis that J is isomorphism-free. This means that any equivalence class $\mathcal{E}_i$ of isomorphic witness blocks in $\mathcal{E}^J_{\mathsf{mostInf}}$ is a singleton. Therefore, $\mathsf{reduce}()$ is the identity mapping on $\mathsf{mostInformative}(\mathsf{mostCompact}(\mathcal{W}^{<I,J>}))$, and therefore the thesis is proven.

Part 1. – $\mathcal{E}^J = \mathcal{W}^{<I,J>}$

We shall first prove that $\mathcal{E}^J_m \subseteq \mathcal{W}^{<I,J>}_m$, and then that $\mathcal{W}^{<I,J>}_m \subseteq \mathcal{E}^J_m$, for each $m \in \Sigma_{st}$.

Part 1.(first half) – $\mathcal{E}^J \subseteq \mathcal{W}^{<I,J>}$

To show that $\mathcal{E}^J \subseteq \mathcal{W}^{<I,J>}$, consider a set of facts $w_\epsilon \in \mathcal{E}^J$. We need to show that w_ϵ is a witness block for some tgd $m. \forall \overline{x} : \phi(\overline{x}) \rightarrow \exists \overline{y}(\psi(\overline{x}, \overline{y}))$, i.e., there exists a vector of constants $\overline{a}_w$ such that $w_\epsilon \in \mathcal{W}^{<I,J>}_{m,\overline{a}_w}$. This amount to prove that there exists an assignment a_w that satisfies the following two conditions:

- $I \models \phi(a_w(\overline{x}))$, i.e., $\overline{a}_w = a_w(\overline{x})$;
- w_ϵ has the form $\psi(a_w(\overline{x}), a_w(\overline{y}))$.

In the following, we construct such an assignment a_w. We know that w_ϵ belongs to some $\mathcal{E}_m^J$, for some tgd $\phi(\overline{x}_2) \rightarrow \exists \overline{y}_2(\psi^l(\overline{x}_2, \overline{y}_2))$ and some expansion

$$\epsilon = \chi^l(\overline{x}_1, \overline{y}_1) \wedge \exists \overline{x}_2, \overline{y}_2 : (\psi^l(\overline{x}_2, \overline{y}_2) \bigwedge \text{EQUAL}_{h^f{}_\epsilon}(\overline{x}_1, \overline{x}_2))$$

Recall that we know that $J \models a_1(\epsilon(\overline{x}_1, \overline{y}_1))$, for some assignment a_1, i.e., $J \models a_1(\chi^l(\overline{x}_1, \overline{y}_1)) = w_\epsilon$; call a_2 the assignment such that $J \models a_2(\psi^l(\overline{x}_2, \overline{y}_2))$. We also know that $\text{EQUAL}_{h^f{}_\epsilon}(a_1(\overline{x}_1), a_2(\overline{x}_2))$ evaluates to true.

Since, by definition of expansion, there exists a surjective formula homomorphism h^f of $\psi(\overline{x}_2, \overline{y}_2)$ into $\chi(\overline{x}_1, \overline{y}_1)$, we know by Lemma 8 that there exists a surjective homomorphism $h : a_2(\psi^l(\overline{x}_2, \overline{y}_2)) \rightarrow a_1(\chi^l(\overline{x}_1, \overline{y}_1))$. Since h is surjective, we have that:

$$w_\epsilon = a_1(\chi^l(\overline{x}_1, \overline{y}_1)) = h(a_2(\psi^l(\overline{x}_2, \overline{y}_2))) = \psi^l(h(a_2(\overline{x}_2)), h(a_2(\overline{y}_2)))$$

If we take $a_w = h \circ a_2$, then w_ϵ has the form $a_w(\psi^l(\overline{x}_2, \overline{y}_2))$.

In order to complete the proof that w_ϵ is a witness block for m, we also need to prove that a_w is such that $I \models \phi(a_w(\overline{x}_2))$. Recall that we know that $J \models a_2(\psi(\overline{x}_2, \overline{y}_2))$. Since $a_2(\psi^l(\overline{x}_2, \overline{y}_2))$ is a witness block for m in J, and J is a canonical universal solution, we know that it must be the case that $I \models \phi(a_2(\overline{x}_2))$. We now show that $a_w(\overline{x}_2) = a_2(\overline{x}_2)$, and therefore $I \models \phi(a_w(\overline{x}_2))$. In fact, for any variable $x_{2i} \in \overline{x}_2$, by definition we have that $a_w(x_{2i}) = h(a_2(x_{2i}))$. But x_{2i} is a universal variable, and therefore $a_2(x_{2i})$ is a constant. As a consequence, h is the identity on it. It follows that $a_w(x_{2i}) = a_2(x_{2i})$.

This proves that w_ϵ is a witness block for tgd m, and concludes the proof of the first half of Part 1.

Part 1. (second half) – $\mathcal{W}^{<I,J>} \subseteq \mathcal{E}^J$

Consider a witness block $w \in \mathcal{W}^{<I,J>}$. Call m. $\phi(\overline{x}_2) \rightarrow \exists \overline{y}_2(\psi^l(\overline{x}_2, \overline{y}_2))$ a tgd such that $w \in \mathcal{W}_m^{<I,J>}$. We need to prove that $w \in \mathcal{E}_m^J$.

Since w is a witness block in $\mathcal{W}_m^{<I,J>}$, we know that there exists an assignment a_w such that $w = \psi(a_w(\overline{x}_2), a_w(\overline{y}_2))$.

We need to prove that w is an instance of some expansion ϵ of m. In order to show that ϵ exists, we now construct $\chi^l(\overline{x}_1, \overline{y}_1)$ as follows: since w is a set of facts in a canonical universal solution $J \in \mathsf{USol}_{\mathcal{M}}(I)$, for each fact t_i in w, we consider its provenance, $\mathsf{provenance}(t_i)$, and we pick exactly one of the labeled atoms in it. We notice that w is a canonical block for $\chi^l(\overline{x}_1, \overline{y}_1)$, i.e., there exists a canonical assignment a_χ such that $w = a_\chi(\chi^l(\overline{x}_1, \overline{y}_1))$. Also, a_χ is an invertible assignment. In fact, by construction, $|\chi(\overline{x}_1, \overline{y}_1)| = |w| = |a_\chi(\chi(\overline{x}_1, \overline{y}_1))|$. We shall now prove that

$$\epsilon = \chi^l(\overline{x}_1, \overline{y}_1) \wedge \exists \overline{x}_2, \overline{y}_2 : (\psi^l(\overline{x}_2, \overline{y}_2) \bigwedge \text{EQUAL}_{h^f{}_\epsilon}(\overline{x}_1, \overline{x}_2))$$

is actually a valid expansion for m. In order to do this, it is necessary to prove that there is a surjection $h^f : \psi(\overline{x}_2, \overline{y}_2) \rightarrow \chi(\overline{x}_1, \overline{y}_1)$.

By Lemma 9, we know that there exists a formula homomorphism h^f of $\psi^l(\overline{x}_2, \overline{y}_2)$ into $\chi(\overline{x}_1, \overline{y}_1)$. In fact, we know that $w = a_w(\psi(\overline{x}_2, \overline{y}_2)) = a_\chi(\chi(\overline{x}_1, \overline{y}_1))$. Therefore, there is a trivial automorphism h_i of $a_w(\psi(\overline{x}_2, \overline{y}_2))$ into $a_\chi(\chi(\overline{x}_1, \overline{y}_1))$; since a_χ is a canonical assignment for $\chi(\overline{x}_1, \overline{y}_1)$, Lemma 9 holds. Moreover, since h_i is surjective, and a_χ is invertible, also h^f is surjective. This proves that h^f is a valid formula homomorphism and a surjection, and that ϵ is a valid expansion.

To complete the proof, we need to show that $J \models a_\chi(\epsilon(\overline{x}_1, \overline{y}_1))$. Since we already know that $J \models a_\chi(\chi^l(\overline{x}_1, \overline{y}_1)) = w$, this amounts to show that there exists an assignment a_b such that $J \models a_b(\psi(\overline{x}_2, \overline{y}_2))$, and $\text{EQUAL}_{h^f{}_\epsilon}(a_\chi(\overline{x}_1), a_b(\overline{x}_2))$ evaluates to true.

Let's take $a_b = a_w$, and show that $\text{EQUAL}_{h^f{}_\epsilon}(a_\chi(\overline{x}_1), a_w(\overline{x}_2))$ evaluates to true. But this immediately follows from Lemma 10. In fact, we have already noted that there is a trivial automorphism h_i of $a_w(\psi(\overline{x}_2, \overline{y}_2))$ into $a_\chi(\chi^l(\overline{x}_1, \overline{y}_1))$. Since h^f is by construction compatible with h_i, a_χ, a_w, then by Lemma 10 it follows that $\text{EQUAL}_{h^f{}_\epsilon}(a_\chi(\overline{x}_1), a_w(\overline{x}_2))$ evaluates to true.

This proves that $J \models a_\chi(\epsilon(\overline{x}_1, \overline{y}_1))$, and concludes the proof of Part 1.

Part 2. – $\mathcal{E}^J_{\mathsf{mostComp}} = \mathsf{mostCompact}(\mathcal{W}^{<I,J>})$

Part 2. (first half) – $\mathcal{E}^J_{\mathsf{mostComp}} \subseteq \mathsf{mostCompact}(\mathcal{W}^{<I,J>})$

Before proving the claim, let us introduce a preliminary lemma.

Lemma 11 *Given a universal solution J and an expansion ϵ, any assignment a such that $J \models a(\mathsf{mostComp}\,(\epsilon))$ is an invertible assignment.*

Proof: We shall prove the claim by contradiction. Suppose, in fact, that a is not invertible. This means that there exist two different atoms $R^l(\ldots), R^{'l}(\ldots)$ in $\chi^l(\overline{x}_1, \overline{y}_1)$ that generate the same fact. Consider now the formula $\chi^{'l}(\overline{x}'_1, \overline{y}'_1)$ obtained from $\chi^l(\overline{x}_1, \overline{y}_1)$ by removing any atom like $R^{'l}(\ldots)$. Call a' the restriction of a to $\overline{x}'_1, \overline{y}'_1$. Notice that $a(\chi^l(\overline{x}_1, \overline{y}_1)) = a'(\chi^{'l}(\overline{x}'_1, \overline{y}'_1)) = w'$, and that a' is invertible by construction.

Based on Lemma 9, we know that, since there is a surjective homomorphism (the identity) from $a(\chi^l(\overline{x}_1, \overline{y}_1))$ to $a'(\chi^{'l}(\overline{x}'_1, \overline{y}'_1))$, and a' is invertible, there must be a surjective formula homomorphism $h^{f'}$ of $\chi^l(\overline{x}_1, \overline{y}_1)$ into $\chi^{'l}(\overline{x}'_1, \overline{y}'_1)$. Therefore, we have that:

$$\epsilon' = \chi^{'l}(\overline{x}'_1, \overline{y}'_1) \wedge \exists \overline{x}_2, \overline{y}_2 : (\psi^l(\overline{x}_2, \overline{y}_2) \bigwedge \text{EQUAL}_{{h^f}_{\epsilon'}}(\overline{x}'_1, \overline{x}_2))$$

is a valid expansion of m. In fact, since ϵ is an expansion, we know there exists a surjection ${h^f}_{\epsilon} : \psi^l(\overline{x}_2, \overline{y}_2) \rightarrow \chi^l(\overline{x}_1, \overline{y}_1)$, and therefore there exists a surjection ${h^f}_{\epsilon'} : \psi^l(\overline{x}_2, \overline{y}_2) \rightarrow \chi^{'l}(\overline{x}'_1, \overline{y}'_1)$, obtained by the composition of the two surjective formula homomorphisms ${h^f}_{\epsilon}$ and ${h^f}_{\epsilon'}$.

Consider now the two expansions ϵ, ϵ'. We know there exists a surjection $h^{f'}$ of $\chi^l(\overline{x}_1, \overline{y}_1)$ into $\chi^{'l}(\overline{x}'_1, \overline{y}'_1)$. But notice that $h^{f'}$ is also compacting, since $|\chi^{'l}(\overline{x}'_1, \overline{y}'_1)| < |\chi^l(\overline{x}_1, \overline{y}_1)|$. Therefore, $\mathsf{mostComp}\,(\epsilon)$ has the following form:

$$\mathsf{mostComp}\,(\epsilon) \quad = \quad \epsilon \wedge \neg \exists \overline{x}'_1, \overline{y}'_1 : (\epsilon' \wedge \text{EQUAL}_{h^{f'}}(\overline{x}_1, \overline{x}'_1)) \wedge \ldots$$

But then, by Lemma 10, it is not possible that $J \models a(\mathsf{mostComp}\,(\epsilon))$, since the negated subformula evaluates to true. This contradicts our hypothesis. Therefore, a must be invertible. ◇

In order to prove the claim, we need to prove that any block of facts $w_\epsilon \in \mathcal{E}^J_{\mathsf{mostComp}}$ belongs also to $\mathsf{mostCompact}(\mathcal{W}^{<I,J>})$. This amounts to show that w_ϵ is maximal with respect to $\prec$, i.e., there is no other witness block w' such that $w' \prec w_\epsilon$. We shall prove this by contradiction.

Call ϵ the expansion such that, for assignment a, $J \models a(\mathsf{mostComp}\,(\epsilon)) = w_\epsilon$. By Lemma 11, we know that a is invertible. Suppose there exists a witness block w' such that $w' \prec w_\epsilon$. By Part 1 of the proof, we know there exists an expansion ϵ' in $\mathsf{expansions}(\mathcal{M})$ and an invertible assignment a' such that $J \models a'(\epsilon'(\overline{x}'_1, \overline{y}'_1)) = w'$.

Since we assume that $w' \prec w_\epsilon$, i.e., there is a compacting homomorphism $h' : w_\epsilon \rightarrow w'$, we know by Lemma 9 that there must be a formula homomorphism $h^{f'}$ of $\chi(\overline{x}_1, \overline{y}_1)$ into $\chi'(\overline{x}'_1, \overline{y}'_1)$, defined as follows:

$$h^{f'}(R_i.A_j : v_k) = occ_{A_j}(atom_{a'}(h'(a(R_i(\ldots A_j : v_k \ldots)))))$$

We also know that $h^{f'}$ is a surjection, since h' is a surjection, and a' is invertible. We want to prove that $h^{f'}$ is compacting, i.e., either $|\chi'(\overline{x}'_1, \overline{y}'_1)| < |\chi(\overline{x}_1, \overline{y}_1)|$ or $|\overline{y}'_1| < |\overline{y}_1|$.

Since $h^{f'}$ is a surjection, we know that $|\chi'(\overline{x}'_1, \overline{y}'_1)| \leq |\chi(\overline{x}_1, \overline{y}_1)|$. If $|\chi'(\overline{x}'_1, \overline{y}'_1)| < |\chi(\overline{x}_1, \overline{y}_1)|$ then $h^{f'}$ is compacting.

Suppose, on the contrary, that $|\chi'(\overline{x}'_1, \overline{y}'_1)| = |\chi(\overline{x}_1, \overline{y}_1)|$. Since we know that h' is compacting, it is the case that it is a surjection and $|vars(w')| < |vars(w_\epsilon)|$. It is possible to see that this may only happen if at least one of the following cases occurs:

(*a*) h' maps some null $N \in vars(w_\epsilon)$ to a constant;

(*b*) h' is not an injective mapping of $vars(w_\epsilon)$ into $vars(w')$, i.e., it maps two different nulls $N_i, N_j \in vars(w_\epsilon)$ to the same null $N_k \in vars(w')$.

Let us consider the two cases separately. In case (*a*), call y the existential variable such that $a(y) = N$. If $h'(a(y))$ is a constant, all occurrences of the form $occ_{A_j}(atom_{a'}(R_k(\ldots A_j : h'(a(y)) \ldots)))$ are universal occurrences, since a' is a

canonical assignment, and therefore a constant can be generated by a' only from a universal variable; this means that all occurrences of y are mapped to universal occurrences, and it is the case that $\mathcal{V}_{h^{f'}}(y)$ does not contain any existential variables. Consider the mapping of variables $\mathcal{I} : \overline{y}_1 \rightarrow \overline{y}'_1$ that associates with each existential variable y_i in $\chi(\overline{x}_1, \overline{y}_1)$ the existential variable $\mathcal{I}(y_i)$ in $\chi'(\overline{x}'_1, \overline{y}'_1)$ that is image of y_i via $h^{f'}$, i.e., such that $\mathcal{I}(y_i) \in \mathcal{V}_{h^{f'}}(y_i)$, if this exists. This is in fact a mapping since, by definition, a formula homomorphism either maps all occurrences of an existential variable to occurrences of the same existential variable, or to universal occurrences. It can be seen that $\mathcal{I}$ is surjective, since $h^{f'}$ is surjective, but it is not total, since we know that $\mathcal{V}_{h^{f'}}(y)$ only contains universal variables. As a consequence, it must be the case that $|\overline{y}_1| > |\overline{y}'_1|$, i.e., $h^{f'}$ is compacting.

Consider now case (b) above. By a similar argument, it can be seen that also in this case $\mathcal{I}$ is surjective but it is not injective; in fact, two different existential variables are mapped to the same image. Also in this case, this can happen only if $|\overline{y}_1| > |\overline{y}'_1|$, i.e., if $h^{f'}$ is compacting.

We have shown that there exists an expansion ϵ' such that there is a compacting homomorphism $h^{f'}$ from $\chi(\overline{x}_1, \overline{y}_1)$ into $\chi'(\overline{x}'_1, \overline{y}'_1)$. This means that $\mathsf{mostComp}\,(\epsilon)$ is of the following form:

$$\mathsf{mostComp}\,(\epsilon) \quad = \quad \epsilon \wedge \neg\exists \overline{x}'_1, \overline{y}'_1 : (\epsilon' \wedge \textsc{equal}_{h^{f'}}(\overline{x}_1, \overline{x}'_1)) \wedge ...$$

but this in turn implies that it is not possible that $w_\epsilon \in \mathcal{E}^J_{\mathsf{mostComp}}$. In fact, it must be the case that $J \not\models a(\mathsf{mostComp}\,(\epsilon))$. To see this, notice that there are assignments a, a' such that $w_\epsilon = a(\chi(\overline{x}_1, \overline{y}_1)) \subseteq J$, $w' = a'(\chi'(\overline{x}'_1, \overline{y}'_1)) \subseteq J$, and a homomorphism $h' : w_\epsilon \rightarrow w'$. Therefore, by Lemma 10, $\textsc{equal}_{h^{f'}}(a(\overline{x}_1), a'(\overline{x}'_1))$ evaluates to true, and the existentially quantified subformula evaluates to true. This means that we have reached a contradiction, since w_ϵ cannot belong to $\mathcal{E}^J_{\mathsf{mostComp}}$, and the claim is proven.

This proves the first half of Part 2.

Part 2. (second half) – $\mathsf{mostCompact}(\mathcal{W}^{<I,J>}) \subseteq \mathcal{E}^J_{\mathsf{mostComp}}$

In order to prove the claim, we need to show that, for any witness block w that belongs to $\mathsf{mostCompact}(\mathcal{W}^{<I,J>})$, it is the case that $w \in \mathcal{E}^J_{\mathsf{mostComp}}$. We know, by Part 1 of the proof, that there exists an expansion ϵ such that, for some invertible assignment a, it is the case that $J \models a(\epsilon(\overline{x}_1, \overline{y}_1)) = w$. We need to prove that $J \models a(\mathsf{mostComp}\,(\epsilon))$.

We shall prove the claim by way of contradiction. More specifically, suppose that $J \not\models a(\mathsf{mostComp}\,(\epsilon))$. Since $J \models a(\epsilon(\overline{x}_1, \overline{y}_1)) = w$, by definition of $\mathsf{mostComp}\,(\epsilon)$, there must be some expansion ϵ' such that there is a compacting homomorphism $h^{f'}$ of $\chi(\overline{x}_1, \overline{y}_1)$ into $\chi'(\overline{x}'_1, \overline{y}'_1)$, and, for some assignment a', $J \models a'(\epsilon'(\overline{x}'_1, \overline{y}'_1))$, and a, a' are such that the formula $\textsc{equal}_{h^{f'}}(a(\overline{x}_1), a'(\overline{x}'_1))$ evaluates to true. In this case, in fact, by construction of $\mathsf{mostComp}\,(\epsilon)$, ϵ' appears in it in a negated subformula which evaluates to true.

Consider now $w' = a'(\chi'(\overline{x}'_1, \overline{y}'_1))$. Based on Lemma 8, we know that there exists a homomorphism $h' : w \rightarrow w'$. We also know that h' is a surjection, since $h^{f'}$ is a surjection. We now want to prove that h' is compacting, i.e., $|vars(w')| < |vars(w)|$. Since we know that $h^{f'}$ is compacting, we also know that either $|\chi'(\overline{x}'_1, \overline{y}'_1)| < |\chi(\overline{x}_1, \overline{y}_1)|$ or $|\overline{y}'_1| < |\overline{y}_1|$.

Let us first consider the case in which $|\chi'(\overline{x}'_1, \overline{y}'_1)| < |\chi(\overline{x}_1, \overline{y}_1)|$. Since we know that a is invertible, it must be the case that

$$|w'| = |a'(\chi'(\overline{x}'_1, \overline{y}'_1))| \leq |\chi'(\overline{x}'_1, \overline{y}'_1)| < |\chi(\overline{x}_1, \overline{y}_1)| = |a(\chi(\overline{x}_1, \overline{y}_1))| = |w|$$

i.e., $|w'| < |w|$. This may only happen if w contains at least two distinct atoms $R(t), R(t')$ such that $h'(R(t)) = h'(R(t'))$. It can be seen that these atoms must contain some labeled nulls. In fact, any ground atom can be mapped by h' only to itself, and therefore must belong to both w and w'. Moreover, since $h'(R(t)) = h'(R(t'))$, it must be the case that:

(a) at least one labeled null in $R(t)$ or in $R(t')$ is mapped by h' to a constant;

(b) some null N' in $R(t')$ is mapped to the same null to which a null N in $R(t)$ is mapped, i.e., $h'(N) = h'(N')$.

In case (a), since there is at least one null that is mapped by h' to a constant, and h' is a surjection, it must be the case that $|vars(w')| < |vars(w)|$, i.e., h' is compacting.

In case (b), we know that the size of the image of $vars(w)$ according to h', $h'(vars(w))$, is smaller than the size of $vars(w)$, since two different nulls in w are mapped to the same null in w'; but we know that h' is a surjection, and therefore it must be the case that $|h'(vars(w))| = |vars(w')|$. Therefore, we have that:

$$|vars(w')| = |h'(vars(w))| < |vars(w)|$$

and also in this case h' is compacting.

Let us now consider the second case, the one in which $h^{f'}$ is such that $|\overline{y}'_1| < |\overline{y}_1|$. Since $h^{f'}$ is a surjection, this may happen in the following cases:

- there exists $y \in \overline{y}_1$ such that its image according to $h^{f'}$, $\mathcal{V}_{h^{f'}}(y)$ contains only universal variables;
- there exist $y_i, y_j \in \overline{y}_1$ such that $\mathcal{V}_{h^{f'}}(y_i) = \mathcal{V}_{h^{f'}}(y_j) = y' \in \overline{y}'_1$, i.e., two different variables are mapped to the same existential variable.

We know that h' is a surjection by construction. But then, it must be the case that $|vars(w')| < |vars(w)|$. Suppose, in fact, that for every other existential variable $y_k \in \overline{y}, y_k \neq y$, y_k is mapped to a different variable $y'_k \in \overline{y}'_1$. Then, since a, a' are canonical, $vars(w)$ contains a distinct element $a(y_k)$, and $vars(w')$ a distinct element $h'(a(y_k))$, for any of such variables. But in turn, in both cases, $vars(w)$ contains a distinct element $a(y)$ for which there is no counterpart in $vars(w')$.

Therefore, h' is compacting, and $w \prec w'$. But this is obviously a contradiction, since $w \in \mathsf{mostCompact}(\mathcal{W}^{<I,J>})$, and therefore w is maximal with respect to $\prec$. Therefore, we have proven the claim.

This concludes the proof of Part 2.

Part 3. – $\mathcal{E}^J_{\mathsf{mostInf}} = \mathsf{mostInformative}(\mathsf{mostCompact}(\mathcal{W}^{<I,J>}))$

Part 3. (first half) – $\mathcal{E}^J_{\mathsf{mostInf}} \subseteq \mathsf{mostInformative}(\mathsf{mostCompact}(\mathcal{W}^{<I,J>}))$

In order to prove the claim, we need to prove that any block of facts $w_\epsilon \in \mathcal{E}^J_{\mathsf{mostInf}}$ belongs also to

$$\mathsf{mostInformative}(\mathsf{mostCompact}(\mathcal{W}^{<I,J>}))$$

This amounts to prove that w_ϵ is maximal with respect to $<$, i.e., there exists no witness block w' such that there is a proper homomorphism $h' : w_\epsilon \to w'$.

The proof is very similar to that of the first half of Part 2. Also in this case we proceed by way of contradiction. We call ϵ the expansion such that $J \models a(\mathsf{mostInf}\,(\epsilon)) = w_\epsilon$, for some assignment a. Notice that, by construction of $\mathsf{mostInf}\,(\epsilon)$, any assignment such that $J \models a(\mathsf{mostInf}\,(\epsilon))$ is also such that $J \models a(\mathsf{mostComp}\,(\epsilon))$; as a consequence, by Lemma 11, we know that a is invertible.

Assume that there exists a witness block w' in $\mathsf{mostCompact}(\mathcal{W}^{<I,J>})$ such that there exists a proper homomorphism $h' : w_\epsilon \to w'$. In this case, we know by Part 2 of the proof that there exists an expansion ϵ' and an invertible assignment a' such that $J \models a'(\mathsf{mostComp}\,(\epsilon'))$. Also, since h' is proper and a, a' are invertible, by Lemma 9 we know that there is a proper formula homomorphism $h^{f'}$ of $\chi(\overline{x}_1, \overline{y}_1)$ into $\chi'(\overline{x}'_1, \overline{y}'_1)$. This means that $\mathsf{mostInf}\,(\epsilon)$ has the following form:

$$\mathsf{mostInf}\,(\epsilon) \quad = \quad \mathsf{mostComp}\,(\epsilon) \wedge \neg\exists \overline{x}'_1, \overline{y}'_1 : (\mathsf{mostComp}\,(\epsilon') \wedge \textsc{equal}_{h^{f'}}(\overline{x}_1, \overline{x}'_1)) \wedge ...$$

It follows that $J \not\models a(\mathsf{mostInf}\,(\epsilon))$. In fact, $\textsc{equal}_{h^{f'}}(a(\overline{x}_1), a'(\overline{x}'_1))$ evaluates to true by Lemma 10. This means that it is not possible that $w_\epsilon \in \mathcal{E}^J_{\mathsf{mostInf}}$, i.e., we have reached a contradiction. This proves the claim.

Part 3. (second half) – $\mathsf{mostInformative}(\mathsf{mostCompact}(\mathcal{W}^{<I,J>})) \subseteq \mathcal{E}^J_{\mathsf{mostInf}}$

In order to prove the claim, we need to show that, for any witness block w that belongs to

$$\mathsf{mostInformative}(\mathsf{mostCompact}(\mathcal{W}^{<I,J>}))$$

it is the case that $w \in \mathcal{E}^J_{\mathsf{mostInf}}$. The proof is very similar to that of the second half of Part 2. We know from Part 1 of the proof that there exists an expansion ϵ and invertible assignment a such that $J \models a(\epsilon(\overline{x}_1, \overline{y}_1)) = w$. We need to prove that $J \models a(\mathsf{mostInf}\,(\epsilon))$.

Again, this is done by way of contradiction. Assume $J \not\models a(\mathsf{mostInf}\,(\epsilon))$; there must be some expansion ϵ' such that there is a proper homomorphism $h^{f'}$ of $\chi(\overline{x}_1, \overline{y}_1)$ into $\chi'(\overline{x}'_1, \overline{y}'_1)$, for some assignment a', $J \models a'(\mathsf{mostComp}\,(\epsilon'))$, and a, a' are such that $\textsc{equal}_{h^{f'}}(a(\overline{x}_1), a'(\overline{x}'_1))$ evaluates to true.

Consider now $w' = a'(\chi'(\overline{x}'_1, \overline{y}'_1))$. Based on Lemma 8, we know there must be a homomorphism $h' : w \to w'$. We want to show that h' is proper. By Lemma 11, we know that a' is invertible. Since a' is invertible, by Lemma 8, we know that h is proper, and therefore $w < w'$. But this is not possible, since w is maximal with respect to $<$.

This proves the claim and concludes the proof. ◇

B.4 Proof of Theorem 3

Theorem 3 *Given a $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st})$, and an isomorphism-invariant skolemization strategy, skol, $\Sigma^{core}_{\mathcal{M},\mathsf{skol}}$ is a core schema mapping for $\mathcal{M}$.*

Proof: Recall that:

$$\Sigma^{core}_{\mathcal{M},\mathsf{skol}} = \mathsf{finalRew}\,(\mathsf{normalize}(\Sigma^{exp}_{\mathcal{M},\mathsf{skol}}))$$

In order to prove the claim, we need to show that, given a source instance I, the result of the chase of $\Sigma^{core}_{\mathcal{M},\mathsf{skol}}$ over I is the core universal solution for $\mathcal{M}$ over I, J_0, i.e.:

$$J_{chase} = \Sigma^{core}_{\mathcal{M},\mathsf{skol}}(I) \cong J_0$$

We shall make use of the strong connection between the two possible strategies suggested in the paper to generate the core: the two-step one, and the single-step one that uses source rewritings. More specifically, recall that, as an alternative to chasing $\Sigma^{core}_{\mathcal{M},\mathsf{skol}}$, we might generate the core following a two step process. Based on Theorem 2, we could first generate an isomorphism-free solution, J, by standard chasing Σ_{st} on I, and then could chase the following set of full rules, $\Sigma^{full}_{\mathcal{M}}$, one for each expansion $\epsilon \in \mathsf{expansions}(\mathcal{M})$:

$$\Sigma^{full}_{\mathcal{M}} = \{r^{full}_{\epsilon}.\, \forall \overline{x}_1, \overline{y}_1 : \mathsf{mostInf}\,(\epsilon) \to \chi(\overline{x}_1, \overline{y}_1) \mid \epsilon \in \mathsf{expansions}(\mathcal{M})\}$$

Notice how this double-exchange approach uses a composition of s-t tgds plus full FO-rules. Our strategy in the proof is to show that chasing the skolemized FO-rules in $\Sigma^{core}_{\mathcal{M},\mathsf{skol}}$ generates the same result using a single exchange. However, there is a significant difference in structure among these two sets of rules: the rules in $\Sigma^{core}_{\mathcal{M},\mathsf{skol}}$ are the product of a normalization step and of a further rewriting step, according to $\mathsf{finalRew}\,()$.

We shall therefore apply the same transformations also to $\Sigma^{full}_{\mathcal{M}}$; in doing this, despite the fact that these rules only contain universally quantified variables, in each rule we shall treat the variables in $\overline{y}_1$ as existentially quantified. This will generate the following set of full FO-rules:

$$\Sigma^{norm}_{\mathcal{M}} = \mathsf{finalRew}\,(\mathsf{normalize}(\Sigma^{full}_{\mathcal{M}}))$$

As a first intermediate result, we now want to prove that this normalization and this final rewriting do not have impact on the generation of the core.

Lemma 12 *Given an isomorphism-free solution J, the result of chasing the two sets of rules $\Sigma^{full}_{\mathcal{M}}$ and $\Sigma^{norm}_{\mathcal{M}}$ over J is the same.*

Proof: Let us call J_0 the result of chasing $\Sigma_{\mathcal{M}}^{full}$, and J_* the result of chasing $\Sigma_{\mathcal{M}}^{norm}$ over J.

Being $\Sigma_{\mathcal{M}}^{full}$ a set of full dependencies, the normalization procedure generates a set of logically equivalent new dependencies. Since $\mathsf{finalRew}\,()$ only adds negated atoms to the premises of these equivalent dependencies, we know that $J_* \subseteq J_0$. We now want to prove that it is also the case that $J_0 \subseteq J_*$.

Consider a witness block w in J_0. Assume w is generated by a rule of the form:

$$r_{\epsilon}^{full}.\ \forall \overline{x}_1, \overline{y}_1 : \mathsf{mostInf}\,(\epsilon) \rightarrow \chi(\overline{x}_1, \overline{y}_1)$$

and assignment a.

Consider $a(\mathsf{mostInf}\,(\epsilon))$. If $\chi(\overline{x}_1, \overline{y}_1)$ is normalized, w also belongs to J_*. Assume $\chi(\overline{x}_1, \overline{y}_1)$ is not normalized. Then, let us consider each of its connected components. Each component $\varphi_i(\overline{x}_i, \overline{y}_i)$ generates a rule of the form

$$r_{\epsilon,i}^{full}.\ \mathsf{mostInf}\,(\epsilon) \rightarrow \varphi_i(\overline{x}_i, \overline{y}_i)$$

We want to prove that all sets of facts corresponding to instances of the connected components, $\bigcup_i \{a(\varphi_i(\overline{x}_i, \overline{y}_i))\}$ belong to J_*. There are two possible cases:

(*a*) $J \models a(\mathsf{finalRew}\,(r_{\epsilon,i}^{full})(\overline{x}_1, \overline{y}_1))$, and therefore $a(\varphi_i(\overline{x}_i, \overline{y}_i))$ belongs to J_* as well;

(*b*) $J \not\models a(\mathsf{finalRew}\,(r_{\epsilon,i}^{full})(\overline{x}_1, \overline{y}_1))$; this means that there must be a different rule $r_{\epsilon',j}^{full}$ with a conclusion $\varphi'(\overline{x}', \overline{y}')$ such that there is a proper formula homomorphism h^f of $\varphi(\overline{x}_i, \overline{y}_i)$ into $\varphi'(\overline{x}', \overline{y}')$, and some assignment a' such that: (*i*) $J \models a'(\mathsf{mostInf}\,(\epsilon'))$, and (*ii*) the formula $\textsc{equal}_{h^f}(a(\overline{x}_i), a'(\overline{x}'))$ evaluates to true. Suppose, without loss of generality, that $r_{\epsilon',j}^{full}$ is maximal with respect to proper homomorphisms, i.e., its conclusion does not have homomorphisms into other rule conclusions. Notice that, since $J \models a'(\mathsf{mostInf}\,(\epsilon', j))$, it must be the case that $a'(\varphi'(\overline{x}', \overline{y}'))$ also belongs to J_0; moreover, since we assume that $r_{\epsilon',j}^{full}$ is maximal with respect to $<$, it also belongs to J_*.

In this case, by Lemma 8, we know there exists a homomorphism h of $a(\varphi(\overline{x}_i, \overline{y}_i))$ into $a'(\varphi'(\overline{x}', \overline{y}'))$, and that both belong to J_0. Let us consider the image of $a(\varphi(\overline{x}_i, \overline{y}_i))$ according to h: $h(a(\varphi(\overline{x}_i, \overline{y}_i)))$. It is possible to see that $a(\varphi(\overline{x}_i, \overline{y}_i)) = h(a(\varphi(\overline{x}_i, \overline{y}_i)))$, i.e., h must be the identity mapping.

In fact, assume $a(\varphi(\overline{x}_i, \overline{y}_i)) \neq h(a(\varphi(\overline{x}_i, \overline{y}_i)))$. In this case, consider the original witness block $w \in J$, of which $a(\varphi(\overline{x}_i, \overline{y}_i))$ is a connected component. By taking the other connected components, and adding to them $h(a'(\varphi'(\overline{x}', \overline{y}')))$ it would be possible to construct a new witness block w' that also belongs to J_0, such that $w \not\subseteq w'$, i.e., w is not an induced witness block of w', and there exists a homomorphism of w into w'. Notice that this homomorphism is either proper, or compacting, or an isomorphism. But this obviously contradicts the hypothesis, since by definition of J_0 w is not an induced block, it cannot have isomorphic witness blocks, and is maximal with respect to $\prec$ and $<$. Since h is the identity mapping, then $a(\varphi_i(\overline{x}_i, \overline{y}_i))$ belongs to J_*.

This proves that J_* contains all connected components of w, and therefore w itself. ◇

We are now ready to correlate the results of the two sets of rules, $\Sigma_{\mathcal{M},\mathsf{skol}}^{core}$, and $\Sigma_{\mathcal{M}}^{norm}$. We call $\mathsf{premise}\,(r)$ the premise of rule r. For each expansion ϵ, we know that, for each rule $r_{\epsilon,i}^{c} \in \Sigma_{\mathcal{M},\mathsf{skol}}^{core}$ of the form:

$$r_{\epsilon,i}^{c}.\ \mathsf{premise}\,(r_{\epsilon,i}^{c}) \rightarrow \varphi_{\mathsf{skol}}(\overline{x}_1)$$

there is a corresponding rule $r_{\epsilon,i}^{n} \in \Sigma_{\mathcal{M}}^{norm}$ of the form:

$$r_{\epsilon,i}^{n}.\ \mathsf{premise}\,(r_{\epsilon,i}^{n}) \rightarrow \varphi(\overline{x}_1, \overline{y}_1)$$

We concentrate on the two queries:

$$Q_{\mathsf{premise}\,(r_{\epsilon,i}^{c})}(I) \quad \textit{and} \quad Q_{\mathsf{premise}\,(r_{\epsilon,i}^{n})}(J)$$

We now want to prove the following Lemma.

Lemma 13 *Consider an expansion $\epsilon \in \mathsf{expansions}(\mathcal{M})$, a source instance I and a canonical universal solution $J \in \mathsf{USol}_{\mathcal{M}}(I)$. There is a block of facts*

$$\mathsf{premise}\,(r^n_{\epsilon,i})(a(\overline{x}), b(\overline{y})) \in Q_{\mathsf{premise}\,(r^n_{\epsilon,i})}(J)$$

if and only if there is a block of facts

$$\mathsf{premise}\,(r^c_{\epsilon,i})(a(\overline{x})) \in Q_{\mathsf{premise}\,(r^c_{\epsilon,i})}(I)$$

Proof: Consider expansion ϵ. It is possible to see that, by construction, a block of facts of the form $\epsilon(a(\overline{x}_1), b(\overline{y}_1))$ may exist in J if and only if a block of facts generated by $\mathsf{sourceRew}\,(\epsilon)$ – i.e., a block of the form $\mathsf{sourceRew}\,(\epsilon)(a(\overline{x}_1))$ – exists in I. In fact, recall that

$$\epsilon = \chi^l(\overline{x}_1, \overline{y}_1) \wedge \exists \overline{x}_2, \overline{y}_2 : (\psi^l(\overline{x}_2, \overline{y}_2) \bigwedge \mathrm{EQUAL}_{hf_\epsilon}(\overline{x}_1, \overline{x}_2))$$
$$\mathsf{sourceRew}\,(\epsilon) = \mathsf{premise}\,(\chi^l(\overline{x}_1, \overline{y}_1)) \wedge \exists \overline{x}_2 : (\phi(\overline{x}_2) \bigwedge \mathrm{EQUAL}_{hf_\epsilon}(\overline{x}_1, \overline{x}_2))$$

Let us consider the three parts of each formula. Recall that any assignment c such that $J \models \chi^l(c(\overline{x}_1), c(\overline{y}_1))$ must be a canonical assignment. For $\chi^l(a(\overline{x}_1), b(\overline{y}_1))$ to be contained in J, each fact $R^l(a(\overline{x}_i), b(\overline{y}_i))$ in it must be contained in J. But, by definition of canonical universal solution, this may happen only if each premise of the corresponding tgds is satisfied by a, i.e., if $\mathsf{premise}\,(\chi^l(\overline{x}_1, \overline{y}_1))(a(\overline{x}_1))$ is contained in I.

Consider now the existentially quantified subformula. Obviously there exist assignments a_2, b_2 such that $J \models \psi^l(a_2(\overline{x}_2), b_2(\overline{y}_2))$ if and only if $I \models \phi(a_2(\overline{x}_2))$. Note also that the two sets of equalities are exactly the same. Therefore we may conclude that a block of facts of the form $\epsilon(a(\overline{x}_1), b(\overline{y}_1))$ may exist in J if and only if a block of facts of the form $\mathsf{sourceRew}\,(\epsilon(a(\overline{x}_1)))$ exists in I.

A very similar argument holds for $\mathsf{mostComp}\,(\epsilon)$ and $\mathsf{sourceRew}\,(\mathsf{mostComp}\,(\epsilon))$.

Similarly for $\mathsf{mostInf}\,(\epsilon)$ and $\mathsf{sourceRew}\,(\mathsf{mostInf}\,(\epsilon))$. Since the two sets of rules are normalized in the same way, the claim also holds for the final rewritings generated by $\mathsf{finalRew}\,()$. ◇

Based on Lemma 13, we have established a very close connection between expansions and their source rewriting. More specifically, given an isomorphism-free solution J, consider the two sets:

$$J_{chase} = \Sigma^{core}_{\mathcal{M},\mathsf{skol}}(I) \qquad J_0 = \Sigma^{norm}_{\mathcal{M}}(J)$$

We can show that the two instances are equal up to isomorphisms. In fact, consider a rule $r^c_{\epsilon,i} \in \Sigma^{core}_{\mathcal{M},\mathsf{skol}}$, and the corresponding rule $r^n_{\epsilon,i} \in \Sigma^{norm}_{\mathcal{M}}$, as defined above.

According to Lemma 13, the premise of $r^c_{\epsilon,i}$ is satisfied by I for an assignment a if and only if the premise of $r^n_{\epsilon,i}$ is also satisfied by J for assignment a on $\overline{x}_1$. Let us consider the facts generated by firing the two rules. We know that $\varphi(a(\overline{x}_1), b(\overline{y}_1))$ is a fact block in J_0, while $\varphi_{\mathsf{skol}}(a(\overline{x}_1))$ is a block of facts generated by properly assigning values to Skolem terms.

However, the two blocks are isomorphic. In fact, we know that $\varphi(a(\overline{x}_1), b(\overline{y}_1))$ is a canonical block, and therefore it has been generated by the standard skolemization strategy over $\varphi(a(\overline{x}_1), b(\overline{y}_1))$. But, by hypothesis, the chosen skolemization strategy, skol, is isomorphism-invariant, and therefore it produces blocks of facts isomorphic to those produced by the standard skolemization strategy.

Moreover, we know that, since J is isomorphism-free, J_0 contains exactly one isomorphic copy of each witness block. But this is true also for J_{chase}, since skol is isomorphism-invariant, and therefore by definition isomorphic instances of rule conclusions collapse into a single representative.

Since we have proven that $J_{chase} \cong J_0$, this concludes the proof. ◇

B.5 Proof of Theorem 4

THEOREM 4 *Given a $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st})$ such that Σ_{st} does not contain self-joins in tgd conclusions. Given a source instance I, call J a canonical universal solution for $\mathcal{M}$ over I, and J_0 the core universal solution for $\mathcal{M}$ over I. Then:*
(1) *for any fact block b_f in J, either all tuples in b_f belong also to J_0, or none of them does;*
(2) *for each tgd $m \in \Sigma_{st}$ whose conclusion has size k, all witness blocks in $\mathcal{W}_m^{<I,J>}$ have size exactly k.*

Proof: Let us first prove item 1 of the claim. Assume there is a fact block $b_f \subseteq J$ such that $b_f \not\subseteq J_0$ but at least one tuple in b_f belongs to J_0. Let us first note that, since b_f must contain more than one tuple, it therefore contains at least one null value. Call b_{f0} the proper subset of b_f that belongs to J_0. Call N any null value in $b_f - b_{f0}$, and $R_i(t_N), R_j(t_{N0})$ two tuples that contain N, such that $R_j(t_{N0}) \in b_{f0}$, $R_i(t_N) \in b_f - b_{f0}$. Notice that, since $\mathcal{M}$ does not contain self-joins in tgd conclusions, it must be the case that $R_i \neq R_j$.

Since J_0 is the core of J, there must be an endomorphism $h : J \to J_0$. Consider the image of b_f according to h, $h(b_f)$. It is possible to see that $R_i(t_N) \notin h(b_f)$. In fact, since $R_i(t_N) \in b_f$ but $R_i(t_N) \notin b_{f0}$, it must be the case that $h(R_i(t_N)) \neq R_i(t_N)$. This, in turn, means that $h(N) \neq N$. In fact, $R_i(t_N)$ cannot be mapped to any other tuple $R_j(t_{N0}) \in b_{f0}$ that contains N, since we know that $R_i \neq R_j$. Therefore, $R_i(t_N)$ must be mapped to a tuple that not contains N, and $h(N) \neq N$.

Consider now a tuple $R_j(t_{N0}) \in b_{f0}$. Since $h(N) \neq N$, it must be the case that $h(R_j(t_{N0})) \neq R_j(t_{N0})$. But this means that J_0 contains two different tuples, $R_j(t_{N0})$ and $h(R_j(t_{N0}))$, and therefore it has an endomorphism into its proper subset $J_0 - \{R_j(t_{N0})\}$. As a consequence, J_0 is not the core universal solution. This contradicts the assumption and proves the claim.

Let us now prove item 2. Assume there is a tgd m with conclusion $\psi(\overline{x}, \overline{y})$ of size k such that there exists a witness block $w \in \mathcal{W}_m^{<I,J>}$ of size $k' \neq k$. By definition of witness block, we know that it must be $k' \leq k$. Therefore, it must be the case that $k' < k$. But this is clearly impossible, since any witness block for m must be an instance of $\psi(\overline{x}, \overline{y})$ according to some assignment c. Since $\psi(\overline{x}, \overline{y})$ does not contain self-joins, for any assignment c, $c(\psi(\overline{x}, \overline{y}))$ is a collection of facts each belonging to a different relation and therefore has size exactly k, which contradicts the assumption. This concludes the proof. ◇