Roma Tre University
Ph.D. in Computer Science and Engineering

Quality of Mappings for Data Exchange Applications

Salvatore Raunich

Quality of Mappings for Data Exchange Applications

A thesis presented by
Salvatore Raunich
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy
in Computer Science and Engineering

Roma Tre University
Dept. of Informatics and Automation

March 2009

Committee:
Prof. Giansalvatore Mecca
Prof. Paolo Atzeni

Reviewers:
Prof. Renee J. Miller
Prof. Val Tannen

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.

However, these two research lines have progressed in a rather independent way and we are still far away from having a complete understanding of the properties that a "good" schema mapping system should have; to give an example, there are many possible solutions for a data exchange problem.

In fact, there is no consensus yet on a notion of quality for schema mappings. In this thesis, based on concepts provided by schema mapping and data exchange research, we aim at investigate such a notion.

Our goal is to identify a fairly general formal context that incorporates the different mapping-generation systems proposed in the literature, and to develop algorithms, tools and methods for characterizing the quality of mappings generated by those systems.

Acknowledgments

Several people contributed to this thesis, in different ways. First of all, I would like to thank my family that supported me in this years. In particular, my parents, who gave me the opportunity to study and achieve my goals; my brother, who introduced me to the "Computer Science" world. I know I can always rely on him.

I would like to express my deep gratitude to Prof. Gianni Mecca, because he believed in me for many years and he taught me everything about how to do research. During last years, he has been for me a positive example to follow.

I am grateful to Prof. Paolo Atzeni who gave me the opportunity to follow the PhD Course at Roma Tre University and to conduct this doctoral research.

Thanks also to Paolo Papotti who contributed to the results of this thesis. I will remember long nights at work together.

I am also grateful to Prof. Erhard Rahm and his database group, who gave me the opportunity to visit the Department of Computer Science at the University of Leipzig. A special thank to David, Anika, Michael and Andreas who welcomed me not only as a colleague, but also as a friend.

Even if indirectly, other people contributed to this thesis. Many thanks to my friends Alberto, always present and helpful in each situation, with whom I shared most of the highlights of my life, and Michele, whose friendship comes with me from childhood.

However, my greatest gratitude is for Lucia who chose to share her life with me. This work would be meaningless without her. Thanks for everything.

Contents

List of Tables

List of Figures

Chapter 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 [63, 70]. The original algorithm has been subsequently extended in several ways [46, 22, 9, 72, 27] and various tools have been proposed to support users in the mapping generation process. More recently, a benchmark has been developed [8] 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 practical results obtained on schema mapping systems. This consideration has motivated a rich body of research in which the notion of a *data exchange problem* [42] 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*) [16] – 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 or invert them.

However, these two research lines have progressed in a rather independent way and we are still far away from having a complete understanding of the properties that a "good" schema mapping system should have; to give an example, there are many possible solutions for a data exchange problem.

In fact, there is no consensus yet on a notion of quality for schema mappings. In this thesis, based on concepts provided by schema mapping and data exchange research, we aim at investigate such a notion. Our goal is to identify a fairly general formal context that incorporates the different mapping-generation systems proposed in the literature, and to develop algorithms, tools and methods for characterizing the quality of mappings generated by those systems.

More specifically, we identify three crucial viewpoints under which we believe quality should be studied:

(*i*) which solutions a mapping system should materialize and how it should generate them;

(*ii*) how to generate transformation rules (for example *tgds*), starting from a minimal and high-level specification of the mapping;

(*iii*) finally, how to semi-automatically generate this high-level specification of the mapping.

Choice of Solutions

About the first point, a key contribution of data exchange research was the formalization of the notion of *core* [43] 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 represents a "good" instance for answering 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. Unfortunately, there is yet no schema mapping generation algorithm that natively produces executable scripts that compute the core. On the contrary, the solution produced by known schema mapping systems – called a *canonical universal solution* –

typically contains 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 to generate a canonical solution by chasing the tgds, and then to apply a post-processing algorithm for core identification. Several polynomial algorithms have been identified to this end [43, 51]. These algorithms provide a very general solution to the problem of computing core solutions for a data exchange setting. Also, an implementation of the core-computation algorithm in [51] has been developed [74], thus making a significant step towards the goal of integrating core computations in schema mapping systems. However, experience with these algorithms shows that, although polynomial, they require very high computing times since they look for all possible endomorphisms among tuples in the canonical solution. As a consequence, they hardly scale to large mapping scenarios: even for databases of a few thousand tuples computing the core may require several hours.

Our goal is to introduce a core computation algorithm that lends itself to a more efficient implementation as an executable script and that scales well to large databases.

Transformation Rule Generation

With respect to the second point, manually writing transformation rules under the form of logical formulas can be very difficult also for expert users. It would be much simpler and more natural to provide a minimal and high-level specification of the mapping and let the system generate the rules from that. For these reasons, a typical schema mapping system takes as input an abstract specification of the mapping under the form of *value correspondences* among schema elements and generates the tgds and the executable transformations to run them.

Each value correspondence states that an attribute of the target is semantically related to one (or more) attribute in the source, and it is usually drawn as a *line* from the source attribute to the corresponding target attribute.

Clio [70] was the first system to introduce value correspondences and to implement a sophisticated mapping algorithm to generate source-to-target transformations. The mapping generation algorithm captures all semantical relationships embedded in the source and the target schemas, and is guaranteed to produce legal instances of the target with respect to constraints.

Our goal is to extend the expressive power of the mapping system with respect to Clio system.

Discovery of Correspondences

Finally, it is apparent how a crucial step in the mapping generation process is the discovery of the initial value correspondences. In fact, the quality of the mappings produced by the mapping generation system is strongly influenced by the quality of the input lines: starting from faulty correspondences incorrect mappings are inevitably produced, thus impairing the quality of the overall integration process. To avoid these problems, it is usually assumed that value correspondences are interactively provided to the system by a human expert after carefully browsing the source and target repositories. However, such manual process is very labor-intensive, and does not scale well to medium and large integration tasks.

To alleviate the burden of manually specifying lines, one alternative is to couple the mapping generation system with a *schema matching system*, i.e., a system that automatically or semi-automatically tries to discover matching attributes in a source and target schema. The study of automatic techniques for schema matching has received quite a lot of attention in recent years; for a survey see [73, 38, 76]. Clio itself has been complemented with a companion schema matching module based on attribute feature analysis [67]; this tool may be asked to suggest attribute correspondences to the user.

Unfortunately, schema matching has been recognized as a very challenging problem [49], for which no definitive solution exists: although current schema matching systems perform well in some application categories, in other cases they suffer from poor precision. According to [50], there is no perfect schema matching tool. [50] reports that on a recent benchmark of ontology–matching tasks [4], participating matchers on average achieved 40% precision and 45% recall. Also, even for datasets for which such tools reached higher precision and recall, they still produced inconsistent or erroneous mappings.

As a consequence, outputs of the attribute matching phase are hardly ready to be fed to the mapping generation module, and human intervention is necessary in order to analyze and validate them. It is worth noting that, in general, human intervention is also necessary after mappings have been generated, since several alternative ways of mapping the source into the target may exist. Mapping systems usually produce all alternatives, and offer the user the possibility of inspecting the result of each of them in order to select the preferred one.

In the following, we present an original architecture to integrate schema matching and mapping generation and we introduce an algorithm that combines schema matching, mapping generation and *mapping verification* in order to achieve good scalability and high matching quality.

The thesis is organized as follows:

- in Chapter 2 we introduce the reference context and we recall some definitions related to data exchange;
- in Chapter 3 we formally present a core computation algorithm, based on the rewriting of the original mappings, that lends itself to an efficient implementation as an executable script and that scales well to large databases;
- in Chapter 4 we present a new tgd generation algorithm as a generalization of the *basic mapping* generation algorithm introduced in [70];
- Chapter 5 is devoted to describe an original architecture to integrate schema matching and mapping generation tool and to define the notion of *mapping quality* and the automation of *mapping verification*.
- some related works in the fields of schema mappings and data exchange will be discussed in Chapter 6;

The architecture and the algorithms described in this thesis were developed in the +Spicy[1] system, a research effort at Università della Basilicata. The prototype represents the first mapping system that integrates all quality aspects presented above in a single tool and that contributes to bridge the gap between the practice of mapping generation and the theory of data exchange.

[1] http://db.unibas.it/projects/spicy/

Chapter 2

Data Exchange

In this section we recall some definitions related to data exchange and we describe data models used in our algorithms. In the presentation we follow the main definitions given in [42] and [43].

As it is common, data sources are represented in the system according to an abstract model, with the advantage of reducing different concrete data models (for example, relational, XML, OWL, etc.) to an uniform representation. As in [70], we use a simple *nested relational data model* that can be formalized as follows.

Nested Relational 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 special values, called oids. A type is either a base type (e.g., string, integer, date, etc.) or a set or tuple complex type. A *set type* has the form $\mathsf{set}(A : \tau)$, where A is a label and τ is a type. A *set node* in a schema is a labeled set type, of the form $A : set(\tau)$, with a child node $A : \tau$. A *tuple type* has the form $\mathsf{tuple}(A_0 : \tau_0, A_1 : \tau_1, \ldots, A_n : \tau_n)$, where each A_i is a label and each τ_i is a type. A *tuple node* is a labeled tuple type with a child node $A_i : \tau_i$, for $i = 0, 1 \ldots, n$. Schemas are trees of typed nodes. More specifically, a *schema* is a named type $A : \tau$. Constraints may be imposed over a schema. A *constraint* is either a key or a foreign key constraint.

Instances of types are defined as usual. An instance of a schema tree is a tree of instance nodes. As in [70], an instance node for a schema node of type τ, has a distinct *oid* value, plus a number of children, according to the respective type.

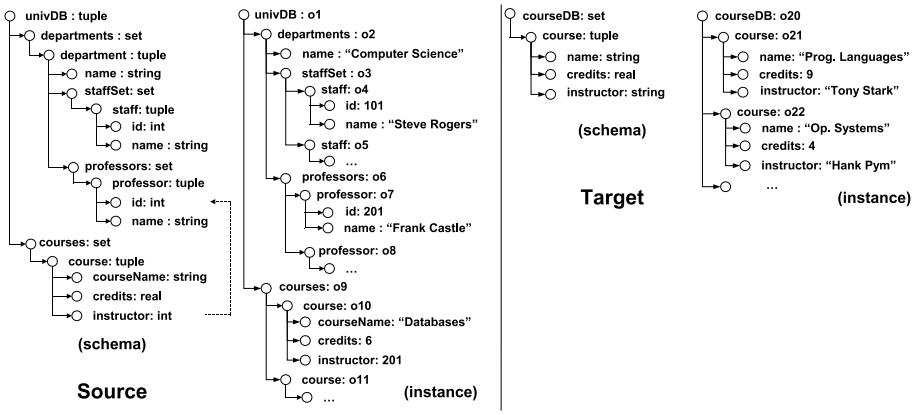

Figure 2.1: An example of Nested Relational Model

Example 2.1 *Consider the data sources in Figure 2.1. The source schema represents a XML schema and contains informations about departments with their staff people and professors, and courses in an University. Professors teach some courses, as indicated by the key-foreign key constraint between professors and courses. On the other side, the target schema represents a relational schema with one table and contains informations about courses with their instructors.*

It can be seen that the data model is essentially a tree-based representation of a nested-relational model, in which set and tuple nodes alternate. Instances are trees as well; while leaf nodes have atomic values – strings, integers, dates etc. – values for intermediate nodes are oids. Foreign key constraints are drawn as dashed edges going from the foreign key to the corresponding primary key. In the example above, each course *tuple in the source schema has a foreign key constraint referencing a* professor *tuple.*

Notice that a relational database is easily represented as a root tuple node, plus a number of children set nodes, one for each relation.

The nested relational model described here is a generalization of the traditional *flat relational model*, introduced by E.F.Codd in 1970 [34]. In the following we will use both data models, depending on which is more convenient in the particular context. The flat relational model is formalized as follows.

Flat Relational Data Model

We fix 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 $\mathit{consts}(I)$ the set of constants occurring in I, and by $\mathit{vars}(I)$ the set of labeled nulls in I. $\mathit{dom}(I)$, its *active domain*, will be $\mathit{consts}(I) \cup \mathit{vars}(I)$. A *ground* instance is an instance I without labeled nulls (where $\mathit{dom}(I) = \mathit{consts}(I)$).

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 and Mapping Scenario

Given two schemas, $\mathbf{S}$ and $\mathbf{T}$, an *embedded dependency* [16] 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. $\phi(\overline{x})$ 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 $\mathbf{T}$.

A *mapping scenario* (also called a *data-exchange scenario* or a *schema mapping* or a *data-exchange setting* [42]) 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 thesis, i.e., the case in which the set of target dependencies Σ_t is empty, we will use the notation $(\mathbf{S}, \mathbf{T}, \Sigma_{st})$.

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* $h : J \to J'$ 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'. 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 $h : J \to J'$ and $h' : J' \to J$.

A solution J is *universal* [42] (denoted $J \in \mathsf{USol}(\mathcal{M}, I)$) iff for every solution K there is an homomorphism from J to K. 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 $J \in \mathsf{USol}(\mathcal{M}, I)$.

The following example summarizes these concepts:

Example 2.2 *Consider the following scenario $\mathcal{M}$, in which the source schema $\mathbf{S}$ has four relation symbols A, B, C, D and the target schema $\mathbf{T}$ has two relation symbols S and T. Let's assume that Σ_t is empty. The set of source-to-target tgds Σ_{st} is:*

$$m_1.\forall x_1, x_2 : A(x_1, x_2) \to \exists Y_1 : S(x_1, Y_1), T(Y_1, x_2)$$
$$m_2.\forall x_3, x_4 : B(x_3, x_4) \to S(x_3, x_4)$$
$$m_3.\forall x_5, x_6 : C(x_5, x_6) \to T(x_5, x_6)$$
$$m_4.\forall x_7 : D(x_7) \to \exists Y_0 : S(x_7, Y_0)$$

and the source instance: $I = \{A(1,2), B(1,3), C(3,2), D(1)\}$.

A possible solution J for $\mathcal{M}$ over I is:

$$J = \{S(1, N_0), T(N_0, 2), S(1,3), T(3,2), S(1, N_1)\}.$$

Here, N_0 and N_1 represent labeled nulls *[42], i.e., distinguished null values that are "invented" to satisfy the existential quantifiers.*

But there may exist more than one solution. For example, also the following ones are solutions:

$$J_0 = \{S(1,3), T(3,2)\} \;\; J_1 = \{S(1,3), T(3,2), S(1, N_1), S(1, N_2)\}.$$

In this example, the solution J is a universal solution*, in fact for every solution K (for example J_0 or J_1) there is an homomorphism from J to K. Instead, the solution J_0 seems to be more compact than J, even if this is not*

required by the mapping specification. Finally, J_1 is also a solution for $\mathcal{M}$ over I, but it has extra information that is not a consequence of the dependencies in Σ_{st}.

While J satisfies the tgds it is possible to see that some of the tuples in it are redundant. Consider for example tuples $t_1 = S(1, N_0)$ and $t_2 = S(1, 3)$. 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 the nulls of t_1 into the constants of t_2, and therefore t_1 itself into t_2. This means that J has an* endomorphism*, i.e., an homomorphism into a sub-instance – the one obtained by removing t_1.*

The core universal solution *[43] is the smallest among the solutions for a given source instance that has homomorphisms into all other solutions. In this example, the core of J is J_0.*

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 $\phi(a(\overline{x})) \subseteq I$. In this case, we write $I \models \phi(a(\overline{x}))$.

Definition 2 ***[Standard Chase]*** *[42] Given an instance $\langle I, J\rangle$, during the* standard chase *a tgd $\phi(\overline{x}) \rightarrow \exists\overline{y}(\psi(\overline{x}, \overline{y}))$ 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 assignining 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]*** *[78] Given an instance $\langle I, J\rangle$, during the* naive chase *a tgd $\phi(\overline{x}) \rightarrow \exists\overline{y}(\psi(\overline{x}, \overline{y}))$ 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 assignining 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 $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st})$, chasing the dependencies in Σ_{st} yields a *canonical solution.*

Example 2.3 *Consider the following scenario $\mathcal{M}$:*

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

and the source instance: $I = \{A(1, 2, 3), B(1, 2), C(1, 2)\}$. 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* [29] 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*, 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 PROVENANCE(t) the symbol R^i, where i is the label of R.

In Example 2.3, assume tgds are labeled as follows:

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

The provenance of tuples in J would be as follows:

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

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 $R^i \in$ 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) $R^i \in$ 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 [42]. Also, the naive chase 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. 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 Chapter 3, 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*.

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 $f^i : \text{CONSTS}^k \rightarrow \text{VARS}$, and, for each value assignment c for $\overline{x}$, we compute the labeled null as the result of f^i over $c(\overline{x})$, $f^i(c(\overline{x}))$.

There are several possible skolemization strategies to pick from and we will discuss them in more detail in Chapter 3.

Just to give an example, 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 rule m, and taking as arguments all universal variables occurring in the conclusion. Consider the following formula $S(1, N_0), T(N_0, 2)$ coming from the tgd m_1 in Example 2.2.
The *uninterpreted* Skolem function for N_0 could have the following form:

$$f_{m_1,N_0}(1, 2)$$

As we will show, in some cases, the standard strategy is not enough to properly handle isomorphisms between tuples and we need to use intepreted Skolem functions.

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 $\overline{x}$.*

Consider the scenario in Example 2.2. Following is a FO-rule from its rewriting:

$$r.\ \forall x_1, x_2 : A(x_1, x_2) \wedge \neg(\exists x_3, x_4, x_5, x_6 : B(x_3, x_4) \wedge C(x_5, x_6) \wedge \\ x_4 = x_5 \wedge x_1 = x_3 \wedge x_2 = x_6) \rightarrow S(x_1, f(x_1, x_2)), T(f(x_1, x_2), x_2)$$

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. 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 )
```

Chapter 3

Generating Core Solutions

As seen earlier, the *core* universal solution was identified as the "optimal" solution, since it is "irredundant" – i.e. it is the smallest among the solutions that preserve the semantics of the exchange – and it represents a "good" instance for answering queries over the target database.

It can be seen how it is crucial to develop algorithms that natively produce executable scripts to compute the core. On the contrary, the solution produced by known schema mapping systems – called a *canonical universal solution* [42] – typically contains quite a lot of redundancy. This is partly due to the fact that computing cores is a challenging task.

Our goal is to introduce a core computation algorithm that lends itself to a more efficient implementation as an executable script and that scales well to large databases. To this end, we rely on two key ideas: the notion of *homomorphism among formulas* and the use of *negation* to rewrite tgds.

3.1 Overview

Consider the following example:

Example 3.1.1 *Consider the mapping scenario in Figure 3.1. A source instance is shown in Figure 3.2.*

A constraint-driven mapping system as Clio would generate for this scenario several mappings, like the ones below.[1] *Mappings are tgds that state how tuples*

[1]Note that the generation of mapping m_1 requires an extension of the algorithms described in [70, 46].

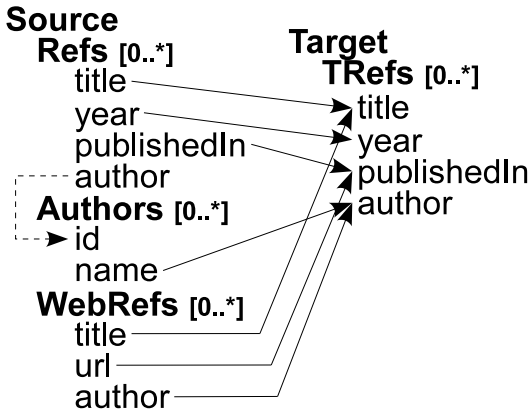

Figure 3.1: Mapping Bibliographic References

should be produced in the target based on tuples in the source. Mappings can be expressed using different syntax flavors. In schema mapping research [46], an XQuery-like syntax is typically used. Data exchange papers use a more classical logic-based syntax that we also adopt in this Chapter.

$$m_1.\ \forall t, y, p, i\colon \text{Refs}(t, y, p, i) \rightarrow \exists N\colon \text{TRefs}(t, y, p, N)$$
$$m_2.\ \forall i, n\colon \text{Auths}(i, n) \rightarrow \exists T, Y, P\colon \text{TRefs}(T, Y, P, n)$$
$$m_3.\ \forall t, y, p, i, n\colon \text{Refs}(t, y, p, i) \wedge \text{Auths}(i, n) \rightarrow \text{TRefs}(t, y, p, n)$$
$$m_4.\ \forall t, p, n\colon \text{WebRefs}(t, p, n) \rightarrow \exists Y\colon \text{TRefs}(t, Y, p, n)$$

Mapping m_3 above states that for every tuple in Refs that has a join with a tuple in Authors, a tuple in TRefs must be produced. Mapping m_1 is needed to copy into the target references that do not have authors, like "The SQL92 Standard". Similarly, mapping m_2 is needed in order to copy names of authors for which there are no references (none in our example). Finally, mapping m_4 copies tuples in WebRefs.

Given a source instance, executing the tgds amounts to running the standard *chase* algorithm on the source instance to obtain the canonical universal solution; note that a natural way to chase the dependencies is to execute them as SQL statements in the DBMS.

Refs

title	year	publishedIn	author
A Relational Model...	1970	CACM	a1
DB Teaching Materials	1985	http://www....	a2
The SQL92 Standard	1992	ANSI Stds	NULL

WebRefs

title	url	author
DB Teaching Materials	http://www....	C. Date

Auths

id	name
a1	E. F. Codd
a2	C. Date

Target: TRefs

title	year	publishedIn	author
A Relational Model...	1970	CACM	E. F. Codd
DB Teaching Materials	1985	http://www...	C. Date
The SQL92 Standard	1992	ANSI Stds	NULL
DB Teaching Materials	NULL	http://www...	C. Date
A Relational Model...	1970	CACM	NULL
DB Teaching Materials	1985	http://www...	NULL
NULL	NULL	NULL	E. F. Codd
NULL	NULL	NULL	C. Date

Figure 3.2: Instances for the References Scenario

These expressions materialize the target instance in Figure 3.2. While this instance satisfies the tgds, still it contains many redundant tuples, those with a gray background. As shown in [46], for large source instances the amount of redundancy in the target may be very large, thus impairing the efficiency of the exchange and the query answering process. This has motivated several practical proposals [30, 46, 27] 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.

Data exchange research [43] has introduced the notion of *core* solutions as "optimal" solutions for a data exchange problem. Consider for example tuples t_1 = *(null, null, null, E.F.Codd)* and t_2 = *(A Relational Model..., 1970, CACM, E.F.Codd)* in Figure 3.2. Notice that t_1 is redundant with respect to t_2 because there is an *homomorphism* from t_1 to t_2. The core of the solution in Figure 3.2 is in fact the portion of the *TRefs* table with a white background.

A possible approach to the generation of the core for a relational data exchange problem is to generate a canonical solution by chasing the tgds, and then to apply a post-processing algorithm for core identification. Several polynomial algorithms have been identified to this end [43, 51]. These algorithms provide a very general solution to the problem of computing core solutions for a data exchange setting. Also, an implementation of the core-computation algorithm in [51] has been developed [74], thus making a significant step towards the goal of integrating core computations in schema mapping systems.

However, experience with these algorithms shows that, although polynomial, they require very high computing times since they look for all possible endomorphisms among tuples in the canonical solution. As a consequence, they hardly scale to large mapping scenarios. Our goal is to introduce a core computation algorithm that lends itself to a more efficient implementation as an executable script and that scales well to large databases. To this end, in the following sections we introduce two key ideas: the notion of *homomorphism among formulas* and the use of *negation* to rewrite tgds.

The first intuition is that it is possible to analyze the set of formulas in order to recognize when two tgds may generate redundant tuples in the target. This happens when it is possible to find a homomorphism between the right-hand sides of the two tgds. Consider tgds m_2 and m_3 in the Example 3.1.1; with an abuse of notation, we consider the two formulas as sets of tuples, with existentially quantified variables that correspond to nulls; it can be seen that the conclusion $TRefs(T, Y, P, n)$ of m_2 can be mapped into the conclusion $TRefs(t, y, p, n)$ of m_3 by the following mapping of variables: $T \rightarrow t$, $Y \rightarrow y$, $P \rightarrow p$. This gives us a nice necessary condition to intercept possible redundancy (i.e., possible endomorphisms among tuples in the canonical solution). Note that the condition is merely a necessary one, since the actual generation of endomorphisms among facts depends on values coming from the source. Note also that we are checking for the presence of homomorphisms among formulas, i.e., conclusions of tgds, and not among instance tuples; since the number of tgds is typically much smaller than the size of an instance, this task can be carried out quickly.

A second important intuition is that, whenever we identify two tgds m, m' such that there is a homomorphism between conclusions, as above, we may prevent the generation of redundant tuples in the target instance by executing them according to the following strategy:

(i) generate target tuples for m, the "more informative" mapping;

(ii) for m', generate only those tuples that actually add some new content to the target.

In order to do this, we may rewrite the original tgds into a new set of dependencies by adding to the premise of m' the negation of the premise of m. The set of rewritten dependencies for Example 3.1.1 is the following (universally quantified variables have been omitted since they should be clear from the context):

$$
\begin{aligned}
&m'_3.\ \mathit{Refs}(t,y,p,i) \wedge \mathit{Auths}(i,n) \rightarrow \mathit{TRefs}(t,y,p,n)\\
&m'_1.\ \mathit{Refs}(t,y,p,i) \wedge \neg(\mathit{Refs}(t,y,p,i) \wedge \mathit{Auths}(i,n))\\
&\qquad\qquad \rightarrow \exists N\colon \mathit{TRefs}(t,y,p,N)\\
&m'_2.\ \mathit{Auths}(i,n) \wedge \neg(\mathit{Refs}(t,y,p,i) \wedge \mathit{Auths}(i,n)) \wedge\\
&\qquad \neg(\mathit{WebRefs}(t,p,n)) \rightarrow \exists X,Y,Z\colon \mathit{TRefs}(X,Y,Z,n)\\
&m'_4.\ \mathit{WebRefs}(t,p,n) \wedge \neg(\mathit{Refs}(t,y,p,i) \wedge \mathit{Auths}(i,n))\\
&\qquad\qquad \rightarrow \exists Y\colon \mathit{TRefs}(t,Y,p,n)
\end{aligned}
$$

Once we have rewritten the original tgds in this form, we can easily generate an executable transformation under the form of relational algebra expressions. Here, negations become difference operators; in this simple case, nulls can be generated by outer-union operators, $\cup_*$, that have the semantics of the `insert into` SQL statement:[2]

$$
\begin{aligned}
&m'_3:\ \mathit{TRefs} = \pi_{t,y,p,n}(\mathit{Refs} \bowtie \mathit{Auths})\\
&m'_1:\quad \cup_*(\pi_{t,y,p}(\mathit{Refs}) - \pi_{t,y,p}(\mathit{Refs} \bowtie \mathit{Auths}))\\
&m'_2:\quad \cup_*(\pi_n(\mathit{Auths}) - \pi_n(\mathit{Refs} \bowtie \mathit{Auths}) - \pi_a(\mathit{WebRefs}))\\
&m'_4:\quad \cup_*(\pi_{t,p,n}(\mathit{WebRefs}) - \pi_{t,p,n}(\mathit{Refs} \bowtie \mathit{Auths}))
\end{aligned}
$$

The algebraic expressions above can be easily implemented in an executable script, say in SQL or XQuery, to be run in any database engine. As a consequence, there is a noticeable gain in efficiency with respect to the algorithms for core computation proposed in [43, 51, 74].

Despite the fact that this example looks pretty simple, it captures a quite common scenario. However, removing redundancy from the target may be a much more involved process, as discussed in the following.

Example 3.1.2 *Consider now the mapping scenario in Figure 3.3. The target has two tables, in which genes reference their protein via a foreign key. In the source we have data coming from two different biology databases. Data in the PDB tables comes from the* Protein Database*, which is organized in a way that*

[2]Note also that in the more general case Skolem functions are needed to properly generate nulls.

is similar to the target. On the contrary, the EMBL table contains data from the popular EMBL *repository; there, tuples need to be partitioned into a gene and a protein tuple. In this process, we need to "invent" a value to be used as a key-foreign key pair for the target. This is usually done using a* Skolem function *[70].*

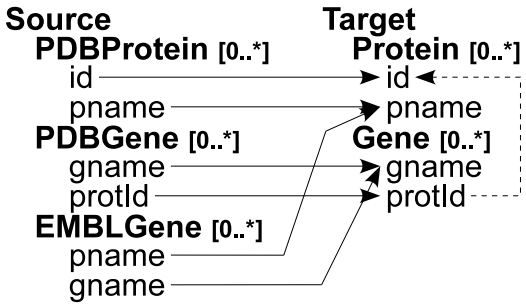

Figure 3.3: Genes

This transformation can be expressed using the following tgds:

m_1. $PDBProtein(i, p) \rightarrow Protein(i, p)$
m_2. $PDBGene(g, i) \rightarrow Gene(g, i)$
m_3. $EMBLGene(p, g) \rightarrow \exists N$: $Gene(g, N) \wedge Protein(N, p)$

Sample instances are in Figure 3.4. It can be seen that the canonical solution contains a smaller endomorphic image – the core – since the tuples (*14-A, N2*) and (*N2, 14-A-antigen*), where *N2* was invented during the chase, can be mapped to the tuples (*14-A, p1*) and (*p1, 14-A-antigen*). In fact, if we look at the right-hand sides of tgds, we see that there is a homomorphism from the right-hand side of m_3, $\{Gene(g, N), Protein(N, p)\}$, into the right-hand sides of m_1 and m_2, $\{Gene(g, i), Protein(i, p)\}$: it suffices to map N into i. However, this homomorphism is a more complex one with respect to those in the previous example. There, we were mapping the conclusion of one tgd into the conclusion of another. We may rewrite the original tgds as follows to obtain the core:

m'_1. $PDBProtein(i, p) \rightarrow Protein(i, p)$
m'_2. $PDBGene(g, i) \rightarrow Gene(g, i)$
m'_3. $EMBLGene(p, g) \wedge \neg(PDBGene(g, i) \wedge PDBProtein(i, p))$
$\rightarrow \exists N \;\; Gene(g, N) \wedge Protein(N, p)$

PDBProtein

id	pname
p1	14-A-antigen
p2	ACC synthase

PDBGene

gname	protId
14-A	p1
16-ACC	p2

EMBLGene

gname	pname
14-A	14-A-antigen
15-B	alpha-precursor

Protein (Target)

id	pname
p1	14-A-antigen
p2	ACC synthase
N1	alpha precursor
N2	14-A-antigen

Gene (Target)

gname	proteinId
14-A	p1
16-ACC	p2
15-B	*N1*
14-A	*N2*

Figure 3.4: Instances for the genes example

From the algebraic viewpoint, mapping m_3' above requires to generate in *Gene* and *Protein* tuples based on the following expression:

$$EMBLGene - \pi_{p,g}(PDBGene \bowtie PDBProtein)$$

In the process, we also need to generate the appropriate Skolem functions to correlate tuples in *Gene* with the corresponding tuples in *Protein*. A key difference with respect to the previous example is that there can be a much larger number of possible rewritings for a tgd like m_3, and therefore a larger number of additional joins and differences to compute. This is due to the fact that we need to look for homomorphisms of every single atom into other atoms appearing in right-hand sides of the tgds, and then combine them in all possible ways to obtain the rewritings. To give an example, suppose the source also contains tables *XProtein*, *XGene* that write tuples to *Protein* and *Gene*; then, we might have to rewrite m_3 by adding the negation of four different joins: (*i*) *PDBProtein* and *PDBGene*; (*ii*) *XProtein*, *XGene*; (*iii*) *PDBProtein* and *XGene*; (*iv*) *XProtein* and *PDBGene*. This obviously increases the time needed to execute the exchange.

Special care must be devoted to tgds containing *self-joins* in the conclusion, i.e., tgds in which the same relation symbol occurs more than once in the right-hand side. One example of this kind is the "self-join" scenario in STMark [8], or the "RS" scenario in [43].

Consider a simplified version of the latter, in which the source schema contains a single relation R, the target schema a single relation S, and a single tgd is given:

$$m_1.\ R(a,b) \rightarrow \exists x_1, x_2 : S(a,b,x_1) \wedge S(b,x_2,x_1)$$

Assume table R contains a single tuple: $R(1,1)$; by chasing m_1, we generate two tuples in the target: $S(1,1,N1)$, $S(1,N2,N1)$. It is easy to see that this set has a proper endomorphism, and therefore its core corresponds to the single tuple $S(1,1,N1)$.

Even though the example is quite simple, eliminating this kind of redundancy in more complex scenarios can be rather tricky. Intuitively, the techniques discussed above are of little help, since, regardless of how we rewrite the premise of the tgd, on a tuple $R(1,1)$ the chase will either generate two tuples or none of them.

As a consequence, we introduce a more sophisticate treatment of these cases. These ideas are made more precise in the following sections.

Based on the ideas above, in this chapter we introduce a number of novel algorithms that contribute to bridge the gap between the practice of mapping generation and the theory of data exchange. In particular:

(*i*) +Spicy integrates the computation of core solutions in the mapping generation process in a highly efficient way; after a set of tgds has been generated based on the input provided by the user, core solutions are computed by a natural rewriting of the tgds as a new set of dependencies; this allows for an efficient implementation of the rewritten mappings using common runtime languages like SQL (or XQuery) and guarantees very good performances, orders of magnitude better than those of previous core-computation algorithms; we show in the following that our strategy scales up to large databases in practical scenarios;

(*ii*) we classify data exchange settings in several categories, based on the structure of the mappings and on the complexity of computing the core; correspondingly, we identify several approximations of the core of increasing quality; the rewriting algorithm is designed in a modular way, so that, in those cases in which computing the core requires heavy computations, it is possible to fine tune the trade off between quality and computing times;

(*iii*) finally, the rewriting algorithm can be applied both to mappings generated by the mapping system, or to pre-existing tgds that are provided as part of the input. +Spicy is the first mapping system that brings together a sophis-

ticate and expressive mapping generation algorithm with an efficient strategy to compute irredundant solutions.

In light of these contributions, we believe this work makes a significant advancement towards the goal of integrating data exchange concepts and core computations into existing database technology.

3.2 Preliminaries

In this Section we give some new definitions related to data exchange and core computation, needed for the results of this thesis. Since in this Chapter we focus on relational data, for convenience we will adopt the flat relational model presented in Chapter 2 and, as it is common in data exchange papers, we use a logic-based syntax.

Computing Core Solutions

Given a scenario $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st})$, and an instance I, the *core* [43] 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 [43] 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*.

In the following sections, we detail several algorithms that, given a mapping scenario, rewrite the given tgds as a set of FO-rules that represent a *core schema mapping*.

Definition 1 ***[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.*

A very similar notion of *laconic schema mapping* was introduced in [78]. Note that we concentrate on data exchange settings expressed as a set of source-to-target tgds, i.e., we do not consider target tgds and egds. In fact, it was shown in [78] that it is in general not possible to rewrite a scenario with target tgds into a laconic one. The authors conjecture that the same also holds for target egds.

We also make the assumption that the set Σ_{st} is *source-based*.

Definition 2 ***[Source-Based Tgd]*** *A tgd* $\phi(\overline{x}) \rightarrow \exists\overline{y}(\psi(\overline{x},\overline{y}))$ *is* source-based *if:* (i) *the left-hand side* $\phi(\overline{x})$ *is not empty;* (ii) *the vector of universally quantified variables* $\overline{x}$ *is not empty;* (iii) *at least one of the variables in* $\overline{x}$ *appears in the right hand side* $\psi(\overline{x},\overline{y})$.

This definition, while restricting the variety of tgds handled by the algorithm, captures the notion of a "useful" tgd in a schema mapping scenario. In fact, note that tgds in which the left-hand side is empty or it contains no universally quantified variables – like, for example $\rightarrow \exists X, Y : T(X, Y)$, or $\forall a : S(a)$ $\rightarrow \exists X, Y : R(X, Y) \wedge S(Y, X)$ – would generate target tuples made exclusively of nulls, which are hardly useful in practical cases.

Besides requiring that tgds are source-based, without loss of generality we also 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. To formalize this second 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 3 ***[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.

Example 3.2.1 *Consider the following set of tgds* Σ_{st}*:*

$m_1.\ A(x_1, x_2, x_3) \rightarrow \exists Y_0, Y_1, Y_2 : S(x_1, Y_0), T(x_2, Y_0, Y_1), U(Y_2, x_2), V(Y_2, x_3)$
$m_2.\ B(x_4, x_5) \rightarrow \exists Y_3, Y_4 : S(x_4, Y_3), T(Y_3, x_5, Y_4), U(Y_4, x_4)$

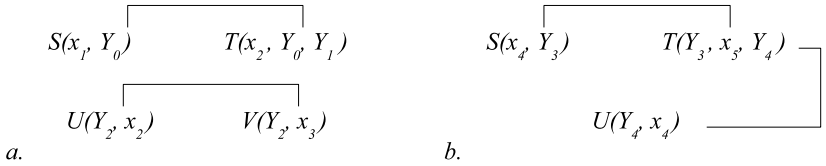

Figure 3.5: Example of dual Gaifman graphs

Dual Gaifman graphs for tgd m_1 and m_2 are represented respectively in Figure 3.5 (a) and (b). Notice that m_2 is in normal form, since its dual Gaifman graph is connected; on the contrary, m_1 is not in normal form. Applying the transformation NORMALIZE(Σ_{st})*, we will obtain a new set of tgds, Σ'_{st}, in which each tgd is rewritten in normal form. Σ'_{st} is as follows:*

m'_1. $A(x_1, x_2, x_3) \rightarrow \exists Y_0, Y_1 : S(x_1, Y_0), T(x_2, Y_0, Y_1)$
m''_1. $A(x_1, x_2, x_3) \rightarrow \exists Y_2 : U(Y_2, x_2), V(Y_2, x_3)$
m'_2. $B(x_4, x_5) \rightarrow \exists Y_3, Y_4 : S(x_4, Y_3), T(Y_3, x_5, Y_4), U(Y_4, x_4)$

3.3 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 $\overline{a}$. More formally:

Definition 4 ***[Witness Block]*** *Given a scenario $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st})$, a source instance I, and an 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, $\mathcal{W}_{m,\overline{c}}^{<I,J>}$ 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 3.3.1 *Reconsider the scenario in Example 2.2. We report here the set of source-to-target tgds Σ_{st}:*

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

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

$$J = \{S(1, N_0), T(N_0, 2), S(1,3), T(3,2), S(1, N_1)\}.$$

Following are the witness blocks for J:

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

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

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

Example 3.3.2 *Consider now the following scenario $\mathcal{M}$:*

$$\begin{array}{l}
m_1.\ R(x_0,x_1,x_2) \rightarrow \exists Y_0,Y_1,Y_2,Y_3,Y_4,Y_5 : S(Y_0,x_0,Y_2,Y_3), \\
\qquad\qquad\qquad\qquad\qquad\qquad S(Y_0,x_1,x_1,Y_1), S(Y_4,x_2,Y_5,Y_1) \\
m_2.\ R(z_0,z_1,z_2) \rightarrow \exists V_0,V_1,V_2,V_3,V_4,V_5 : S(V_0,z_1,z_2,V_2), \\
\qquad\qquad\qquad\qquad\qquad\qquad S(V_0,z_2,z_1,V_1), S(V_3,V_4,V_5,V_1)
\end{array}$$

and the source instance: $I = \{R(e,a,a), R(e,a,b)\}$. The canonical universal solution J is the following:

$$\begin{array}{lcll}
J & = & \{ & S(N_0,e,N_1,N_2), S(N_0,a,a,N_3), S(N_4,a,N_5,N_3), \\
 & & & S(N_6,e,N_7,N_8), S(N_6,a,a,N_9), S(N_{10},b,N_{11},N_9), \\
 & & & S(N_{12},a,a,N_{13}), S(N_{12},a,a,N_{14}), S(N_{15},N_{16},N_{17},N_{14}), \\
 & & & S(N_{18},a,b,N_{19}), S(N_{18},b,a,N_{20}), S(N_{21},N_{22},N_{23},N_{20}) \quad \}
\end{array}$$

Following are the four sets of witness blocks:

$$\begin{array}{lll}
\mathcal{W}^{<I,J>}_{m_1,\langle e,a,a\rangle} = \{ & \{S(N_0,e,N_1,N_2), S(N_0,a,a,N_3), S(N_4,a,N_5,N_3)\}, & \\
 & \{S(N_0,e,N_1,N_2), S(N_0,a,a,N_3)\}, & \\
 & \{S(N_6,e,N_7,N_8), S(N_6,a,a,N_9)\} & \} \\
\mathcal{W}^{<I,J>}_{m_1,\langle e,a,b\rangle} = \{ & \{S(N_6,e,N_7,N_8), S(N_6,a,a,N_9), S(N_{10},b,N_{11},N_9)\} & \} \\
\mathcal{W}^{<I,J>}_{m_2,\langle e,a,a\rangle} = \{ & \{S(N_{12},a,a,N_{13}), S(N_{12},a,a,N_{14}), S(N_{15},N_{16},N_{17},N_{14})\}, & \\
 & \{S(N_{12},a,a,N_{13}), S(N_{12},a,a,N_{14})\}, \{S(N_{12},a,a,N_{13})\}, & \\
 & \{S(N_{12},a,a,N_{14})\}, \{S(N_0,a,a,N_3)\}, \{S(N_6,a,a,N_9)\} & \} \\
\mathcal{W}^{<I,J>}_{m_2,\langle e,a,b\rangle} = \{ & \{S(N_{18},a,b,N_{19}), S(N_{18},b,a,N_{20}), S(N_{21},N_{22},N_{23},N_{20})\}, & \\
 & \{S(N_{18},a,b,N_{19}), S(N_{18},b,a,N_{20})\} & \}
\end{array}$$

The core of J is as follows:

$$J_0 = \{ \begin{array}{l} S(N_6, e, N_7, N_8), S(N_6, a, a, N_9), S(N_{10}, b, N_{11}, N_9) \\ S(N_{18}, a, b, N_{19}), S(N_{18}, b, a, N_{20}) \end{array} \}$$

The witness blocks for J_0 are as follows:

$$\begin{array}{lcll} \mathcal{W}^{<I,J_0>}_{m_1,\langle e,a,a\rangle} & = & \{ & \{S(N_6, e, N_7, N_8), S(N_6, a, a, N_9)\} \} \\ \mathcal{W}^{<I,J_0>}_{m_1,\langle e,a,b\rangle} & = & \{ & \{S(N_6, e, N_7, N_8), S(N_6, a, a, N_9), S(N_{10}, b, N_{11}, N_9)\} \} \\ \mathcal{W}^{<I,J_0>}_{m_2,\langle e,a,a\rangle} & = & \{ & \{S(N_6, a, a, N_9)\} \} \\ \mathcal{W}^{<I,J_0>}_{m_2,\langle e,a,b\rangle} & = & \{ & \{S(N_{18}, a, b, N_{19}), S(N_{18}, b, a, N_{20})\} \} \end{array}$$

An important observation is that other algorithms for core computation ([43, 51, 78]) 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 3.3.3 *Consider now the following scenario $\mathcal{M}$:*

$$\begin{array}{l} m_1.\ R(x_0, x_1, x_2, x_3) \rightarrow \exists Y_0, Y_1, Y_2, Y_3, Y_4 : S(x_3, x_0, x_0, Y_0, x_1), \\ \qquad\qquad\qquad\qquad\qquad\qquad S(Y_1, x_0, x_0, Y_0, x_0), S(Y_1, x_2, Y_2, Y_3, Y_4) \end{array}$$

and the source instance: $I = \{R(d, d, c, d), R(c, d, c, d)\}$. The core universal solution for $\mathcal{M}$ over I is as follows:

$$J_0 = \{S(d, d, d, N_0, d), S(d, c, c, N_5, d), S(N_6, c, c, N_5, c)\}$$

The witness blocks for J_0 are as follows:

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

Notice how the witness block in $\mathcal{W}^{<I,J_0>}_{m_1,\langle d,d,c,d\rangle}$ *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 an 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'.

There are several ways in which tuples can be made redundant in a solution. Generally speaking, tuples are redundant whenever they introduce unnecessary nulls. One example of this is the case of a witness block w for some tgd m and assignment a such that there is a "more compact" witness block w' for the same tgd and assignment.

To give an example, consider Example 3.3.2; in $\mathcal{W}^{<I,J>}_{m_1,\langle e,a,a\rangle}$, the witness block $w_1 = \{S(N_0, e, N_1, N_2), S(N_0, a, a, N_3), S(N_4, a, N_5, N_3)\}$ is made redundant by witness block $w_2 = \{S(N_0, e, N_1, N_2), S(N_0, a, a, N_3)\}$; w_2, in fact, contains a lower number of nulls than w_1. To formalize this notion, we introduce a classification of homomorphisms, as follows.

Definition 5 ***[Classification of Homomorphisms]*** *Given two instances* J, J', *and an homomorphism* $h : J \to J'$*:*

- h *is* proper *if* $|J| < |J'|$*, i.e.,* J' *contains at least one atom that is not the image of an atom of* J*; in symbols, we write that* $J < J'$*; otherwise, we say that* h *is* surjective*, or that it is a* surjection*;*
- h *is* compacting *if it is a surjection, and* $|vars(J')| < |vars(J)|$*; we write* $J \prec J'$ *if there is a compacting homomorphism of* J *into* 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* $J \cong J'$*.*

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". Consider again the Example 3.3.2; witness block $w_2 = \{S(N_0, e, N_1, N_2), S(N_0, a, a, N_3)\}$, which is a maximal block for m_1 and $\langle e, a, a\rangle$ is made redundant by witness block $w_3 = \{S(N_6, e, N_7, N_8), S(N_6, a, a, N_9),\ S(N_{10}, b, N_{11}, N_9)\}$, since there is a proper homomorphism of w_2 into w_3.

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: (a) first, we shall select the most compact witness blocks in any set $\mathcal{W}_{m,\overline{a}}^{<I,J>}$, i.e., all maximal elements with respect to the $\prec$ partial order; (b) 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:

$$\begin{array}{lcl} \textsc{max-compact}(\mathcal{W}) & = & \{w \mid w \in \mathcal{W} \ \wedge \ \neg\exists\, w' \in \mathcal{W} : w \prec w'\} \\ \textsc{max-informative}(\mathcal{W}) & = & \{w \mid w \in \mathcal{W} \ \wedge \ \neg\exists\, w' \in \mathcal{W} : w < w'\} \end{array}$$

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 3.3.4 *Consider the following scenario $\mathcal{M}$:*

$$\begin{array}{l} m_1.\ R(x_3, x_4) \rightarrow \exists Y_3, Y_4 : S(x_3, x_4, Y_3), S(x_3, Y_4, Y_3) \\ m_2.\ R(x_5, x_6) \rightarrow \exists Y_5, Y_6 : S(x_5, x_6, Y_5), S(Y_6, x_6, Y_5) \end{array}$$

and the source instance: $I = \{R(1, 2)\}$. The canonical universal solution J for $\mathcal{M}$ over I is the following:

$$J \ = \ \{ \quad S(1, 2, N_3), S(1, N_4, N_3), S(1, 2, N_5), S(N_6, 2, N_5) \quad \}$$

Following are some sets of witness blocks for J:

$$\begin{array}{lcl} \mathcal{W}_{m_1,\langle 1,2\rangle}^{<I,J>} & = & \{ \quad \{S(1, 2, N_3), S(1, N_4, N_3)\}, \{S(1, 2, N_3)\}, \{S(1, 2, N_5)\} \quad \} \\ \mathcal{W}_{m_2,\langle 1,2\rangle}^{<I,J>} & = & \{ \quad \{S(1, 2, N_5), S(N_6, 2, N_5)\}, \{S(1, 2, N_3)\}, \{S(1, 2, N_5)\} \quad \} \end{array}$$

By taking the union of the set of maximal witness blocks, we obtain the following solution: $J^* = \{S(1, 2, N_3), S(1, 2, N_5)\}$ *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.*

After we have selected a set of maximal witness blocks, in order 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 3.3.5 *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:*

$$J_0 = \bigcup \text{REDUCE}(\text{MAX-INFORMATIVE}(\text{MAX-COMPACT}(\mathcal{W}^{<I,J>}))) \tag{3.1}$$

Then, J_0 *is the core of* J*.*

The proof is in the Appendix.

3.4 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 order to do this, we shall rely on the characterization of the core introduced in Section 3.3. 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.

Expansions and Formula Homomorphisms

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. To discuss how this is done, we shall mainly refer to the scenario in Example 3.3.2, of which we report the tgds here, complete of labels.

$$\begin{aligned} m_1.\ R(x_0, x_1, x_2) \rightarrow \exists Y_0, Y_1, Y_2, Y_3, Y_4, Y_5 : S^1(Y_0, x_0, Y_2, Y_3), \\ S^2(Y_0, x_1, x_1, Y_1), S^3(Y_4, x_2, Y_5, Y_1) \\ m_2.\ R(x_3, x_4, x_5) \rightarrow \exists Y_6, Y_7, Y_8, Y_9, Y_{10}, Y_{11} : S^4(Y_6, x_4, x_5, Y_8), \\ S^5(Y_6, x_5, x_4, Y_7), S^6(Y_9, Y_{10}, Y_{11}, Y_7) \end{aligned}$$

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: it states that the target must contain a number of tuples in S that satisfy the two joins in the tgd conclusion. Recall that in our labeling system, atom S^i corresponds to all tuples in relation S whose provenance contains label i. The formula:

$$\epsilon_{11}.\ S^1(Y_0, x_0, Y_2, Y_3) \wedge S^2(Y_0, x_1, x_1, Y_1) \wedge S^3(Y_4, x_2, Y_5, Y_1)$$

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. However, not all witness blocks. 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, S^3, since they may also come from S^4, S^5 or S^6; (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^1 and one from S^2, the second one in join with itself on the last

attribute – i.e., S^2 is used to "cover" the S^3 atom. However, this may work as long as the two atoms generate tuples that do not conflict with the constants in the base expansion; in our example, the values generated by the S^2 atom must be consistent with those that would be generated by the S^3 atom we are eliminating, i.e., $x_1 = x_2$. We write this second expansion as follows:

$$\epsilon_{12}.\ S^1(Y_0, x_0, Y_2, Y_3) \wedge S^2(Y_0, x_1, x_1, Y_1) \wedge \\ \exists x_{2b}, Y_{4b}, Y_{5b} : (S^3(Y_{4b}, x_{2b}, Y_{5b}, Y_1) \wedge x_1 = x_{2b})$$

It is possible to see that – from the algebraic viewpoint – the formula requires to compute a join between S^1 and S^2, and then an *intersection* with the content of S^3. This is even more apparent if we look at another possible expansion, the one that replaces the three atoms with a single covering atom from S^5:

$$\epsilon_{13}.\ S^5(Y_6, x_5, x_4, Y_7) \wedge \exists x_{0b}, x_{1b}, x_{2b}, Y_{0b}, Y_{1b}, Y_{2b}, Y_{3b}, Y_{4b}, Y_{5b} : \\ (S^1(Y_{0b}, x_{0b}, Y_{2b}, Y_{3b}) \wedge S^2(Y_{0b}, x_{1b}, x_{1b}, Y_{1b}) \wedge S^3(Y_{4b}, x_{2b}, Y_{5b}, Y_{1b}) \wedge \\ x_5 = x_{0b} \wedge x_5 = x_{1b} \wedge x_4 = x_{1b} \wedge x_5 = x_{2b})$$

In algebraic terms, expansion ϵ_{13} corresponds to taking the intersection on the appropriate attributes of S^5 with the base expansion, i.e., $S^1 \bowtie S^2 \bowtie S^3$.

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

$$\epsilon_{21}.\ S^4(Y_6, x_4, x_5, Y_8) \wedge S^5(Y_6, x_5, x_4, Y_7) \wedge S^6(Y_9, Y_{10}, Y_{11}, Y_7)$$

However 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 third atom is useless (it generates tuples that only contain nulls and no constants).[3] A more interesting expansion is the following;

$$\epsilon_{22}.\ S^4(Y_6, x_4, x_5, Y_8) \wedge S^5(Y_6, x_5, x_4, Y_7)$$

Notice that no intersection is present, here, since we are removing from the base expansion an atom that only contains existential variables.

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 [57]. We find it useful to define homomorphisms among variable occurrences, and not among variables. In the following, we use the notation $\varphi(\overline{x}, \overline{y})$ to denote a (labeled) conjunctive formula with universally quantified variables $\overline{x}$, and existentially quantified variables $\overline{y}$.

[3] This was first noted in [78] with respect to the different notion of a *fact-block type*.

Definition 6 ***[Variable Occurrence]*** *Given an atom* $R^l(A_1 : v_1, \ldots, A_k : v_k)$ *in a formula* $\varphi(\overline{x}, \overline{y})$*, a* variable occurrence *is a pair* $R^l.A_j : v_i$*. A variable occurrence* $R^l.A_j : v_i$ *in* $\varphi(\overline{x}, \overline{y})$ *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 $occ(\varphi(\overline{x}, \overline{y}))$ the set of all variable occurrences in $\varphi(\overline{x}, \overline{y})$; $u\text{-}occ(\varphi(\overline{x}, \overline{y}))$, $e\text{-}occ(\varphi(\overline{x}, \overline{y}))$ 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 7 ***[Formula Homomorphism]*** *Given two conjunctive formulas* $\varphi(\overline{x}, \overline{y})$*,* $\varphi'(\overline{x}', \overline{y}')$*, a* formula homomorphism *is an injective mapping* h^f *from the set* $occ(\varphi(\overline{x}, \overline{y}))$ *to* $occ(\varphi'(\overline{x}', \overline{y}'))$ *such that: (i)* h^f *maps universal occurrences into universal occurrences; (ii) for each atom* $R^l(A_1 : v_1, \ldots, A_k : v_k) \in \varphi(\overline{x}, \overline{y})$*, it is the case that* $R(h^f(R^l.A_1 : v_1), \ldots, h^f(R^l.A_k : v_k)) \in \varphi'(\overline{x}', \overline{y}')$*; (iii) 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* $y' \in \overline{y}'$*.*

It is useful to classify formula homomorphisms in several categories, as follows.

Definition 8 ***[Classification of Formula Homomorphisms]*** *Given two formulas* $\varphi(\overline{x}, \overline{y})$*,* $\varphi'(\overline{x}', \overline{y}')$*, and a formula homomorphism* h^f *from* $occ(\varphi(\overline{x}, \overline{y}))$ *to* $occ(\varphi'(\overline{x}', \overline{y}'))$*,*

- h^f *is said to be* proper *if* $|\varphi(\overline{x}, \overline{y})| < |\varphi'(\overline{x}', \overline{y}')|$*, i.e., there is at least one atom in* $\varphi'(\overline{x}', \overline{y}')$ *which is not the image of an atom of* $\varphi(\overline{x}, \overline{y})$*; on the contrary,* h^f *is* surjective*, or a* surjection*, if all atoms in* $\varphi'(\overline{x}', \overline{y}')$ *are the image of some atom in* $\varphi(\overline{x}, \overline{y})$*;*

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

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.

Note 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. Therefore, the homomorphism may be "realized" or not among facts, i.e., instances of the formulas, depending on values assumed by the universal variables. Given a formula homomorphism h^f, we introduce several sets of equalities among universal variables that are associated with h^f:

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

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

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

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

The set EQUAL_{h^f} will be the union of the two, as follows:

$$\text{EQUAL}_{h^f}(\overline{x}, \overline{x}') = \text{INTERSECT}_{h^f}(\overline{x}, \overline{x}') \cup \text{JOINS}_{h^f}(\overline{x}')$$

We are now ready to introduce the notion of an *expansion*. To do that, we shall repeatedly make use of *labeled formulas*, i.e., formulas in which atom labels may appear. Given a formula $\psi(\overline{x}, \overline{y})$, the associated *labeled formula* is a formula $\psi^l(\overline{x}, \overline{y})$ in which each atom $R(\overline{x}, \overline{y})$ is replaced by a labeled atom $R^i(\overline{x}, \overline{y})$. In the following, we will often go from a labeled formula $\psi^l(\overline{x}, \overline{y})$ to its unlabeled version $\psi(\overline{x}, \overline{y})$.

Given a scenario $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st})$, we shall denote by $\bigcup_i \psi_i(\overline{x}_i, \overline{y}_i)$ the union of all tgd conclusions in Σ_{st}.

Definition 9 ***[Expansion]*** *Given a tgd* $m.\, \phi(\overline{x}_2) \rightarrow \exists \overline{y}_2(\psi^l(\overline{x}_2, \overline{y}_2))$ *in* Σ_{st}, *an* expansion *of* m *is a logical formula 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* $\bigcup_i \psi_i^l(\overline{x}_i, \overline{y}_i)$ *such that there is 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.*

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 show 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: (i) $J \models a_1(\chi^l(\overline{x}_1, \overline{y}_1))$; (ii) there exists an assignment a_2 such that $J \models a_2(\psi^l(\overline{x}_2, \overline{y}_2))$; (iii) a_1, a_2 are such that $\textsc{equal}_{h^f{}_\epsilon}(a_1(\overline{x}_1), a_2(\overline{x}_2))$ evaluates to true.

Algorithm 1 describes how to derive all expansions of a tgd m, denoted by $\textsc{expan}(m)$.

In order to find them, for each tgd $m : \forall \overline{x} : \phi(\overline{x}) \rightarrow \exists \overline{y}(\psi(\overline{x}, \overline{y}))$ we need to consider the set $\mathcal{R}^{pow}$ of all multisets of atoms from $\bigcup\{\psi_i^l(\overline{x}_i, \overline{y}_i) \mid \phi_i(\overline{x}_i) \rightarrow \exists \overline{y}_i(\psi_i(\overline{x}_i, \overline{y}_i)) \in \Sigma_{st}\}$, and select those into which $\psi(\overline{x}, \overline{y})$ has a surjective homomorphism. Some care must be devoted to properly renaming variables; in particular, whenever multiple copies of the same atom appear in a multiset, we assume that they have been properly renamed to avoid variable collisions.

It can be seen that the number of expansions of a tgd conclusion may significantly increase when the number of self-joins increase, and is in general exponential in the size of the input tgds.

Rewriting Expansions

Expansions represent all possible ways in which the original constraints may be satisfied, and in fact they capture all possible witness blocks of a tgd. However, to identify redundant witness blocks, we need to properly rewrite expansions according to their formula homomorphisms, as follows.

Definition 10 ***[More Compact, More Informative]*** *Given expansions:*

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

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

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 $\text{EXPAN}(m)$

Let $\text{EXPAN}(m) = \emptyset$
Let $\mathcal{R} = \bigcup\{\psi_i^l(\overline{x}_i, \overline{y}_i) \mid \phi_i(\overline{x}_i) \rightarrow \exists \overline{y}_i(\psi_i(\overline{x}_i, \overline{y}_i)) \in \Sigma_{st}\}$
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 $\mathcal{R}^{pow}$ be the set of all (renamed) multisets of atoms in $\mathcal{R}$ of size k or less
For each $\chi^l(\overline{x}_1, \overline{y}_1) \in \mathcal{R}^{pow}$
  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 $\text{EQUAL}_{h^f}(\overline{x}_1, \overline{x}_2) = \text{INTERSECT}_{h^f}(\overline{x}_1, \overline{x}_2) \cup \text{JOINS}_{h^f}(\overline{x}_1)$
      $\text{EXPAN}(m) = \text{EXPAN}(m) \cup \{\chi^l(\overline{x}_1, \overline{y}_1) \wedge \exists \overline{x}_2, \overline{y}_2 : (\psi^l(\overline{x}_2, \overline{y}_2) \bigwedge \text{EQUAL}_{h^f}(\overline{x}_1, \overline{x}_2))\}$

Given an expansion ϵ, we generate a first rewriting of ϵ, called REW-C_ϵ, by adding to ϵ the negation of each expansion ϵ' that is more compact than ϵ, with the appropriate equalities. With respect to our reference example, it can be seen that expansion ϵ_{12} and ϵ_{13} generate witness blocks that are more compact than those generated by ϵ_{11}. Called ${h^f}_{12}, {h^f}_{13}$ the compacting homomorphisms of ϵ_{11} into ϵ_{12}, ϵ_{13}, respectively, ϵ_{11} must be rewritten into a new formula $\text{REW-C}_{\epsilon_{11}}$ as follows (the actual formula is omitted since it is quite long):

$$\begin{aligned}\text{REW-C}_{\epsilon_{11}} = {} & \epsilon_{11}(\overline{x}_1, \overline{y}_1) \wedge \\ & \neg\exists \overline{x}_1', \overline{y}_1' : (\epsilon_{12}(\overline{x}_1', \overline{y}_1') \wedge \text{EQUAL}_{{h^f}_{12}}(\overline{x}_1, \overline{x}_1')) \wedge \\ & \neg\exists \overline{x}_1'', \overline{y}_1'' : (\epsilon_{13}(\overline{x}_1'', \overline{y}_1'') \wedge \text{EQUAL}_{{h^f}_{13}}(\overline{x}_1, \overline{x}_1''))\end{aligned}$$

This also means that ϵ_{21} must be rewritten into a new formula REW-C$_{\epsilon_{21}}$ as follows:

$$\text{REW-C}_{\epsilon_{21}} = \epsilon_{21}(\overline{x}_1, \overline{y}_1) \wedge \neg\exists \overline{x}'_1, \overline{y}'_1 : (\epsilon_{22}(\overline{x}'_1, \overline{y}'_1) \wedge \text{EQUAL}_{h^f{}_{22}}(\overline{x}_1, \overline{x}'_1))$$

The actual formula is reported below:

$$\begin{aligned}\text{REW-C}_{\epsilon_{21}} = {} & S^4(Y_6, x_4, x_5, Y_8) \wedge S^5(Y_6, x_5, x_4, Y_7) \wedge S^6(Y_9, Y_{10}, Y_{11}, Y_7) \wedge \\ & \neg\exists x'_4, x'_5, Y'_6, Y'_7, Y'_8 : (S^4(Y'_6, x'_4, x'_5, Y'_8) \wedge S^5(Y'_6, x'_5, x'_4, Y'_7) \wedge \\ & x_4 = x'_4 \wedge x_5 = x'_5)\end{aligned}$$

Note that REW-C$_{\epsilon_{21}}$ is never satisfied. In fact, for any set of tuples such that ϵ_{21} is true, also ϵ_{22} is true. This justifies our observation that, whenever an expansion is generated, if we verify that $\chi(\overline{x}, \overline{y})$ generates canonical blocks that are not a core, we can safely discard the expansion since it will not contribute to the core.

After this first rewriting, we look among other expansions to favor those that generate more informative witness blocks in the target, and we further rewrite REW-C$_\epsilon$ accordingly. To see an example, consider expansion ϵ_{22} above: it is easy to see that – once we have removed tuples for which there are more compact expansions – we have to ensure that expansion ϵ_{11} of the other tgd does not generate a more informative witness block in the target (we omit the rewriting here because it is quite long).

To summarize, to generate the final rewriting, we consider each expansion, ϵ of a tgd m; then: (i) we first rewrite ϵ into a new formula REW-C$_\epsilon$ by adding the negation of all expansions ϵ_i such that ϵ_i is more compact than ϵ; (ii) we further rewrite REW-C$_\epsilon$ into a new formula REW-I$_\epsilon$ by adding the negation of REW-C$_{\epsilon_j}$, for all expansions ϵ_j such that ϵ_j is more informative than ϵ. Intuitively, the union of all REW-I$_\epsilon$ for a tgd m generates all maximal witness blocks for that tgd.

Definition 11 *[**Rewriting Expansions**:* REW-C *and* REW-I*] Given an expansion* $\epsilon = \chi^l(\overline{x}_1, \overline{y}_1) \wedge \exists \overline{x}_2, \overline{y}_2 : (\psi^l(\overline{x}_2, \overline{y}_2) \bigwedge \text{EQUAL}_{h^f{}_\epsilon}(\overline{x}_1, \overline{x}_2))$*, the formula* REW-C$_\epsilon$ *is obtained as follows:*

- *initialize* REW-C$_\epsilon = \epsilon$*;*
- *for any 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* EXPAN$(\mathcal{M})$ *such that* ϵ' *is more compact than* ϵ*, call* $h^f{}_c$ *the compacting homomorphism of* $\chi^l(\overline{x}_1, \overline{y}_1)$ *into* $\chi^{l'}(\overline{x}'_1, \overline{y}'_1)$*; then, add to* REW-C$_\epsilon$ *a formula*

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

The formula REW-I$_\epsilon$ *is obtained as follows:*

- *initialize* REW-I$_\epsilon$ = REW-C$_\epsilon$*;*
- *for any 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* EXPAN($\mathcal{M}$) *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* $\chi^{l'}(\overline{x}'_1, \overline{y}'_1)$*; then, add to* REW-I$_\epsilon$ *a formula*

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

Isomorphisms

We can now get back to our original goal, that is, find the set of witness blocks that belong to the core. Rewriting expansions in order to capture maximal witness blocks is a promising step forward. Unfortunately, it is not sufficient to generate the core. In fact, it is still possible that expansions select witness blocks that 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 multiple 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 [78], this may happen if an expansion has non-trivial automorphisms.

Consider for example expansion ϵ_{22} above.

$$\epsilon_{22}.\ S^4(Y_6, x_4, x_5, Y_8) \wedge S^5(Y_6, x_5, x_4, Y_7)$$

it can be seen that $\chi(\overline{x}_1, \overline{y}_1)$ has a non-trivial automorphism that maps x_4 into x_5 and viceversa.

Therefore, the expansion will select pairs of isomorphic witness blocks in J of the form $S^4(N_0, a, b, N_1), S^5(N_0, b, a, N_2)$ and $S^4(N_3, b, a, N_4), S^5(N_3, a, b, N_5)$. This problem does not arise if J is *isomorphism–free*, i.e., it does not contain witness blocks that are isomorphic to each other.

Definition 12 ***[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.

We are now ready to introduce our main result about expansions.

Theorem 3.4.1 *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 following set:*

$$\mathcal{E}^J_{\text{REW-I}} = \bigcup_{\epsilon \in \text{EXPAN}(\mathcal{M})} \{a(\chi^l(\overline{x}_1, \overline{y}_1)) \mid a \text{ s.t. } J \models a(\text{REW-I}_\epsilon(\overline{x}_1, \overline{y}_1))\}$$

then it is the case that:

$$\mathcal{E}^J_{\text{REW-I}} = \text{REDUCE}(\text{MAX-INFORMATIVE}(\text{MAX-COMPACT}(\mathcal{W}^{<I,J>})))$$

The full proof is reported in the Appendix. From Theorem 3.3.5 it follows that $\mathcal{E}^J_{\text{REW-I}}$ is exactly the core of J.

3.5 The Rewriting Algorithm

Theorems 3.4.1 and 3.3.5 suggest a natural rewriting strategy for the original scenario $\mathcal{M}$. In fact, once an isomorphism-free solution J has been generated for $\mathcal{M}$ over a source instance I (for example by running the standard chase of the original tgds), it is possible to compute the core of J by executing the following set of full tgds, one for each expansion $\epsilon \in \text{EXPAN}(\mathcal{M})$:

$$\forall \overline{x}_1, \overline{y}_1 : \text{REW-I}_\epsilon(\overline{x}_1, \overline{y}_1) \rightarrow \chi(\overline{x}_1, \overline{y}_1) \tag{3.2}$$

The tgds are based on the idea of using (rewritten) expansions to select maximal witness blocks inside J, and copy them to the core. No new null value needs to be invented in this process. This is, in fact, the strategy proposed in [59]. Notice that this strategy is a two-step strategy, 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 [78] it was shown that this two-step approach is not strictly 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 core computation strategy in order to compute the core within a single exchange.

Source Rewriting

An expansion is a formula over the target schema. However, in this section we show that it is rather straightforward to rewrite it in terms of source relations. We call this the *source rewriting* of the expansion. In order to do this, we first introduce the notion of *premise* of an atom.

Definition 13 ***[Premise of an Atom and of a Formula]*** *Given a tgd $\phi(\overline{x}) \rightarrow \exists \overline{y}(\psi(\overline{x}, \overline{y}))$, and an atom $R^l(\overline{x}_i, \overline{y}_i) \in \psi^l(\overline{x}, \overline{y})$, its* premise $\text{PREM}_{R^l(\overline{x}_i, \overline{y}_i)}$ *is the formula $\phi(\overline{x})$. Given a conjunctive formula, $\chi^l(\overline{x}_1, \overline{y}_1)$, its* premise *is the formula:*

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

Definition 14 ***[Source Rewriting]*** *Given a tgd $\phi(\overline{x}_2) \rightarrow \exists \overline{y}_2(\psi^l(\overline{x}_2, \overline{y}_2))$ and an expansion $\epsilon = \chi^l(\overline{x}_1, \overline{y}_1) \wedge \exists \overline{x}_2, \overline{y}_2 : (\psi^l(\overline{x}_2, \overline{y}_2) \bigwedge \text{EQUAL}_{h^{f_\epsilon}}(\overline{x}_1, \overline{x}_2))$ of m, its* source rewriting, REW-S_ϵ, *is the following formula:*

$$\text{REW-S}_\epsilon = \text{PREM}_{\chi^l(\overline{x}_1, \overline{y}_1)} \wedge \exists \overline{x}_2 : (\phi(\overline{x}_2) \bigwedge \text{EQUAL}_{h^{f_\epsilon}}(\overline{x}_1, \overline{x}_2))$$

Example 3.5.1 *Consider the scenario in Example 3.3.1. Tgd m_1 has the following expansions:*

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

Their source rewritings are as follows:

$$\begin{array}{lcl}\text{REW-S}_{\epsilon_{11}} & = & A(x_1, x_2) \\ \text{REW-S}_{\epsilon_{12}} & = & B(x_3, x_4) \wedge C(x_5, x_6) \wedge \exists x_1, x_2 : \\ & & (A(x_1, x_2) \wedge x_3 = x_1 \wedge x_6 = x_2 \wedge x_4 = x_5)\end{array}$$

Similarly for tgd m_4:

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

Example 3.5.2 *Consider the scenario in Example 3.3.2:*

$$\begin{array}{l} m_1.\ R(x_0, x_1, x_2) \rightarrow \exists Y_0, Y_1, Y_2, Y_3, Y_4, Y_5 : S^1(Y_0, x_0, Y_2, Y_3), \\ \qquad\qquad\qquad\qquad\qquad\qquad S^2(Y_0, x_1, x_1, Y_1), S^3(Y_4, x_2, Y_5, Y_1) \\ m_2.\ R(x_3, x_4, x_5) \rightarrow \exists Y_6, Y_7, Y_8, Y_9, Y_{10}, Y_{11} : S^4(Y_6, x_4, x_5, Y_8), \\ \qquad\qquad\qquad\qquad\qquad\qquad S^5(Y_6, x_5, x_4, Y_7), S^6(Y_9, Y_{10}, Y_{11}, Y_7)\end{array}$$

Here are some expansions and their source rewritings:

$$\begin{array}{lcl}\epsilon_{11} & = & S^1(Y_0, x_0, Y_2, Y_3) \wedge S^2(Y_0, x_1, x_1, Y_1) \wedge S^3(Y_4, x_2, Y_5, Y_1) \\ \epsilon_{12} & = & S^1(Y_0, x_0, Y_2, Y_3) \wedge S^2(Y_0, x_1, x_1, Y_1) \wedge \\ & & \quad \exists x_{2b}, Y_{4b}, Y_{5b} : (S^3(Y_{4b}, x_{2b}, Y_{5b}, Y_1) \wedge x_1 = x_{2b}) \\ \epsilon_{13} & = & S^5(Y_6, x_5, x_4, Y_7) \wedge \exists x_{0b}, x_{1b}, x_{2b}, Y_{0b}, Y_{1b}, Y_{2b}, Y_{3b}, Y_{4b}, Y_{5b} : \\ & & \quad (S^1(Y_{0b}, x_{0b}, Y_{2b}, Y_{3b}) \wedge S^2(Y_{0b}, x_{1b}, x_{1b}, Y_{1b}) \wedge \\ & & \quad S^3(Y_{4b}, x_{2b}, Y_{5b}, Y_{1b}) \wedge x_5 = x_{0b} \wedge x_5 = x_{1b} \wedge \\ & & \quad x_4 = x_{1b} \wedge x_5 = x_{2b}) \\ \text{REW-S}_{\epsilon_{11}} & = & R(x_0, x_1, x_2) \\ \text{REW-S}_{\epsilon_{12}} & = & R(x_0, x_1, x_2) \wedge x_1 = x_2 \\ \text{REW-S}_{\epsilon_{13}} & = & R(x_3, x_4, x_5) \wedge \exists x_{1b}, x_{2b}, x_{3b} : (R(x_{0b}, x_{1b}, x_{2b}) \wedge \\ & & \quad x_5 = x_{0b} \wedge x_5 = x_{1b} \wedge x_4 = x_{1b} \wedge x_5 = x_{2b})\end{array}$$

Given an expansion ϵ, we can now introduce the formulas REW-SC and REW-SI, that are analogous to REW-C and REW-I, but are based on the source rewriting of the expansion.

Definition 15 ***[Rewriting Source Expansions:** REW-SC and REW-SI] Given an expansion ϵ, the formula* REW-SC_ϵ *is obtained as follows:*

- *initialize* $\text{REW-SC}_\epsilon = \text{REW-S}_\epsilon$*;*
- *for any expansion ϵ' in* $\text{EXPAN}(\mathcal{M})$ *such that ϵ' is more compact than ϵ, call ${h^f}_c$ the compacting homomorphism of $\chi^l(\overline{x}_1, \overline{y}_1)$ into $\chi^{l'}(\overline{x}'_1, \overline{y}'_1)$; then, add to* REW-SC_ϵ *a formula*

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

The formula REW-SI_ϵ *is obtained as follows:*

- *initialize* $\text{REW-SI}_\epsilon = \text{REW-SC}_\epsilon$*;*
- *for any expansion ϵ' in* $\text{EXPAN}(\mathcal{M})$ *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 $\chi^{l'}(\overline{x}'_1, \overline{y}'_1)$; then, add to* REW-SI_ϵ *a formula:*

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

Consider, again, Example 3.5.2. Recall that $\text{REW-C}_{\epsilon_{11}}$ is the following formula:

$$\begin{aligned}\text{REW-C}_{\epsilon_{11}} = {} & \epsilon_{11}(\overline{x}_1, \overline{y}_1)\wedge \\ & \neg\exists\overline{x}'_1, \overline{y}'_1 : (\epsilon_{12}(\overline{x}'_1, \overline{y}'_1) \wedge \text{EQUAL}_{{h^f}_{12}}(\overline{x}_1, \overline{x}'_1))\wedge \\ & \neg\exists\overline{x}''_1, \overline{y}''_1 : (\epsilon_{13}(\overline{x}''_1, \overline{y}''_1) \wedge \text{EQUAL}_{{h^f}_{13}}(\overline{x}_1, \overline{x}''_1))\end{aligned}$$

This generates the following source rewriting:

$$\begin{aligned}\text{REW-SC}_{\epsilon_{11}} = {} & R(x_0, x_1, x_2)\wedge \\ & \neg\exists x'_0, x'_1, x'_2 : (R(x'_0, x'_1, x'_2) \wedge x'_1 = x'_2\wedge \\ & \qquad x_0 = x'_0 \wedge x_1 = x'_1 \wedge x_2 = x'_1)\wedge \\ & \neg\exists x''_3, x''_4, x''_5 : (R(x''_3, x''_4, x''_5) \wedge \exists x_{1b}, x_{2b}, x_{3b} : (R(x_{0b}, x_{1b}, x_{2b})\wedge \\ & \qquad x''_5 = x_{0b} \wedge x''_5 = x_{1b} \wedge x''_4 = x_{1b} \wedge x''_5 = x_{2b})\wedge \\ & \qquad x_0 = x''_5 \wedge x_1 = x''_5 \wedge x_1 = x''_4 \wedge x_2 = x''_5)\end{aligned}$$

We are now almost ready to introduce our final rewriting. Before doing that, we need to discuss our skolemization strategy.

There is, in fact, a significant difference between the two–step strategy based on the full tgds in Formula 3.2, and the single exchange we are trying to develop here. While the two-step approach selects witness blocks that have already been generated in the canonical solution, here we need to generate them. To do this, we use REW-SI$_\epsilon$ to select all vectors of universally quantified variables that are actually needed to generate the witness blocks in the core. Then, we need Skolem terms in order to properly invent 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 16 ***[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 $f_{\varphi,y_i}(\overline{x})$.*

In the following, we assume that a Skolemization strategy has been fixed. Based on that, we introduce our rewriting, based on the notion of *expansion rules*:

Definition 17 ***[Expansion Rules]*** *Fixed a skolemization strategy,* SKOL*, and an expansion $\epsilon = \chi^l(\overline{x}_1, \overline{y}_1) \wedge \exists \overline{x}_2, \overline{y}_2 : (\psi^l(\overline{x}_2, \overline{y}_2) \bigwedge \text{EQUAL}_{hf_\epsilon}(\overline{x}_1, \overline{x}_2))$ its* expansion rule*, r_ϵ, is the following formula:*

$$r_\epsilon. \forall \overline{x}_1 : \text{REW-SI}_\epsilon(\overline{x}_1) \rightarrow \text{SKOL}(\chi(\overline{x}_1, \overline{y}_1))$$

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

$$\Sigma^{FO,exp}_{\mathcal{M},\text{SKOL}} = \{r_\epsilon \mid \epsilon \in \text{EXPAN}(M)\}$$

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

Normalization

There is a final issue we need to discuss. As we discussed in Section 3.3, 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. Therefore, once the expansion rules have been generated, we need to normalize them. By doing this, however, we may end up with different, unnecessary rules, that generate portions of witness blocks.

Example 3.5.3 *Consider the scenario in Example 3.3.3. There are three relevant expansions that generate the core:*

$$\begin{aligned}
\epsilon_a &= S^1(x_3,x_0,x_0,Y_0,x_1) \wedge S^2(Y_1,x_0,x_0,Y_0,x_0) \wedge \exists x_2', Y_2', Y_3', Y_4' : \\
&\quad (S^3(Y_1,x_2',Y_2',Y_3',Y_4') \wedge x_0 = x_2') \\
\epsilon_b &= S^1(x_3,x_0,x_0,Y_0,x_1) \wedge \exists x_2', Y_1', Y_2', Y_3', Y_4' : \\
&\quad (S^2(Y_1',x_0,x_0,Y_0,x_0) \wedge S^3(Y_1',x_2',Y_2',Y_3',Y_4') \wedge x_1 = x_0 \wedge x_0 = x_2') \\
\epsilon_c &= S^1(x_3,x_0,x_0,Y_0,x_1) \wedge S^1(x_3,x_{0a},x_{0a},W_0,x_{1a}) \wedge \exists x_2', Y_1', Y_2', Y_3', Y_4' : \\
&\quad (S^2(Y_1',x_0,x_0,Y_0,x_0) \wedge S^3(Y_1',x_2',Y_2',Y_3',Y_4') \wedge x_1 = x_0 \wedge x_{0a} = x_2')
\end{aligned}$$

Of these expansions, ϵ_c is not normalized. Therefore, when we generate the final expansion rules, we would end up with something like this:

$$\begin{aligned}
&r_a : \text{REW-SI}_{\epsilon_a}(\overline{x}) \rightarrow \text{SKOL}(S^1(x_3,x_0,x_0,Y_0,x_1) \wedge S^2(Y_1,x_0,x_0,Y_0,x_0)) \\
&r_b : \text{REW-SI}_{\epsilon_b}(\overline{x}) \rightarrow \text{SKOL}(S^1(x_3,x_0,x_0,Y_0,x_1)) \\
&r_c : \text{REW-SI}_{\epsilon_c}(\overline{x}) \rightarrow \text{SKOL}(S^1(x_3,x_0,x_0,Y_0,x_1) \wedge S^1(x_3,x_{0a},x_{0a},W_0,x_{1a}))
\end{aligned}$$

If we normalize the last rule, we end up with:

$$\begin{aligned}
&r_a : \text{REW-SI}_{\epsilon_a}(\overline{x}) \rightarrow \text{SKOL}(S^1(x_3,x_0,x_0,Y_0,x_1) \wedge S^2(Y_1,x_0,x_0,Y_0,x_0)) \\
&r_b : \text{REW-SI}_{\epsilon_b}(\overline{x}) \rightarrow \text{SKOL}(S^1(x_3,x_0,x_0,Y_0,x_1)) \\
&r_{c1} : \text{REW-SI}_{\epsilon_{c1}}(\overline{x}) \rightarrow \text{SKOL}(S^1(x_3,x_0,x_0,Y_0,x_1)) \\
&r_{c2} : \text{REW-SI}_{\epsilon_{c2}}(\overline{x}) \rightarrow \text{SKOL}(S^1(x_3,x_{0a},x_{0a},W_0,x_{1a}))
\end{aligned}$$

But it is possible to see that we need to generate an atom according to r_b, r_{c1}, r_{c2} 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 18 ***[Rewriting Expansion Rules:*** REW-EI*] Given an expansion rule, $r : \phi(\overline{x}_1) \rightarrow \psi(\overline{x}_1)$, the formula* REW-EI$_r$ *is obtained as follows:*

- *initialize* REW-EI$_r = r$*;*
- *for any expansion rule* $r' : \phi'(\overline{x}'_1) \rightarrow \psi'(\overline{x}'_1)$ *in* $\Sigma^{FO,exp}_{\mathcal{M},\text{SKOL}}$ *such that* $\psi'(\overline{x}'_1)$ *is more informative than* $\psi(\overline{x}_1)$ *according to homomorphism* $h^f{}_i$*, add to* REW-EI$_r$ *a formula*

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

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

$$\Sigma^{FO,core}_{\mathcal{M},\text{SKOL}} = \{\text{REW-EI}_r \mid r \in \text{NORMALIZE}(\Sigma^{FO,exp}_{\mathcal{M},\text{SKOL}})\}$$

Skolemization Strategy

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 rule m, and taking as arguments all universal variables occurring in the conclusion. However, there are alternatives to this scheme. In fact, in order for our rewriting to correctly compute core solutions, we need to properly handle isomorphic 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. Our solution to the problem is to use interpreted Skolem functions, 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 $a(\text{SKOL}(\varphi(\overline{x}, \overline{y})))$ 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 $a(\text{SKOL}_{std}(\varphi(\overline{x}, \overline{y})))$ the *standard instance* of the formula.

Definition 19 ***[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* $a(\text{SKOL}(\varphi(\overline{x}, \overline{y})))$ *is isomorphic to* $a(\text{SKOL}_{std}(\varphi(\overline{x}, \overline{y})))$*;*

- *given two formulas,* $\varphi(\overline{x},\overline{y})$*,* $\varphi'(\overline{x}',\overline{y}')$*, and two assignments* a, a'*, if the standard instances* $a(\text{SKOL}_{std}(\varphi(\overline{x},\overline{y})))$*,* $a'(\text{SKOL}_{std}(\varphi'(\overline{x}',\overline{y}')))$ *are isomorphic, then* $a(\text{SKOL}(\varphi(\overline{x},\overline{y})) = a'(\text{SKOL}(\varphi'(\overline{x}',\overline{y}'))$*.*

There are several ways in which it is possible to build an isomorphism–invariant strategy. One solution is to consider the dual Gaifman graph of the standard instance, and design a Skolem function that returns strings based on an encoding of the graph. To implement this behavior, we embed in the Skolem string a full description of the Gaifman graph, in terms of constants.

Consider for example the following formula: $S(x_0, Y_0), T(x_1, Y_0, Y_1), W(Y_1)$. The Skolem functions for Y_0 and Y_1 will have three arguments: (i) a description of the graph nodes, in terms of constants; (ii) an encoding of the graph edges; (iii) a reference to the specific variable for which the function is used. The actual functions would be as follows:

$$f_{sk}(\{S(A:x_0), T(A:x_1), W()\}, \{S.B = T.B, T.C = W.A\}, S.B = T.B)$$
$$f_{sk}(\{S(A:x_0), T(A:x_1), W()\}, \{S.B = T.B, T.C = W.A\}, T.C = W.A)$$

An important point here is that set elements 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, like, for example, in $S(x_0, Y_0), S(x_1, Y_0)$, then functions of this form are allowed: $f_{sk}(\{S(A:x_0), S(A:x_1)\}, \{S.B = S.B\})$. To properly handle the non-trivial automorphism of the formula, it can be seen how the description of nodes must be reordered at execution time, based on the actual assignment of values to variables.

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 [78].

Computing the Core

We are now ready to introduce our main result.

Theorem 3.5.4 *Given a* $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st})$ *and an isomorphism-invariant skolemization strategy,* SKOL*,* $\Sigma^{FO,core}_{\mathcal{M},\text{SKOL}}$ *is a core schema mapping for* $\mathcal{M}$*.*

The full proof is in the Appendix. Based on Theorem 3.5.4, we can provide a complete description of our approach to the computation of core solutions. Given a scenario $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st})$, the rewriting algorithm works as follows:

1. it generates EXPAN($\mathcal{M}$) as the union of all expansions EXPAN(m), for each tgd $m \in \Sigma_{st}$;
2. for each expansion $\epsilon \in$ EXPAN($\mathcal{M}$):
   a) it generates its source rewriting REW-S$_\epsilon$;
   b) then, it looks for compacting homomorphisms among expansions, and generates REW-SC$_\epsilon$;
   c) based on that, it looks for proper homomorphisms among expansions, and generates REW-SI$_\epsilon$;
3. it picks an isomorphism-invariant skolemization strategy (for example, the one based on the encoding of the dual Gaifman graph described above), and generates the set of expansion rules $\Sigma^{FO,exp}_{\mathcal{M},\text{SKOL}}$;
4. it normalizes the rules, and further rewrites them to generate the core schema mapping, $\Sigma^{FO,core}_{\mathcal{M},\text{SKOL}}$;
5. finally, from this set of dependencies, it generates a runtime SQL script that can be executed on source instances to generate core universal solutions.

3.6 Complexity and Approximations

A few comments are worth making here on the complexity of the rewriting algorithm. Recall that our goal is to execute the rewritten tgds 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.

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. 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 relational atom may appear more than once in the conclusions of tgds in Σ_{st}. The scenarios in Examples 3.3.2, 3.3.3, 3.3.4 have self-joins in tgd conclusions. The scenario in Example 3.3.1 does not have self-joins.

In [59], 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 3.5 is strongly influenced on 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 $\textsc{expan}(\mathcal{M})$ 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 3.6.1 *Given a $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st})$, 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 belongs 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 3.6.1, 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 3.6.1, 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: (a) for each tgd, generate only fixed-size expansions; (b) then, concentrate on the base expansion alone; (c) find all compacting and proper homomorphisms into all other expansions, in order to remove unnecessary witness blocks from the result; (d) finally, generate one FO-rule for each base expansion in order to produce the result, and disregard rules for other expansions. This significantly reduces the number of final tgds.

Based on this 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 3.3.1. 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.

Complexity

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*[51] 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 asa 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. Therefore, the number of joins, intersections and differences in the final SQL script

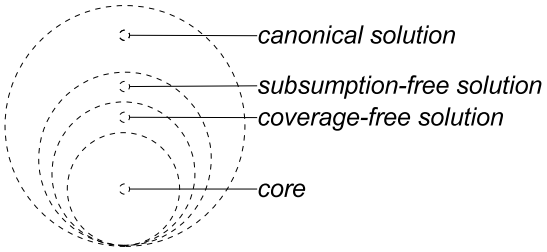

Figure 3.6: Containment of Solutions

may be very high. In fact, it is not difficult to design synthetic scenarios that actually trigger the exponential explosion of rewritings.

However, 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.

It is also worth noting that, when the complexity of the rewriting becomes high, our algorithm allows to produce several acceptable approximations of the core. In fact, the algorithm is modular in nature; when the core computation requires very high computing times and does not scale to large databases, the mapping designer may decide to discard the "full" rewriting, and select a "reduced" rewriting (i.e., a rewriting wrt to a subset of homomorphisms) to generate an *approximation* of the core more efficiently. This can be done by rewriting tgds with respect to subsumptions only or to subsumptions and coverages, as shown in Figure 3.6.

3.7 Experimental Results

The algorithms introduced in the paper have been implemented in a working prototype 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 larger number of relations and join constraints. 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.

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 eight 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 [43], sj_2 from [81]), and from variants of the basic scenarios in STBenchmark [8]. 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 3.3.2, 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 [74] of the core computation algorithm developed in [51], 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.

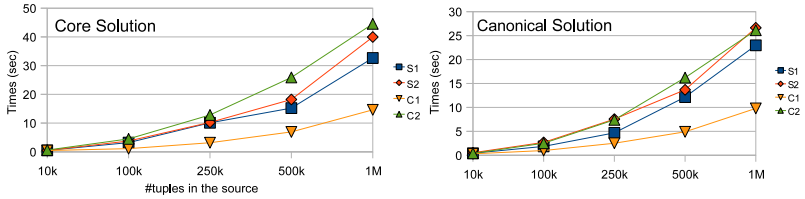

Figure 3.7: SQL Scripts: Execution Times for the First Group

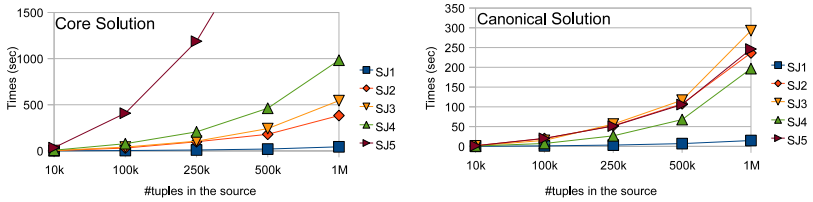

Figure 3.8: SQL Scripts: Execution Times for the Second Group

Since we were interested in comparing computing times of the scripts for core solution to those of scripts for canonical solutions, we ran the two sets of scripts over the same scenarios and reported the results in Figure 3.7 and Figure 3.8. Figure 3.7 shows executing times 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 3.8 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 In data exchange core solutions are preferable with respect to canonical solutions because they preserve the same information but in a more compact databases. To exploit the difference between the two solutions we used different databases with fixed size (10k tuples) and increasing redundancy. We dropped sj_5 from this comparison. For each of the remaining eight scenarios, we generated five synthetic source instances 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 3.9 shows the percent reduction in the

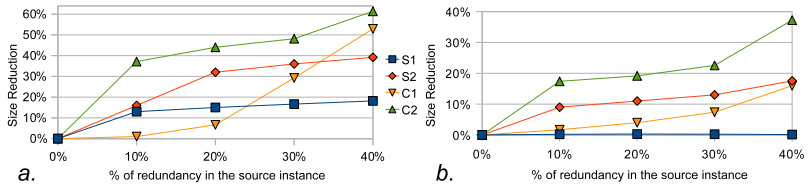

Figure 3.9: Core vs Canonical: Size Reduction in Solutions

output size for core solutions compared to canonical solutions. As output size, we measured the number of tuples in the solutions. Figure 3.9.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 3.9.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.

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 3.10 summarizes the results. In the graph, we

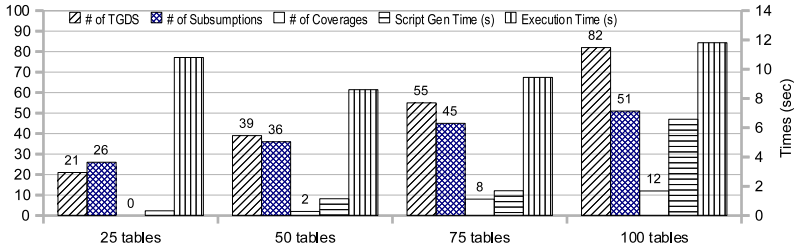

Figure 3.10: Algorithm scalability with large synthetic scenarios

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.

Chapter 4

Schema Mapping Tools

Manually writing transformation rules under the form of logical formulas, mainly for a complex scenario, can be very difficult also for expert users. It is much more simple and natural to provide a minimal and high-level specification of the mapping. For these reasons, a schema mapping system takes as input an abstract specification of the mapping under the form of *value correspondences* among schema elements and generates the tgds and the executable transformations to run them.

Each of these correspondences states that an attribute of the target is semantically related to one (or more) attribute in the source, and it is usually drawn as a *line* from the source attribute to the corresponding target attribute.

Clio [70] was the first system to introduce value correspondences and to implement a sophisticated mapping algorithm to generate source-to-target transformations. The mapping generation algorithm captures all semantical relationships embedded in the source and the target schemas, and is guaranteed to produce legal instances of the target with respect to constraints.

In this Chapter we give a quick overview of how the input tgds are generated by the system. Note that, as an alternative, +Spicy allows to load a set of pre-defined tgds provided as logical formulas encoded in a fixed textual format.

In the following, we use the *nested relational data model* introduced in Chapter 2 to represent data sources, and we adopt a XQuery-like syntax to express transformation rules as in [70].

4.1 Expressive Power

It can be seen that the rewriting algorithm can be applied to any set of tgds, not necessarily generated by the mapping system. To do this, one of our goals was to extend the expressive power of the mapping system with respect to previous ones.

Suppose we are given the following set of pre-defined tgds that refer to a variant of the self-join example in STBenchmark [8]. The target schema contains a single relation *Gene* with attributes *name*, *type* and *protein*, which holds together primary genes and secondary genes, called "synonyms". A primary gene and its synonyms share the same protein. In the source, we have genes organized in separate tables *PrimaryGene* and *Synonym*, connected through a key-foreign key constraint. In addition, we have a *Protein* table, from which we want to copy only tuples about genes coming from the EMBL database. A key feature of this example is the self-join of table *Gene* in the target on the *protein* attribute. The tgds, expressed in logical form, are the following:

$$
\begin{aligned}
&m_1.\ \mathit{Protein}(p, g, \text{`EBML'}) \rightarrow \mathit{Gene}(g, p, \text{`primary'})\\
&m_2.\ \mathit{PrimaryGene}(i, n, p) \rightarrow \mathit{Gene}(n, p, \text{`primary'})\\
&m_3.\ \mathit{Synonym}(n, i) \wedge \mathit{PrimaryGene}(i, n', p)\\
&\qquad \rightarrow \mathit{Gene}(n, p, \text{`synonym'}), \mathit{Gene}(n', p, \text{`primary'})
\end{aligned}
$$

Our goal is to generate a mapping scenario for these tgds, and then rewrite them in order to generate core solutions. In this case, +Spicy proposes to the user the scenario in Figure 4.1.

To handle arbitrary tgds of this form, we had to enrich the set of primitives that can be used to specify a mapping scenario.

We extend these inputs in several ways:

(*i*) we introduce the possibility of duplicating sets in the source and in the target; to handle tgd m_3 above, we duplicate the *Gene* table in the target; each duplication of a set R corresponds to adding to the data source a new set named $R(i)$, for some i, that is an exact *copy* of R;

(*ii*) we give users full control over joins in the two data sources, in addition to those corresponding to foreign key constraints; using this feature, users can specify arbitrary join paths, like the self-join on the *protein* attribute in m_3;

(*iii*) finally, we allow users to express selection conditions on sets, like *source = 'EMBL'* on the protein table in m_1.

This richer set of primitives poses some challenges with respect to the rewriting algorithm. In fact, duplications in the target correspond to different ways

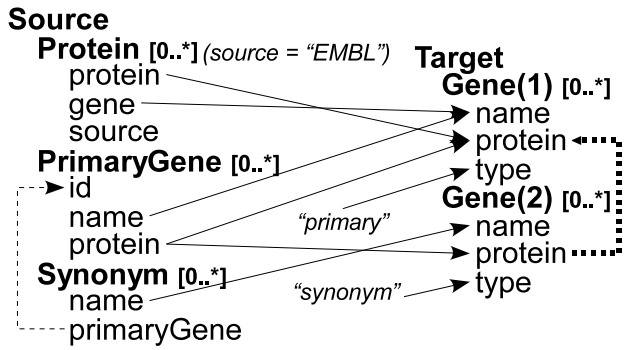

Figure 4.1: Inverse of Self Joins

of contributing tuples to the same set. This makes the search form homomorphisms more delicate, since there exist tgds that write more than one tuple at a time in the same target table, and therefore redundancy can be generated not only across different tgds, but also by firing a single tgd.

4.2 Clio Mapping Discovery Algorithm

The mapping discovery algorithm employed in our system is a generalization of the one developed in Clio. In this Section, we briefly review the main steps of this algorithm [70].

Given a source and a target schema, the algorithm produces a number of executable queries. Each query is the union of several mappings that represent source-to-target tuple generating dependencies. In order to identify such dependencies, several preliminary steps are necessary.

As a first step, we need to identify the *logical relations* both in the source and target schemas. Logical relations are maximal tableaux [6]. In order to generate them, the algorithm finds the so called *primary paths* in each schema. These are essentially linear tableaux, and are obtained by the enumeration of all paths from the root to any intermediate node of set type in such a schema. With respect to the schemas in Figure 4.2, the primary paths generated for the source and target schemas respectively, are the following (with an abuse of notation we use "select *" to refer to all attributes that are directly reachable from a set node in the tableaux):

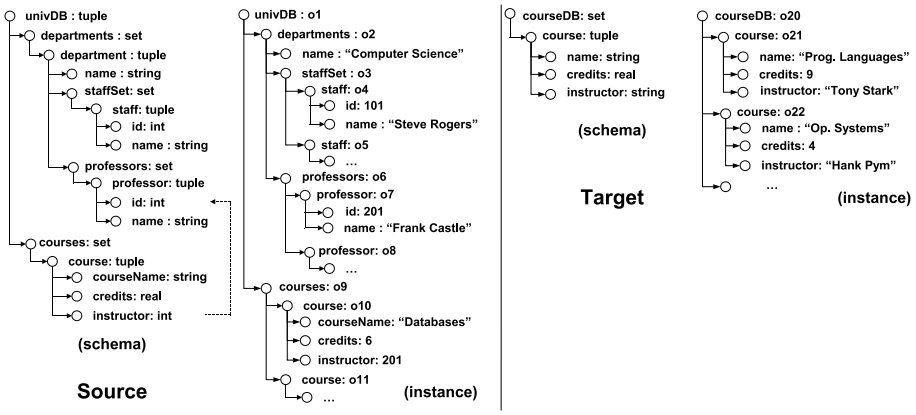

Figure 4.2: A sample mapping task

```
select * from d in univDB.departments
select * from d in univDB.departments, s in d.department.staffSet
select * from d in univDB.departments, p in d.department.professors
select * from c in univDB.courses

select * from p in courseDB
```

Logical relations are obtained by chasing constraints against primary paths. In our example, a single constraint, from univDB.courses.course.instructor to univDB.departments.department.professors.professor.id, must be considered. Thus, we obtain the following logical relations:

```
select * from d in univDB.departments
select * from d in univDB.departments, s in d.department.staffSet
select * from d in univDB.departments, p in d.department.professors
select * from c in univDB.courses,
         d in univDB.departments, p in d.department.professors
where    c.course.instructor = p.professor.id

select * from p in courseDB
```

During the third step, inter-schema correspondences are processed to yield a set of mappings. To generate mappings, all possible pairs made of a source

logical relation and a target logical relation are considered; a pair generates a mapping if it covers one or more value correspondences. As an example, assume the following correspondences have been provided to the algorithm for the mapping task in Figure 4.2 (similarities are omitted since they are not strictly relevant to the mapping generation algorithm):

```
courseDB.course.name       → univDB.courses.course.courseName
courseDB.course.credits    → univDB.courses.course.credits
courseDB.course.instructor → univDB....professor.name
```

The algorithm will generate the following tgds, written in XQuery-like form:

```
for    d in univDB.departments, p in d.professors,
       c in univDB.courses
where  c.course.instructor = p.professor.id
exists c' in courseDB
where  c'.course.name = c.course.courseName and
       c'.course.credits = c.course.credits and
       c'.professor.name = c'.course.instructor

for    d in univDB.departments, p in d.professors,
exists c' in courseDB
where  p.professor.name = c'.course.instructor
```

Such tgds represent different ways to cover the correspondences and generate tuples in the target. The final transformation query is obtained by taking the union of the two. Once transformations have been generated, it is then rather straightforward to translate this abstract syntax into a concrete one, for example XQuery.

4.3 TGD Generation Algorithm

The tgd generation algorithm we describe here is a generalization of the *basic mapping* generation algorithm introduced in [70]. The input to the algorithm is a *mapping scenario*, i.e., an abstract specification of the mapping between source and target. In order to achieve a greater expressive power, as discussed in Section 4.1, we enrich the primitives for specifying scenarios. More specifically, given a source schema **S** and a target **T**, a *mapping scenario* is specified as follows:

(*i*) two (possibly empty) sets of *duplications* of the sets in **S** and in **T**; each duplication of a set R corresponds to adding to the data source a new set named R^i, for some i, that is an exact *copy* of R;

(*ii*) two (possibly empty) sets of *join constraints* over **S** and over **T**; each join constraint specifies that the system needs to chase a join between two sets; foreign key constraints also generate join constraints;

(*iii*) a set of *value correspondences*, or *lines*; for the sake of simplicity in this paper we concentrate on $1:1$ correspondences of the form $A_S \rightarrow A_T$.

In its general form, a correspondence maps n source attributes into a target attribute via a *transformation function*; moreover, it can have an attached filter that states under which conditions the correspondence must be applied; our system handles the most general form of correspondences; it also handles *constant* lines. It is possible to extend the algorithms presented in this thesis to handle the most general form of correspondence; this would be important in order to incorporate *conditional tgds* [26]; while the extension is rather straightforward for constants appearing in tgd premises, it is more elaborate for constants in tgd conclusions, and is therefore left to future work.

The tgd generation algorithm is made of several steps. As a first step, duplications are processed; for each duplication of a set R in the source (target, respectively), a new set R^i is added to the source (target, respectively). Then, the algorithm finds all sets in the source and in the target schema; this corresponds, in the terminology of [70], to finding *primary paths*.

The next step is concerned with generating views over the source and the target. Views are a generalization of *logical relations* in [70] and are the building blocks for tgds. Each view is an algebraic expression over sets in the data source. Let us now restrict our attention to the source (views in the target are generated in a similar way).

The set of views, $\mathcal{V}_{init}$, is initialized as follows: for each set R a view R is generated. This initial set of views is then processed in order to chase join constraints and assemble complex views; intuitively, chasing a join constraint from set R to set R' means to build a view that corresponds to the join of R and R'. As such, each join constraint can be seen as an operator that takes a set of existing views and transforms them into a new set, possibly adding new views or changing the input ones. Join constraints can be *mandatory* or *non mandatory*; intuitively, a mandatory join constraint states that two sets must either appear together in a view, or not appear at all.

Once views have been generated for the source and the target schema, it is possible to produce a number of candidate tgds. We say that a source view v *covers* a value correspondence $A_S \rightarrow A_T$ if A_S is an attribute of a set

that appears in v; similarly for target views. We generate a candidate tgd for each pair made of a source view and a target view that covers at least one correspondence. The source view generates the left-hand side of the tgd, the target view the right-hand side; lines are used to generate universally quantified variables in the tgd; for each attribute in the target view that is not covered by a line, we add an existentially quantified variable.

4.4 The +Spicy System

The +Spicy system has been developed in Java using the NetBeans Platform as a basis for the graphical user interface. A snapshot is shown in Figure 4.3. The system architecture will be described in Chapter 5. The system supports

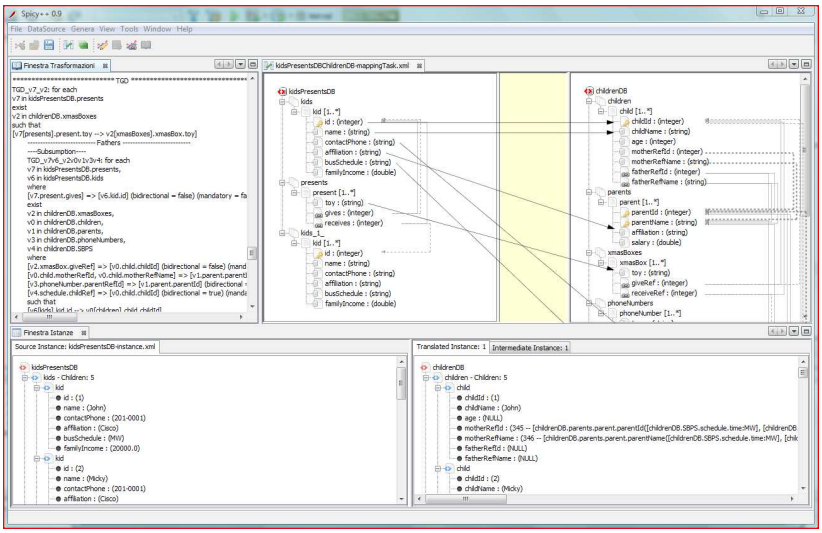

Figure 4.3: A snapshot of the system

various usage scenarios. The typical one is that in which a user provides to the system a mapping specification using the GUI; in doing this, besides specifying the source and target schema, users can rely on the primitives offered by the system, namely: (i) a rich set of correspondences that include traditional 1:1 correspondences but also n:1 value correspondences with complex transforma-

tion functions, constant correspondences, and correspondences with filters; (*ii*) the possibility of duplicating sets in the two schemas; (*iii*) the possibility to define arbitrary join-conditions in the sources; (*iv*) the possibility of specifying selection conditions on sets in the source. The mapping specification is handled by the mapping generation module, which generates the tgds.

As an alternative, a simple parser is available to load a set of pre-defined tgds. The parser will generate a scenario from the tgds, and show it to the user so that s/he can visually inspect and possibly modify it.

At this point, the user has a set of tgds, either generated internally or pre-defined and loaded by the parser. Before moving to the actual query generation phase, the tgds are rewritten by the rewriting engine in order to ensure that core solutions are generated.

Based on these rewritten tgds, an executable query either in SQL or in XQuery can be generated. The system integrates interfaces to various popular SQL and XQuery engines (like PostgreSQL and Saxon), so that the final query can be executed against one or more source instances and results inspected using the GUI. To simplify the debugging of the mapping scenario and to reduce dependencies wrt external systems, +Spicy also incorporates an internal chase engine to execute the tgds and generate solutions internally. In our experience, this is more immediate than sending a query to an external engine, and greatly helps users during their work sessions.

+Spicy can generate both the canonical universal solution generated by the original tgds and the core solution generated by the tgds after the rewriting, allowing the evaluation of the quality of solutions.

In terms of expressiveness, +Spicy can handle all of the mapping scenarios proposed in [8], but can also handle scenarios of the kind discussed in [9] by allowing users to explicitly manipulate join conditions.

Note that all algorithms discussed in the previous sections are applicable to both flat and nested data. However, data exchange research has so far concentrated on relational data and there is still no formal definition of a data exchange setting for nested data. But it's possible compare the solutions produced by the system for nested scenarios with the ones generated by the basic [70] and the nested [46] mapping generation algorithms, that we have reimplemented in our prototype. We show that the rewriting algorithm invariably produces smaller solutions, without losing informative content.

Chapter 5

Schema Mapping Verification

5.1 Introduction

As discussed in the previous chapter, several systems in recent literature have studied the problem of deriving mappings among sources based on value correspondences. A prominent example of a line–based mapping generation system is Clio [63, 70]. Clio implements a sophisticated mapping algorithm to generate source-to-target transformations, i.e., executable queries capable of translating an arbitrary instance of the source into an instance of the target. The mapping generation algorithm captures all semantical relationships embedded in the source and target schemas, and is guaranteed to produce legal instances of the target with respect to constraints.

It can be seen, however, how a crucial step in the mapping generation process is the discovery of the initial value correspondences. In fact, the quality of the mappings produced by the mapping generation system is strongly influenced by the quality of the input lines: starting from faulty correspondences incorrect mappings are inevitably produced, thus impairing the quality of the overall integration process. To avoid these problems, it is usually assumed that value correspondences are interactively provided to the system by a human expert after carefully browsing the source and target repositories. However, such manual process is very labor-intensive, and does not scale well to medium and large integration tasks.

To alleviate the burden of manually specifying lines, one alternative is to couple the mapping generation system with a *schema matching system*, i.e., a system that automatically or semi-automatically tries to discover matching

attributes in a source and target schema. The study of automatic techniques for schema matching has received quite a lot of attention in recent years; for a survey see [73, 38, 76]. Clio itself has been complemented with a companion schema matching module based on attribute feature analysis [67]; this tool may be asked to suggest attribute correspondences to the user.

Unfortunately, schema matching has been recognized as a very challenging problem [49], for which no definitive solution exists: although current schema matching systems perform well in some application categories, in other cases they suffer from poor precision. According to [50], there is no perfect schema matching tool. [50] reports that on a recent benchmark of ontology–matching tasks [4], participating matchers on average achieved 40% precision and 45% recall. Also, even for datasets for which such tools reached higher precision and recall, they still produced inconsistent or erroneous mappings.

As a consequence, outputs of the attribute matching phase are hardly ready to be fed to the mapping generation module, and human intervention is necessary in order to analyze and validate them. It is worth noting that, in general, human intervention is also necessary after mappings have been generated, since several alternative ways of mapping the source into the target may exist. Mapping systems usually produce all alternatives, and offer the user the possibility of inspecting the result of each of them in order to select the preferred one. Based on such an architecture, Figure 5.1 illustrates the overall mapping process and highlights the phases in which user intervention is required.

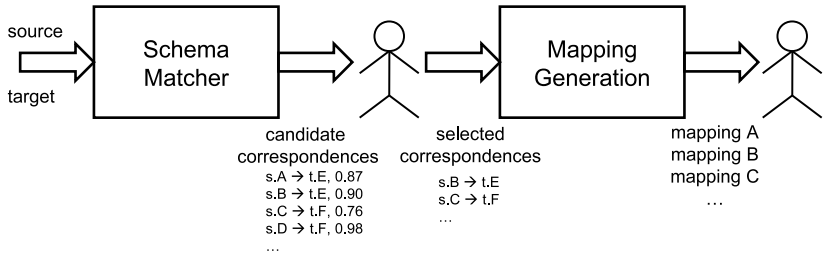

Figure 5.1: Coupling schema matching and mapping discovery

The main intuition behind our approach is that in many cases the mapping generation process can be automated to a larger extent by introducing a third step, which we call *mapping verification*. More specifically, we assume that instances of the target data source are available, and not only its schema. This

is typical in many settings, like, for example, Web data sources. In this case, whenever we select a set of candidate correspondences produced in the schema matching phase, we may think of checking the corresponding source-to-target transformation – i.e., the executable query induced by such correspondences – by running the query on (a subset of) the source, and comparing the result to the available target instance. Based on such comparison, we may on one side identify incorrect transformations due to wrong correspondences produced during the schema matching phase, and, on the other side, rank the remaining candidates to suggest to users the ones that more likely represent correct translations of the source.

The main contributions brought by the project can be summarized as follows:

(*a*) +Spicy proposes an original architecture to integrate schema matching and mapping generation; so far, these two problems have been studied essentially as independent steps of the integration process;

(*b*) +Spicy represents one of the first proposals towards the definition of a notion of *mapping quality* and the automation of *mapping verification*; we believe that such notions are crucial in order to improve the quality of current integration systems; we introduce an algorithm that combines schema matching, mapping generation and mapping verification in order to achieve good scalability and high matching quality;

(*c*) in view of this, +Spicy introduces an original approach, called *structural analysis*, to the comparison of data sources; structural analysis uses electrical circuits to compare the topology and the information content of tree-like structures in order to have a quick measure of their similarity; in the following sections, we show that structural analysis represents a very powerful tool for mapping verification;

(*d*) finally, the system architecture is generic and modular by nature; it can handle both flat (i.e., relational) sources, and nested ones, as XML repositories or OWL ontologies; also, it is designed to work in conjunction with any existing schema matching and mapping generation system, thus allowing to leverage the vast body of research in this field.

We believe the issue of coupling schema matchers with mapping generation systems, and the related issue of verifying mapping quality, represent relevant research problems. To the best of our knowledge, +Spicy represents the first proposal towards the automated verification of schema mappings. So far, the only step in the direction of verifying mappings produced by a mapping generation system has been presented in [31], under the form of an ad hoc tool that serves as a debugger for the mapping generation process. The tool allows users

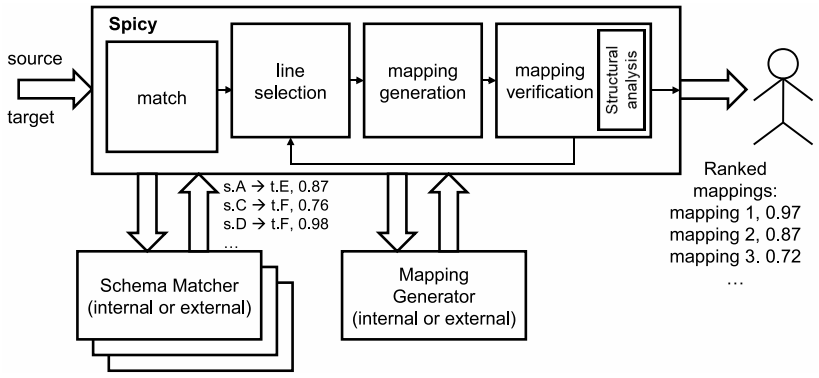

Figure 5.2: Architecture of Spicy

to trace and inspect mappings step by step during their generation, similarly to source code debuggers. Note that our approach is significantly different from the one pursued in [31], since we aim at reducing human intervention, while human users play a major role in the debugging process.

This chapter is organized as follows. Section 5.2 gives an overview of our approach and introduces the mapping verification algorithm by means of examples. Structural analysis is introduced in Section 5.3, and the mapping algorithm in Section 5.4. Experimental results are discussed in Section 5.5.

5.2 Overview

The mapping problem we want to solve can be informally described as follows: given (a portion of) a repository T, called the *target*, and a repository S, called the *source*, find a transformation query – i.e., an executable view – that can be used to map instances of S into instances of T. In the process, we aim at minimizing the amount of human intervention. The overall system architecture is shown in Figure 5.2. In order to solve the problem, we assume the availability of one or more schema matchers, and of a mapping generation module. The main advancement of +Spicy with respect to the simple pipelining described in Figure 5.1 is that these modules are decoupled by a number of intermediate additional modules that coordinate the mapping generation process in order to select the best results.

Given the variety of schema matching systems available, the system architecture has been designed in such a way that any of those can be easily integrated. We have tested the system using both a *schema-based* and an *instance-based* matcher [73]. As an example of a schema-based tool, the system may be integrated with COMA++ [38, 15]. As there were no instance-based tools available to us at the time of writing, we developed an internal instance-based matcher. Spicy's attribute matcher allows for two different matching strategies: on the one side, it may compare attributes in a rather standard way using features extracted by a sample of their values; on the other side, it may adopt structural analysis, as described below. Since the matching module is not the primary subject of this thesis, we will not elaborate further on this issue.

Several points are worth mentioning with respect to the matching phase. First, let us only note that, as it is common in the literature,[1] in this thesis we shall mainly concentrate on 1:1 attribute correspondences. However, the +Spicy schema matching module also handles a common class of n:1 element correspondences. Note also that, for the sake of simplicity, we do not consider more expressive classes of correspondences, like for example *contextual matches* [22], in which the same source attribute must be matched to different target attributes in different contexts. However, our setting can be extended to handle these cases. Finally, our architecture naturally lends itself to the integration of different schema matchers. However, integrating the outputs of different matchers goes beyond the scope of this thesis and is left to future investigations.

The system also assumes the availability of a mapping generation module. The mapping generation module takes as input a set of correspondences, and generates a number of mappings under the form of *source to target dependencies*. These mappings can then be assembled into complex *transformation queries*. A *transformation query* is the union of several mappings; it corresponds to an executable query that can be run over a source instance to obtain a target instance. Examples of transformations are provided below. To derive mappings in +Spicy, we have implemented a version of Clio's mapping algorithm [70]. However, other mapping generation tools, like, for example, HePToX [23], could be easily integrated into the system.

The first step of our mapping discovery procedure consists of running the schema matcher to produce a number of candidate element correspondences.

[1] In fact, essentially all systems in the literature with a few exceptions [37, 77] are restricted to 1:1 correspondences.

Each correspondence is usually labeled with a similarity measure, i.e., a level of confidence. Note that, in most cases, the schema matching system will not be able to produce a single correspondence for each target attribute. It will rather produce a number of compatible source attributes for each target attribute, with different degrees of similarity. Since in our setting each target attribute must be matched to a single source attribute, we need to consider these as alternative lines to be given as input to the mapping generation module.

Example 5.2.1 *Consider for example the data sources in Figure 4.2. We are adopting the nested relational data model presented in Chapter 2. Assume that, when run on the two data sources in our example, the selected schema matcher suggests the following correspondences:*

courseDB.course.name → *univDB.courses.course.courseName, 0.95*
courseDB.course.credits → *univDB.courses.course.credits, 0.92*
courseDB.course.instructor → *univDB....professor.name, 0.87*
courseDB.course.instructor → *univDB ...staff.name, 0.90*

It can be seen how we are deliberately assuming that the schema matcher considers staff names to be more similar to course instructors' names than professors' names. It is also apparent that such similarities cannot be fed to the mapping generation algorithm as they are; since attribute course.instructor *has been matched to two different source attributes, the system must choose between two different consistent line sets: in both sets course names and credits are correctly mapped to their counterparts; then, in set* (a) *instructors' names are mapped to professors' names, while in set* (b) *to staff names.*

If we were to select the preferred lines exclusively based on similarity values, we would choose the second candidate set. However, let us reason for a moment on the two transformations that would be produced. How these transformations are generated was discussed in major details in Section 4.2.

Set (a) *yields the following transformation (we are adopting the syntax used in [70]):*

```
for    d in univDB.departments, p in d.professors,
       c in univDB.courses
where  c.course.instructor = p.professor.id
exists c' in courseDB
where  c'.course.name = c.course.courseName and
       c'.course.credits = c.course.credits and
       p.professor.name = c'.course.instructor
```

```
UNION
for     d in univDB.departments, p in d.professors,
exists  c' in courseDB
where   p.professor.name = c'.course.instructor
```

The transformation is the union of two tgds. The first one states that, in order to obtain an instance of the target, we need to join courses and professors in the source, and then produce a tuple for each course name, credits and professor name. The second mapping, on the contrary, maps professor names. However, it only contributes to the result for those professors that do not have courses and therefore do not participate to the join above. If, how it is reasonable, we assume that the vast majority of professors have courses, very few tuples will be generated by the second tgd. Based on these observations, we may say that the translated target instance, reported in tabular form, would look as follows:

name	credits	instructor
Databases	6	Frank Castle
Networks	3	Scott Summers
...	...	...

Set (b)*, on the contrary, yields a quite different transformation:*

```
for     c in univDB.courses
exists  c' in courseDB
where   c'.course.name = c.course.courseName and
        c'.course.credits = c.course.credits
UNION
for     d in univDB.departments, s in d.staffSet
exists  c' in courseDB
where   c'.course.instructor = s.staff.name
```

In this case, both mappings contribute to the result, because nobody in the staff teaches courses. The resulting instance is quite different from the previous one, since the two mappings generate a number of null values:

name	credits	instructor
Databases	6	NULL
Networks	3	NULL
NULL	NULL	Frank Castle
NULL	NULL	Scott Summers
...	...	...

If we compare this second instance to the target instance shown in Figure 4.2, we may see how information "flows" quite differently through them. This might allow us to correctly infer that set (a) *more likely represents the correct choice.*

Although this example has been kept very small and rather simple for clarity's sake, we may summarize by saying that schema matchers typically output correspondences that may be assembled in different ways. This leads to different candidate transformations. We advocate that, while an a priori selection of the right correspondences is often very difficult and typically leads to poor results, an a posteriori comparison of the instances produced by the various transformations to the original target instance may give very useful insights on the quality of these transformations, and therefore on the correctness of the associated lines.

Note that a similar procedure may also help to rank transformations when the right correspondences are known, as shown in the following example.

Example 5.2.2 *This example is a variant of the running example used in [70]. Suppose both the source and the target are relational databases, as shown in Figure 5.3. We assume that correspondences have been identified without ambiguity. Still, the mapping generation algorithm will suggest two different transformations. This is a consequence of the multiple join paths that exist in the source among companies and funds: to each fund we may attach both the company to which the fund was granted and the company that served as a sponsor. As a consequence, the mapping generation algorithm cannot univocally cover correspondences for company names and budgets, and two different mappings are generated, as follows (variables corresponding to aliases have been emphasized):*

```
for    f in expenseDB.fundings,
       G in expenseDB.companies,
       S in expenseDB.companies
where  f.fund.grantee = G.company.companyId and
       f.fund.sponsor = S.company.companyId
exists p in projectDB
where  f.fund.projectName = p.project.name and
       G.company.name = p.project.company.name and
       G.company.budget = p.project.company.budget

for    f in expenseDB.fundings,
```

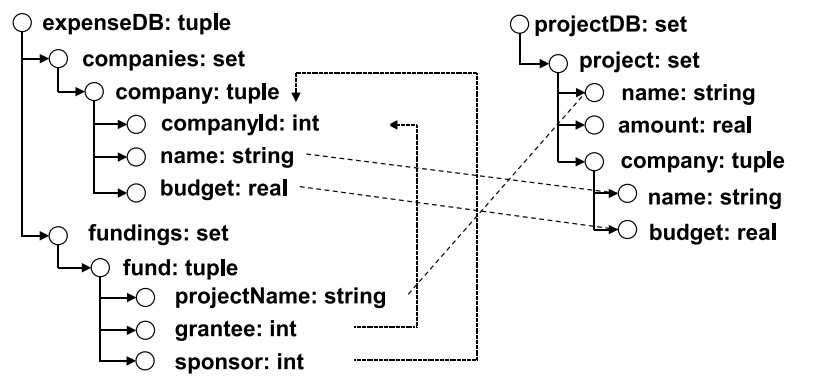

Figure 5.3: A mapping task with alias

```
        G in expenseDB.companies,
        S in expenseDB.companies
where   f.fund.grantee = G.company.companyId and
        f.fund.sponsor = S.company.companyId
exists  p in projectDB
where   f.fund.projectName = p.project.name and
        S.company.name = p.project.company.name and
        S.company.budget = p.project.company.budget
```

*In the first one, the company associated with the project in the target is the grantee (**G**), in the second case is the sponsor (**S**). In general, it is not possible to choose one of these mappings without resorting to explicit user feedback [81]. However, also in this case, by looking at the original target instance, we may have some hints. More specifically, companies in the source are divided in two categories: companies that sponsor projects, and companies that do not. It is conceivable that the first ones have considerably higher budgets that the second ones. As a consequence, the first mapping above will produce a translated instance in which budgets have lower values on average than the second mapping. Again, by comparing the two instances to the original target instance we may use this knowledge to choose the preferred mapping.*

A fundamental role in our architecture is played by the *mapping verification module*. Besides the availability of a source schema and instance, we assume

that instances of the target data source are also available, and not only its schema; this is typical in many settings, like, for example, Web data sources. In these cases, whenever we select a set of candidate correspondences produced in the schema matching phase, we may think of running the mapping generation algorithm to obtain the transformation induced by these correspondences; then, we may check such transformation by running the query on (a subset of) the source, and comparing the result to the available target instance. The level of similarity obtained by this comparison is taken as an estimate of the *quality* of the mapping. Based on such comparison, we may on the one side identify incorrect transformations due to wrong correspondences produced during the schema matching phase, and on the other side rank the remaining candidates to suggest to users the ones that more likely represent correct translations of the source.

These examples show that a post-translation check of translated instances against the original target instance may help to rank them and suggest the preferred ones. It is still open how such comparison must be conducted. In the following section we introduce a novel strategy to compare trees that serves this purpose.

Overview of Structural Analysis

Structural analysis is based on the adoption of a suitable model to analyze and compare data structures. Suppose we are given a portion s of a repository S and a portion t of a repository T; in order to compare them, we might proceed as follows: (i) analyze s using the selected model in order to derive a number of descriptive features; (ii) do the same for t; (iii) compare the features computed by the model and return an index of similarity that is as high as close are the selected features.

To do this, we have selected a model that we believe meets two critical requirements – simplicity and good level of abstraction: the model of electrical circuits.[2] The main intuition behind this choice is that a data structure can be seen as a "generator of information"; in essence, we may imagine that its elements – for example, its atomic fields like strings or numbers – tend to produce a flow of information in the repository – i.e., they generate an internal *stress*; each element, in turn, has some *consistency* due to its nature – for example, numbers may be considered of different consistency with respect to strings or dates; from the interaction between stress and consistencies we may predict the

[2] In fact, the name of the project is inspired by SPICE, the well-known circuit emulator.

flow of information. Intuitively, two instances will be considered to be similar when information flows similarly through them; in this way, circuits allow us to check the validity of mappings as discussed in Examples 5.2.1 and 5.2.2.

Electrical circuits exhibit nice and elegant properties that make them a very effective means to study similarities about data structures. In the following sections, we formalize a transformation function, that, when applied to a tree-like portion of a data source, generates an electrical circuit that can be used for the purpose of comparison. For now let us mention some features of this transformation.

First, the resulting circuit has a tree-like topology that is isomorphic to that of the original tree; also, the circuit embodies instance-based features; in fact, values for resistors and voltage generators associated with attributes are derived from numerical features of a sample of instance values; hence, it is an ideal means to model the informative content of the data structure.

Second, circuits are an excellent summarization technique for the purpose of verifying mappings. Recall that our goal is to evaluate the quality of a mapping by checking the similarity of the instance obtained by executing the mapping with respect to the instance originally provided with the target data source. To do this, in principle, we might adopt a rather straightforward attribute-to-attribute comparison approach, as follows: (*i*) after the translated instance has been generated, we might consider each attribute A_i in the target separately; (*ii*) take a sample of values for A_i in both the original and the translated instance, and derive a number of numerical features based on this sample; (*iii*) calculate the overall similarity between the two instances based on the similarity of the various features. However, combining independent similarity measures into a single similarity value is known to be a hard problem [49]. Circuits provide a nice and elegant solution to this problem. They naturally lend themselves to a black-box analysis to characterize the behavior of the two trees, through which we can collect a small number of descriptive parameters – output current, average current, internal voltages – that can be effectively used to measure the similarity of the original structure to other structures. Our experiments show that circuits perform better than attribute-to-attribute feature comparisons.

Finally, the comparison technique based on circuits is fully automatic and does not require any additional inputs; it is also quite efficient, since, for the kind of stationary, continuous current circuits used in the system, solving the circuit amounts to solve a system of linear equations of the form $Ax = b$.

The Mapping Search Algorithm

We now give an intuition of the mapping discovery and verification process pictured in Figure 5.2. A key observation, here, is that the mapping generation process can be quite demanding from the computational viewpoint. As a consequence, it is mandatory to design the search algorithm in such a way to achieve a good trade-off between precision and scalability. Recall that our starting point is a number of candidate correspondences generated by the schema matching module. Whenever multiple source attributes are matched – possibly with different degrees of similarity – to the same target attribute, we need to generate alternative transformations and then rank them. The higher is the number of correspondences produced by the schema matching step, the more mappings will be generated and verified.

In view of this, our algorithm is based on a feedback loop in which we initially fix a similarity threshold; at each step: (*i*) we consider only candidate correspondences with similarity above the threshold; (*ii*) we combine these correspondences into consistent sets, and run the mapping generation algorithm to generate the corresponding transformations; (*iii*) we use each transformation to translate (a sample of) the source instance into an instance of the target, and then use structural analysis to compare the obtained instances to the original target instance; (*iv*) we rank transformations – and therefore correspondence sets – according to such similarity measure. If no satisfactory mappings are produced at a given iteration, we progressively lower the threshold and analyze more alternative mappings. In essence, using this process, users are not required to manually inspect and select correspondences before mappings are generated; also, when the process stops, the system will present to them a ranked list of source-to-target transformations, suggesting which ones are believed to better reproduce the target.

A key idea in this process is that outputs of the schema matching module are not considered as a definitive proof that a source and target attribute are semantically equivalent, but rather as a measure of compatibility between them. Such compatibility is then checked in subsequent steps. Our experimental results show that our verification algorithm achieves very good precision and allows to handle many errors due to incorrect matches.

The following sections introduce with greater detail the algorithms used in the system.

5.3 Structural Analysis

To compare repository portions we transform their trees into circuits, solve the circuits, and then compare features of the two circuits (e.g., output currents). In the following, we first introduce the circuit mapping function, and then the actual compare procedure.

Electrical Circuits

We consider electrical circuits [32] made of elements connected to nodes. In our case, elements may be voltage generators and resistors. In such circuits, voltage generators cause a flow of current through resistors. A voltage generator imposes a constant difference V_k of electric potential between two nodes. A resistor of value r causes a drop of potential V at its nodes due to the current I that flows through it; current and voltage are related by *Ohm's law* $V = rI$. A circuit usually has two distinctive nodes: (i) a *ground node*, whose potential is conventionally 0; voltages at other nodes are measured as differences of potential with respect to the ground; (ii) an *output node*, i.e., a node at which output voltage and current are measured. We define the *output current* of the circuit as the current flowing through the output node when an *external resistor* of fixed value is added from output to ground. Solving the circuits amounts to calculating voltages at the nodes and currents through elements. This corresponds to solving a system of linear equations based on Kirchhoff's law of size $n \times n$, where n is the number of nodes in the circuit.

Sampling

In order to build the circuit associated with a data source, we need to sample instances. For each leaf A in a tree we select a random sample of instances, $sample(A)$ of size K (or less) from the repository. Two attributes are considered to match when they represent the same concept, and their values are coded using the same format and conventions. As a consequence, we are interested in studying features such as the length of values in the sample and the distribution of characters. For each attribute A we build a number of distributions. The *distributions of lengths* $\mathcal{D}_L(sample(A))$ contains the frequencies of lengths in the sample, measured in bytes. Fixed an ordered collection of character categories $\mathcal{C} = \{c_0, c_1, \ldots c_{n-1}\}$ (i.e., letters, capitalized letters, digits, special characters, at-signs, slashes etc.), we also compute the *distribution*

of character categories, $\mathcal{D}_{\mathcal{C}}(sample(A)) = \{f_{c_0}, f_{c_1}, \ldots f_{c_{n-1}}\}$, where f_{c_i} is the frequency of characters belonging to category c_i in the sample.

Given a distribution of frequencies $\mathcal{D} = \{f_0, f_1, \ldots, f_{n-1}\}$ such that $\sum_{i=0}^{n-1} f_i = 1$, we measure several statistical parameters. Besides the usual ones (mean value, standard deviation, mode), we also consider the *Simpson concentration index* [69] of the distribution, defined as $IC(\mathcal{D}) = \sum_{i=0}^{n-1} f_i^2$; intuitively, this index gives a measure of the entropy of values in the distribution, in the spirit of information theory: the higher is the entropy in the sample, the lower is the concentration index.

Finally, we define the *density* of a sample of values of attribute A, *density*(*sample*(A)) as n/K, where K is the size of *sample*(A), and n is the number of non-null values in *sample*(A). This is a measure of the number of null values generated by a transformation, as discussed in Example 5.2.1.

Circuit Generation Function

We introduce a recursive function $circ(t)$, that, given a tree t generates a circuit. In order to do this, we need to formalize the output of $circ()$ on an atomic attribute, i.e., a leaf in the tree. There are several ways to map a sampled attribute to a circuit; intuitively, this depends on the features one decides to embed into the circuit, and on the topology of circuit elements that represent them. In the following, we describe the strategy that has provided the best results in our experiments. Given an attribute A in a schema tree, annotated with a sample of values, *sample*(A), with length and character distributions $\mathcal{D}_L$ and $\mathcal{D}_{\mathcal{C}}$ respectively, we define the following features:

- *density index* (**ID**): $ID(A) = density(sample(A))$;
- *consistency* (**C**): $C(A) = 2^0 \times f_{c_0} + 2^1 \times f_{c_1} + \ldots 2^{n-1} \times f_{c_{n-1}}$, where $c_0, c_1, \ldots c_{n-1}$ are character categories, and f_{c_0}, $f_{c_1}, \ldots f_{c_{n-1}}$ the respective frequencies in $\mathcal{D}_{\mathcal{C}}(sample(A))$;
  for the sake of convenience we will refer to the polynomial function above simply as $C(A) = pol(\mathcal{D}_{\mathcal{C}}(sample(A)))$;
- *stress* (**S**): $S(A) = mean(\mathcal{D}_L(sample(A))$, i.e., the mean value of the distribution of lengths in the sample;
- *conc. index 1* (**IC1**): $IC1(A) = IC(\mathcal{D}_{\mathcal{C}}(sample(A))$, i.e., the index of concentration of the distribution of character categories;

- *conc. index 2* (**IC2**): $IC2(A) = IC(\mathcal{D}_{\mathcal{L}}(sample(A))$, i.e., the index of concentration of the distribution of lengths.

$circ(A)$ is assembled by assigning these features to a number of resistor and voltage generators, as described in Figure 5.4.

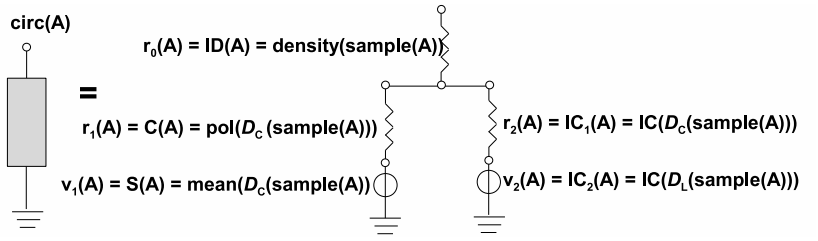

Figure 5.4: Electrical circuit for an atomic attribute

The circuit for a tree is easily constructed starting from building blocks corresponding to atomic attributes; more specifically, for each intermediate node n in a schema tree t, we define a resistance value, $r(n)$. Such value cannot be based on instances, since intermediate nodes do not have a sample of instances, but rather on the topology of the tree. More specifically, we define $r(n) = k \times level(n)$, where k is a constant multiplicative factor, and $level(n)$ is the *level* of n in t, defined as follows: (i) leaves have level 0; (ii) an intermediate node with children $n_0, n_1, \ldots n_k$ has level $max(level(n_0), level(n_1), \ldots level(n_k)) + 1$; $nodes(t)$ will denote the set of nodes in tree t.

We can now give a complete definition of the *circuit mapping function*, $circ(t)$ over a tree t. For a leaf node A, $circ(A)$ is as defined above. For a tree t rooted at node n with children $n_0, n_1, \ldots n_k$, $circ(t)$ is the circuit obtained by connecting in parallel $circ(n_0), circ(n_1), \ldots circ(n_k)$ between ground and an intermediate circuit node n_{top}, and then adding a resistor of value $r(n)$ from node n_{top} to output. Examples of such transformation are given in Figure 5.5. Note how the resulting circuits are essentially isomorphic to the original trees.

Note that, coherently with the *opaque* [55] nature of our approach, labels are not taken into account by the circuit mapping function, and we are treating values essentially as uninterpreted strings. Also, we concentrate on ordinary alphanumeric data: the features discussed above reflect this choice. However, the circuit model is sufficiently flexible to allow the treatment of special data,

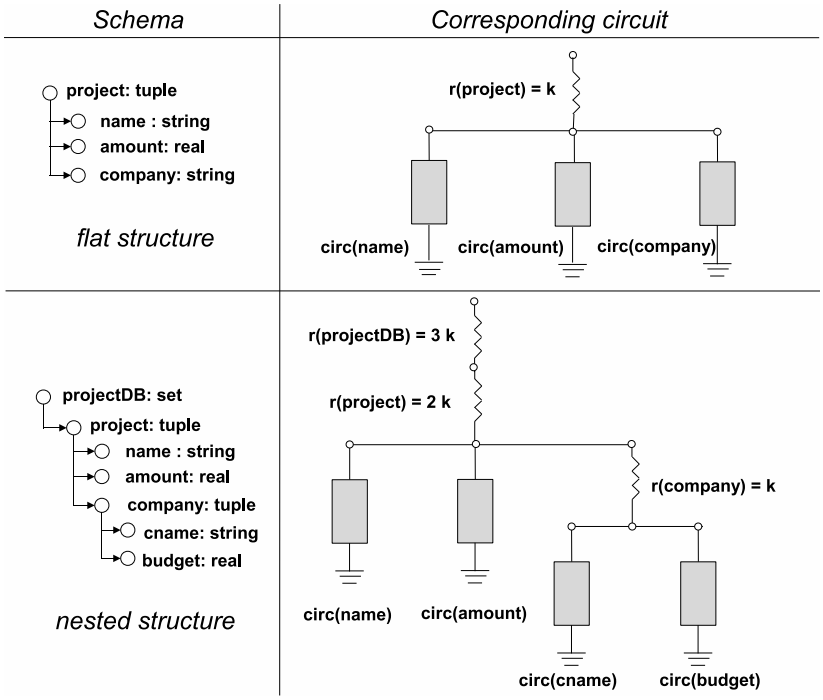

Figure 5.5: Examples of Circuits

like large texts or multimedia, as well; for these data different features must be adopted [44].

Compare Procedure

Similarly to [39], our compare module adopts a modular library of comparators, that can be mixed and matched in various ways. Given two trees t_1 and t_2, we compute a measure of their similarity, as follows: (i) map t_1 and t_2 to the corresponding circuits, $circ(t_1), circ(t_2)$; (ii) solve the two circuits to determine currents and voltages; (iii) choose a number of descriptive features of the corresponding circuits, $f_0, f_1, \ldots f_i$; we introduce the notion of a *comparator* for feature f_i as a module that computes the index of similarity Δ_i between

the two structures with respect to feature f_i, as follows $\Delta_i = abs(f_i(circ(t_1)) - f_i(circ(t_2)))/f_i(circ(t_1))$; (iv) finally, we compute the overall similarity of the two trees based on $\Delta_0, \Delta_1, \ldots \Delta_i$.

A very delicate problem in this setting is represented by assembling different similarity measures into an overall quality estimate; to this end, the system provides three alternative strategies to compute an overall similarity measure based on a collection of feature similarity values like Δ_0, Δ_1, ... Δ_i, namely: standard average (A), i.e., arithmetic mean, of similarity values; harmonic mean (HM) of similarity values; euclidean distance (D); in this case, candidates are considered as points in an n-dimensional space, whose coordinates correspond to $\Delta_0, \Delta_1, \ldots \Delta_i$; euclidean distances give a measure of how far away a point is from the origin or from another point; in this case two points are considered to be the more similar the less is their distance.

Using such a setting, we may compare circuits – and therefore trees – using different collections of comparators and aggregation strategies. To give an example, suppose we decide to use output current alone as a comparison feature. In this case, we would have a single comparator, and similarity would be equal to the percentual difference between the output currents measured in the two circuits. If we decide to employ both output current and average current, than we have two different comparators, and we need to choose an aggregation strategy – say, harmonic means; the overall compare will be the harmonic means of $\Delta_{outCurr}$ and $\Delta_{avgCurr}$. Similarly for other features. In Section 5.5 we discuss how the selection of comparators affect precision.

5.4 Mapping Search Algorithm

We can now describe the mapping discovery and verification process pictured in Figure 5.2.

A *mapping task* for +Spicy is a pair of data sources, $< S, T >$, where S is the source and T the target. As a first step, the source and target are submitted to the schema matching module, and a list of candidate correspondences is generated. This step is performed only once for each mapping task. The line selection module of +Spicy stores the result of the match step, and generates inputs to the subsequent phases. Typically, it will only consider a subset of correspondences, those with a confidence level above a fixed threshold, th.

As discussed in Section 5.2, it is possible that multiple source attributes are matched – possibly with different degrees of similarity – to the same target attribute. We call *unambiguous* a collection of correspondences if each attribute –

both from the source and the target – appears in at most one of the correspondences. Correspondences generated by the matching module contain in general ambiguities, and therefore need to be partitioned into a number of alternative unambiguous sets. Each unambiguous set of correspondences will then be fed to the mapping generation module, and will produce a transformation query. These transformations need to be ranked by running structural analysis.

Before turning to the description of the actual algorithm, we need to make some preliminary observations. A key point, here, is that the mapping generation process can be quite demanding from the computational viewpoint. As a consequence, it is mandatory to design the algorithm in such a way to achieve a good trade-off between precision and scalability. It can be easily seen that the higher is the number of correspondences that are considered, the higher is the probability of finding ambiguities, and therefore the number of instances of the mapping generation process to execute. It is therefore very important to choose the confidence threshold, th, in such a way to be selective enough and discard poor mappings, without excluding promising ones. High values of the threshold generate a few candidate mappings, potentially none, and may reduce precision. Low values increase the number of candidates, and may impose a significant overhead in terms of computing times. In fact, experiments confirm that precision tends to increase as the threshold decreases. However, this improvement in precision has a cost in terms of computing times: the number of candidate mappings tends to increase very rapidly, and computation times with them.

Based on these results, it can be seen that it is very difficult to statically fix an optimal value for the compatibility threshold th. In view of this, we adopt a *dynamic threshold*, initially very high in order to consider only a few candidate correspondences; then, if no satisfactory mapping is produced, we progressively lower the threshold and analyze more alternative mappings. In essence, the mapping algorithm starts with a very high value for the compatibility threshold (0.99). If no mapping of acceptable quality is produced, the algorithm backtracks, it lowers the threshold (of 0.01), and iterates. By lowering the value of th, more correspondences are taken into account, and therefore more potential mappings. This process proceeds until either a mapping is produced, or the threshold falls below a stop value, th_{stop} (0.6).

Given a mapping task $< S, T >$, the algorithm can be sketched as follows:

1. run the external schema matcher module to generate candidate correspondences, $match(S, T)$;

2. fix a step, *step*, and a stop value th_{stop} for the compatibility threshold; initialize $th = 1 - step$; fix a minimum value for mapping quality $minQ$;

3. consider all correspondences in $match(S, T)$ with confidence above th; assemble them into maximal unambiguous subsets;

4. feed each subset of lines $\mathcal{S}_i$ to the mapping generation module, and generate the corresponding transformations (one or more, depending on the presence of aliases in the data sources); process each transformation using the internal execution engine to translate a sample of the source instance into an instance of the target;

5. sample each translated instance and use structural analysis as discussed in Section 5.3 to compare it to the original target instance; the quality of the associated transformation is given by the compare value;

6. if no transformation with quality above $minQ$ is produced, then $th = th - step$, and iterate step 3; otherwise output all transformations with quality above $minQ$, ranked according to their estimated quality.

Note that at each step a number of candidates that have already been processed in previous steps are re-generated. An aggressive caching strategy has been implemented into the system in order to improve efficiency. More specifically, we cache logical relations, mappings and circuits in order to avoid repeated computations along the path. This is quite effective in reducing the time cost but makes the algorithm quite demanding in terms of memory.

5.5 Experimental Results

We have used the prototype presented in Section 4.4 to run a number of experiments. Table 5.1 summarizes the list of experiments that were used to test the system. We analyzed 12 mapping tasks, based on different data sources, both relational and XML. Besides well known data sources like DBLP, Mondial [3], and Amalgam [4], we tested several real–life databases serving the information system of our School of Computer Science (CS-IS, LbDb, Moodle), and some synthetic datasets.

Experiments were run on an Intel core-duo processor machine, with 2 GB of RAM. Mapping tasks were designed in such a way that the source was known

[3] http://www.dbis.informatik.uni-goettingen.de/Mondial/

[4] http://www.cs.toronto.edu/~miller/amalgam/

Source	S. Type	Target	Targ Type	S. size (nodes)	T. size (nodes)
LbDb	rel	Comp. Science IS	rel	54	6
Moodle	rel	LbDb	rel	596	6
Comp. Science IS	rel	LbDb	rel	276	4
Amalgam1	rel	Amalgam2	rel	18	6
DBLP	xml	Amalgam1	rel	89	5
DBLP	xml	Amalgam2	rel	89	5
Mondial	xml	CIA Factbook	xml	222	14
Mondial	xml	Global Statistics	xml	222	4
Mondial	xml	Mondial Flat Europe	xml	222	9
Census1	xml	Census2	xml	18	6
UniversityDB	rel	Personnel	rel	28	5
StatDB	xml	ExpenseDB	rel	25	13

Table 5.1: Summary of Experiments

to contain a mapping for the entire target. For each target, we identified the correct set of correspondences that would generate such mapping. These correspondences were called the *ideal match* $\mathcal{M}_{id}$; then, we ran the system, and considered as output a single transformation, T_{best}, the one with the highest similarity score. We considered the value correspondences from which T_{best} was generated, called $\mathcal{M}_{T_{best}}$. We measured quality in terms of precision and recall of $\mathcal{M}_{T_{best}}$ with respect to $\mathcal{M}_{id}$. Note however that, in this section we report precision results only. This is due to the fact that in all cases the system returned a number of correspondences that was equal to the size of the target (combinations with less correspondences were discarded since they lowered the similarity to the target instance due to excessive presence of nulls); as a consequence, precision and recall are both equal to the number of correct correspondences in $\mathcal{M}_{T_{best}}$ over the size of the target.

A key issue to validate our approach was to compare the quality obtained by selecting attribute matches *a posteriori*, i.e., after mapping generation and

translation, with respect to selecting them *a priori*, i.e., relying only on the output of the attribute matcher. To test this, we have run our experiments with several configurations.

Average Match (AM): the best line set was chosen a priori, as follows: the attribute matcher was run to find candidate correspondences; then, the unambiguous line sets (one or more) with the highest confidence were selected; attribute similarities were aggregated in various ways to give variants of this configuration: **AM-A** (average match computed as average), **AM-HM** (average match computed as harmonic mean), **AM-D** (average match computed as euclidean distance). An alternative to these approaches would be to adopt a *top-K* strategy, as in [48]. These configurations were applied to the various schema matchers employed by the system, i.e., Spicy's instance-based matcher based on attribute features **Spicy-F**, and Spicy's matcher based on structural analysis, **Spicy-SA**.[5]

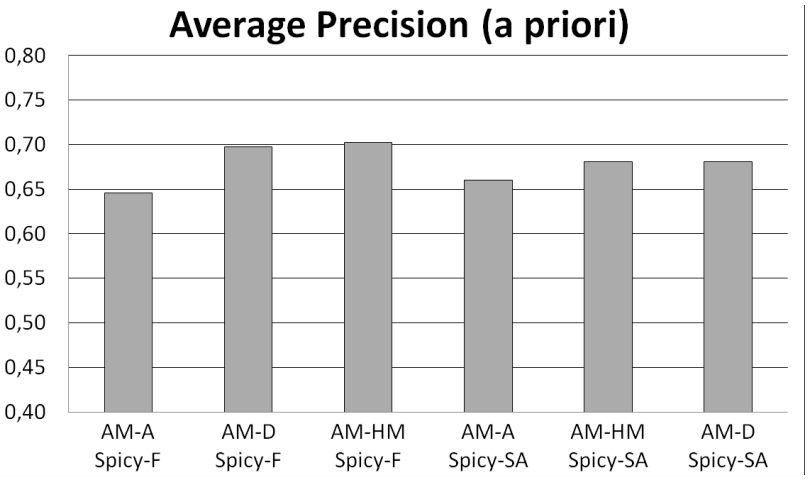

Figure 5.6: Precision of a priori strategies (average match)

To assess a posteriori strategies, a second, important point, is how structural analysis performs with respect to more traditional comparison procedures,

[5]We have also run several preliminary experiments using COMA++. We saw that the performance of COMA degrades on opaque schemas or schemas with very different vocabularies. Overall, its performance was in line with those of Spicy's instance-based matchers.

based on attribute features alone. To do this, we have tested two different configurations:

Attribute Features (AF): the best line set was chosen a posteriori, as follows: the mapping find algorithm was run as described in Section 5.4; however, after candidate transformations were run on the source to obtain a translated instance of the target, their quality was measured by comparing it to the original target instance in a standard fashion, i.e., by comparing each attribute in the translated instance to its counterpart using instance-based features; we adopted the same features that had been used to compare source and target attributes during the match phase (**ID, S, C, IC1, IC2**, as defined in Section 5.3), and then assembled the similarity measures using average and harmonic means (**AF-A, AF-HM**). Since, as discussed in Section 5.3, our comparison algorithm has been designed to allow for the combination of different comparators, we tested different combinations of these.

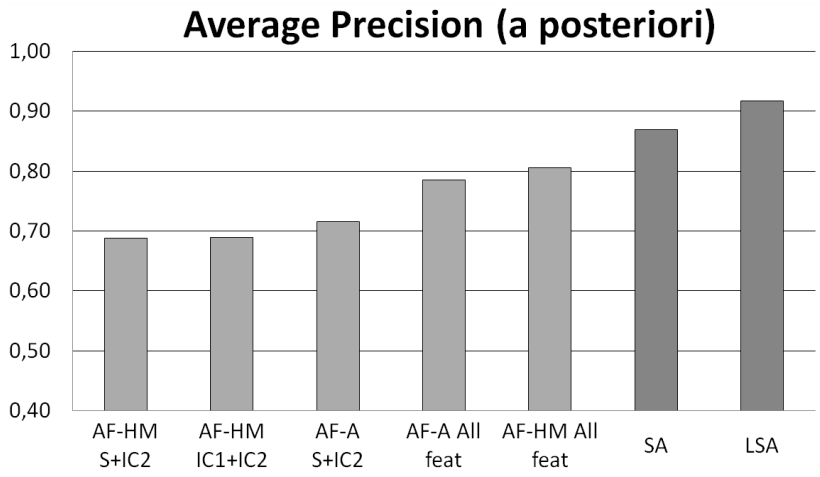

Figure 5.7: Precision of a posteriori strategies (attribute features and structural analysis)

Structural Analysis (SA, LSA): finally, we used structural analysis – i.e., circuits – as described in the previous sections. Among the many different combinations of comparators based on circuit features, we found these two to be the most interesting: **SA**: the current in each branch of one circuit is compared

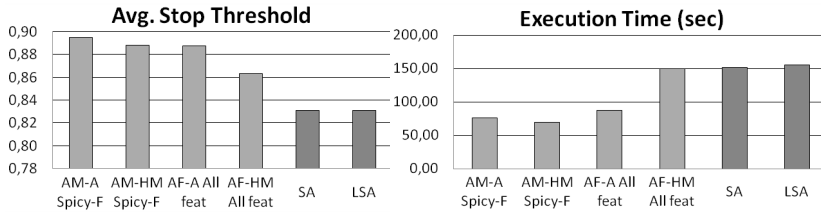

Figure 5.8: Stop thresholds and execution times

to the current in the corresponding branch of the second circuit; **LSA**: the *global* output current, i.e., the output of the whole circuit was considered, plus the *local* output currents, i.e., output currents in each subcircuit corresponding to an attribute.

Average precision over the 12 experiments for a priori strategies is shown in Figure 5.6, while average precision for a posteriori evaluations is shown in Figure 5.7. As a first observation, note that, as it was expected, all a priori configurations had mediocre performance – below 70%, while significantly higher values of precision were obtained by some a posteriori configurations. This confirms the intuition of Examples 5.2.1 and 5.2.2 in Section 5.2: since the schema matcher does not take into account semantic mappings, as a posteriori verification strategies do, it is frequently mistaken.

A second key observation is that structural analysis had excellent performance: both configurations based on circuits outperformed attribute features, whose best precision was around 80%, while the LSA configuration has precision above 90%, thus confirming the effectiveness of circuits.

More results are shown in Figure 5.8, in terms of average stop threshold and total execution times. It can be seen that verification strategies based on structural analysis have lower stop thresholds on average; intuitively, this is due to the fact that, given the search algorithm discussed above, they perform deeper searches before finding high quality solutions; this also explains why they achieve higher precision than other strategies. This higher level of accuracy comes at the price of higher execution times. This increase, however, is acceptable, since under the **LSA** configuration the 12 experiments were run in less than 3 minutes.

Chapter 6

Related Work

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

Data Exchange and Query Answering

Data exchange is the problem of generating an instance consistent with the target schema, given a source instance and a specification of the relationships between source and target schemas. This is not a recent problem, in fact one of the first data exchange systems was EXPRESS [75], presented in 1977. It was a complex system with the main functionality of converting data between hierarchical schemas. In the last years, with the increasing of web data, also the need for data exchange increased.

The *data exchange problem* is strictly related to the *data integration problem*, since both problems manage data stored in different formats, but these are different for some reasons. One of the main differences is that in data exchange the main focus is on *moving* data from the source and on *materializing* a target instance; in data integration, instead, no exchange is required and the main focus is on answering queries posed on a global schema, i.e. one or more views expressing the relationships between the source and the target schemas.

There exist various algorithms that have been developed in the literature. A key problem, however, is represented by the nature of the mappings that these algorithms assume are given as input.

In some cases, for example, the translation system assumes that transformations are specified as declarative rules in suitable languages. Various systems have been defined to this end. Examples are YAT [33], TransCM [66] and WOL

[35]. A survey is contained in [5]. These systems assume that the human developer specifies translation rules using the language; the system will compile and execute the translation based on these rules.

Query answering algorithms, on the other side, typically assume that structures in one source are described in terms of views over the other source. This approach is rather typical in mediator-based systems. There are essentially three ways to specify mappings among the global and the local schemas:

- LAV (Local As View) [80], in which local relations are defined as views over the global schema;
- GAV (Global As View) [52], in which global relations are defined as views on local schemas;
- GLAV (Global Logic As View) [45], in which views over the local schemas are mapped to views over the global schema. Based on such mappings, view expansion algorithms [56] have been defined to execute integrated queries. Similar techniques have been used also in the peer-to-peer context [28] [23] [47].

A large body of research has revolved around formalizing the notion of data exchange problem. Motivated by their experience with Clio project, Fagin et al. presented important theoretical results in [42]. They introduced the use of tuple generating dependencies (tgds) [56] and the definition of universal solution. In [42] it has been shown how it is possible to generate universal solutions by applying the chase algorithm [6] to the source instance and to the dependencies; this work have also pinpointed the computational complexity and the polynomial tractability under certain conditions.

An extension of the chase algorithm for universal models other than the standard chase applied to universal solutions in data exchange has been analyzed in [36]. The standard chase has been proved to be incomplete for universal models and an extended core chase definition is proposed.

Fagin et al. studied the data exchange problem for relational data, while the basic properties of XML data exchange was introduced in [11]. They define XML data exchange settings in which source-to-target dependencies refer to the hierarchical structure of the data, with particular attention to the consistency problem and the complexity.

Core Computation

The notion of a core solution was first introduced in [43]; 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, [53]). Note that computing the core of an arbitrary instance is an intractable problem [43, 51]. 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 [43] 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 [51]. 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 $O(vm^{\lfloor b/2 \rfloor + 2})$, with special benefits if the target constraints of the data exchange scenario are LAV (*local-as-view*) tgds. One of the algorithms introduced [51] has been revised and implemented in a working prototype [74]. 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 Chapter 3, 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 [30, 78, 59]. [30] 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 [59]. The algorithm presented in [59] is the two step algorithm outlined at the beginning of Section 3.5, and then further discussed in Section 3.6. Here, 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 in [59].

In [78] the authors independently developed 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 thesis.

First, the algorithm proposed in [78] is more general than the one proposed in this thesis, 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. However, the only practical algorithm to implement *certain* proposed in the paper relies on a variant of the MiniCon [71] algorithm, which only works for conjunctive formulas.

In terms of dependencies generated by the rewriting, a laconic mapping tends to contain a lower number of dependencies with more complex premises with respect to a core schema mapping, which typically contains more rules. In fact, 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.

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 an high-complexity algorithm (MiniCon). Our algorithm, albeit exponential, looks for formula homomorphisms, whose number is tipically lower.

We believe that both approches generate efficient scripts for the generation of core solution, but no implementation of the laconic mappings algorithm is available at the moment, so that it was not possible to compare the performance of our approach to that of laconic mappings.

Mapping Generation Algorithms and Mapping Tools

The original schema mapping generation algorithm was introduced in [63, 70] 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 [8] or those in [9]. Note that the need for explicit joins was first advocated in [72]; the duplication of symbols in schemas has been first introduced in the MapForce commercial system [1].

The amount of redundancy generated by basic mappings has motivated a revision of the algorithm known as *nested mappings* [46]. 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 [27]. 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.

We note that the notion of provenance in a schema mappings framework has been studied in [31] under the notion of *routes*. In this thesis we make a more restricted use of the notion of provenance.

Our system adopts a Clio-style algorithm for mapping generation. Clio [63, 70] does not offer a module for automatic mapping verification, but supports an interactive mapping refinement process by visualizing mapping examples, i.e. smaller samples of the source instance that has been translated using the current mapping. These may help the user to select among alternative solutions.

An early attempt at semi-automatically generating transformation expressions for different data formats was reported in [66]. Their approach is based on a library of pre-determined mapping-rules, and is better suited to those cases in which the target and source schemas have high similarity.

Clio has pioneered the field of schema mapping generation algorithms and in the last years many schema mapping tools have been developed, both commercial tools such as Altova MapForce [1], StylusStudio [3], BizTalk [2], ADO.Net [7] and research prototypes, such as Rondo [61], HepToX [23], and Spicy [24].

All these tools allow to specify correspondences as a set of lines using a visual interface.

Research prototypes (e.g. Clio, Rondo and Spicy) also allow the generation of a set of mappings in a logic formalism from these correspondences and the executable transformations that materialize the target database in a data exchange setting. On the other hand, no intermediate logic formalism is used by commercial tools (such as MapForce, StylusStudio, etc.), and the executable

transformations are produced in a query or transformation language, such as XSLT or XQuery.

Clio is among those data exchange systems that are restricted to a specific data model - essentially a nested relational model; to give an example, Clio cannot handle the class of transformations that are generated in [60], which involve object hierarchies. Such mappings allow developers to interact with a relational database via both conceptual schemas and object-oriented programming interfaces.

With respect to debugging schema mappings, it is a recent research topic and has been addressed in [31]. There, schema mappings can be traced and inspected, similarly to source code instructions. Their approach gives the user a major role in debugging the schema mappings, thus being significantly different from ours.

In [22], contextual schema matching is introduced, to denote correspondences annotated with a predicate saying when the match is valid. These conditions are then translated into actual views, and the mapping generation is done by extending the Clio algorithm. We currently do not deal with contextual matches, but our setting can be extended to incorporate such kind of matches.

Finally, an early attempt towards defining operations on schemas, including mappings, has been done in model management research [18, 13], as we will discuss in the next Section. The need for more expressive mappings has been advocated in [18]. In their work, they address the so called data programmability problems, which basically deal with the complexity of handling different heterogeneous models and coping with the impedence mismatch between applications and databases. To date, none of the above mapping systems has steadily solved the data programmability problem, or has adopted a more generic class of mappings, that goes beyond the expressiveness of the above models.

Model Management

The general framework of model management has been introduced in [17] [19]. The idea is to develop a generic infrastructure to obtain a significant productivity improvement to builders of applications by offering them high-level operations on schemas and mappings between schemas. The main operators on schemas are *match*, *merge* and *compose*; the *modelGen* operator corresponds to the translation of schemas between different data models.

The problems of schema translation and data translation have been considered in the database literature for various decades. A comparison of data models is reported in [79] [54]. Schema translation and data translation can be seen as related but different problems: schema translation assumes that we are given a "source" model, a "target" model, and a schema in the source model, and we want to obtain a schema in the target model that suitably "represents" the source schema; data translation assumes that we are given a source and target schema (possibly in different models) and a source database and we want to obtain a target database that represents the same information as the source one. Indeed, the two problems can be seen as two facets of a single, composed problem: given a source schema and a source instance, find a target schema (as a result of schema translation) and a target instance (as a result of data translation, as at this point the target schema is available).

In both problems, it is important to understand what are the properties that the target schema and database have to satisfy, in order to be considered suitable representations of the source ones. In this area, important work has been developed over the years [12] [54] [64] [65].

Given the difficulty of the schema translation problem, there is no complete general approach available to its solution, but there have been a few partial efforts [25].

In [14], the notion of a metamodel is introduced. The approach is based on the observation that the constructs used in most known models can be expressed by a limited set of generic metaconstructs: lexical, abstract, aggregation, generalization, function. In MDM, the metamodel allows to describe any data model in terms of these generic metaconstructs. Similarly for schemas of a model. The translation of a schema from one model to another one is defined in terms of translations of metaconstructs, rather than translations for every pair of models.

The solution introduced within the MDM framework is reasonably effective, but it is not really flexible; in particular, it does not consider the data translation problem. Recent extensions [13] [20] represent a first step forward, but they are quite preliminary.

Schema Matching

The fact that schema matching tools may return uncertain results has inspired an active body of research [48, 50]. In [48], it is highlighted that a schema matcher often tries to derive a single best set of correspondences, whereas in most cases the discarded correspondences may convey useful information.

An heuristic based on confidence values is used to verify the top-K mappings yielded by a schema matcher, and refine the matches accordingly.

Many tools for semi-automatic schema matching have been proposed in the past. For a complete reference, we refer the reader to comprehensive surveys in [73, 38, 76]. Schema matching systems are usually classified in two main categories: *schema-based* systems use a combination of linguistic and graph-based techniques in order to find similarities in schema labels; *instance-based* ones rely on actual values in instances to derive attribute features and similarities. Although schema-based tools represent the vast majority, instance-based tools perform better than schema-based ones in those contexts in which the databases are essentially *opaque* [55], i.e., labels and/or values are difficult or impossible to interpret.

COMA++ [39] bases on well-founded software engineering principles to build a fully-fledged set of matching techniques. It introduces the notions of reuse and composition of matchers.

Similarity Flooding [62] (SF, for brevity) proposes a structural algorithm that can be used to compute similarities between arbitrary data structures, such as schemas, instances or both. Despite the apparent similarity between the terms "flood" and "current flow", the two systems have hardly any points in common. Given two structures to compare – called models – SF runs a fixpoint algorithm over an auxiliary data structure, called a *similarity propagation graph* in which elements of the first and second model are embedded together; similarities are propagated along the edges of such graph according to the intuition that two nodes are similar when their adjacent elements are similar. On the contrary, in +Spicy, the two tree structures to compare are kept separate, and no common data structure is employed; moreover, each tree is turned into an isomorphic circuit, and the circuit is solved to calculate currents and voltages; this is based on a very different intuition with respect to SF, namely that currents model the "flow of information" inside the tree; no similarity flows through the circuit. Similarities in +Spicy are explicitly computed by selecting a number of features that describe the circuits response and measuring their distance.

Both SF [62] and COMA++ [39] are schema matchers, thus can be used within our matching module.

Among the instance-based mapping tools, we recall [21], [55], [58], and [40]. A more exhaustive survey is beside the scope of this thesis and can be found in [38].

DUMAS [21] exploits the presence of duplicates within relations to effectively drive the schema matching process. Despite their actual labels, attributes

that are semantically similar are detected and used to output the mapping correspondences. +Spicy does not assume the presence of duplicates.

In [55], the authors develop an instance-based approach to matching opaque databases, i.e., database in which labels and/or values are difficult or impossible to interpret. They measure the pair-wise attribute correlations in the tables and use mutual information and entropy to build a dependency graph, which is then explored to find matching node pairs. While such information-theoretic approach has some points in common with our use of entropy, note that +Spicy does not exploit mutual entropy to find matches. Both [21, 55] are orthogonal to +Spicy, which can be used to verify the outcome of the formers.

SemInt [58] clusters similar attributes by using data patterns and catalog information as inputs to neural network. The entire approach is automated, yet a set of parsers need to be instructed at the beginning of a matching task.

LSD [40] adopts a machine learning approach: a set of learners must be trained by feeding examples of mappings among a smaller set of sources. Then, the accumulated knowledge is reused to automatically derive mappings between other sources in the same domain. The instance-based module in +Spicy is based on electrical circuits and does not require additional input, nor training in order to verify the mappings.

Finally, the idea of using electricity to address computer science problems has also been exploited in other cases. One example are graphs random walks [41, 68]. In [10], the author builds an Hex-playing machine: Hex is a two-player game that aims at connecting two opposite sides of a rhombic board through continuous black or white cells. In the paper, a two-dimensional electrical charge distribution is associated with any given Hex cell. This machine made decisions based on properties of the corresponding potential field.

Conclusions

In this thesis, based on concepts provided by schema mapping and data exchange research, we studied the notion of quality for schema mappings.

We identified three crucial viewpoint under which such a notion should be studied: (*i*) which solutions a mapping system should materialize and how it should generate them; (*ii*) how to generate transformation rules (for example *tgds*), starting from a minimal and high-level specification of the mapping; (*iii*) finally, how to semi-automatically generate this high-level specification of the mapping.

With respect to the first aspect, we have introduced new algorithms for schema mappings that rely on the theoretical foundations of data exchange to generate optimal solutions. From the theoretical viewpoint, it represents a step forward towards answering the following question: "is it possible to compute core solutions by using the chase ?" Moreover, we showed that, despite their intrinsic complexity, core solutions can be computed very efficiently in practical, real-life scenarios by using relational database engines.

About the second point, we extended the expressive power of the mapping system with respect to previous systems, in order to handle any set of dependencies, both generated by the mapping system and provided by the user.

Finally, we presented an original architecture to integrate schema matching and mapping generation and we introduced an algorithm that combines schema matching, mapping generation and mapping verification in order to achieve good scalability and high matching quality.

+Spicy is the first mapping generation system that integrates a feasible implementation of a core computation algorithm into the mapping generation process. Previous mapping algorithms tend to produce non-core solutions of poor quality, and previous core-computation algorithms are actually unfeasible, since even for very small databases they require many hours of computation.

We believe that this represents a concrete advancement towards an explicit notion of quality for schema mapping systems.

In this respect, there are several interesting research problems that are worth studying in order to further bridge the gap between schema mappings and data exchange. A relevant one is the extension of the notion of data exchange setting to nested data. Another one is the revision of existing schema mapping benchmarks in order to incorporate the notion of a core solution, and measure how close to the core is the solution generated by a mapping system.

Appendices

Proofs of the Theorems

Proof of Theorem 3.3.5 *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:*

$$J_0 = \bigcup \textsc{reduce}(\textsc{max-informative}(\textsc{max-compact}(\mathcal{W}^{<I,J>}))) \tag{A.1}$$

Then, J_0 is the core of J.

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

Proposition A.1 *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 $\mathcal{W}^{<I,J>}_{m,\overline{a}}$ is closed under isomorphisms.*

*P*roof: Consider a witness block $w \in \mathcal{W}^{<I,J>}_{m,\overline{a}}$, 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 $w' \in \mathcal{W}^{<I,J>}_{m,\overline{a}}$. Since $w \in \mathcal{W}^{<I,J>}_{m,\overline{a}}$, 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. $\diamond$

Consider instance J_0 defined according to Equation A.1. It can be seen that the witness blocks of J_0 fall in two categories: beside "maximal" 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 *maximal* any witness block that is not induced.

Proposition A.2 *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 maximal witness block, w' is an induced witness block and there exists an homomorphism $h : w \to w'$.*

*P*roof: 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 maximal. But this means that h is also an homomorphism of w into w''; moreover, h is proper, since there are atoms in $w'' - w'$ that do not belong to $h(w)$. However, this is not possible by construction of J_0, since by hypothesis w is a maximal witness block, and therefore cannot have proper homomorphisms into other witness blocks of J_0. $\diamond$

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, $\mathcal{W}^{<I,J_0>}_{m,\overline{a}}$, 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 $\mathcal{W}^{<I,J>}_{m,\overline{a}}$ is not empty, since J is a solution; also, it is a finite set, since J is finite. Consider a maximal element w in $\mathcal{W}^{<I,J_0>}_{m,\overline{a}}$ 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 A.1, the equivalence class of witness blocks isomorphic to w, $\mathcal{E}_w$, is included in $\mathcal{W}^{<I,J>}_{m,\overline{a}}$. 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 $\mathcal{W}^{<I,J_0>}_{m,\overline{a}}$, 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 $\mathcal{W} = \{w_i | w_i \in \mathcal{W}^{<I,J>} \textit{ and } w < w_i\}$, 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 $h'(h(w)) = \psi(\overline{a}, h'(h(\overline{b})))$. This proves that $\mathcal{W}_{m,\overline{a}}^{<I,J_0>}$ 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 an homomorphism $h' : J_0 \to J'$. Since we know that J is universal, this amounts to show that J_0 is homomorphically equivalent to J. However, we know that there exists an homomorphism $h : J \to J_0$, since J is universal and J_0 is a solution. We also know that there exists a straightforward homomorphism $h'' : J_0 \to J$, since J_0 is a subset of J. Therefore, J_0 is homomorphically equivalent to J, and 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 an homomorphism $h_\# : J_0 \to J_\#$. This is obviously true for any subset of J_0. Let us consider one of the maximal witness blocks w that belongs to J_0 but not to $J_\#$, i.e., $w \in \mathcal{W}^{<I,J_0>} - \mathcal{W}^{<I,J_\#>}$. Let's call h_w the restriction of $h_\#$ to w, i.e., $h_w : w \to J_\#$. We shall now prove that such an homomorphism cannot exist.

Let's 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 $\mathcal{W}_{m,\overline{a}}^{<I,J_0>}$ 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 $\mathcal{W}_{m,\overline{a}}^{<I,J_0>}$. As a consequence, we know that $\mathcal{W}_{m,\overline{a}}^{<I,J_0>}$ contains two distinct witness blocks, w and $h_w(w)$.

Let us consider the number of nulls in w and $h_w(w)$. There are three possible cases.

(*a*) $|vars(w)| = |vars(h_w(w))|$ – 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 maximal witness block, by construction of J_0; however, it cannot be an induced witness block, either, because of Proposition A.2;

(*b*) $|vars(w)| < |vars(h_w(w))|$ – in this case, it is not possible that $h_w(w)$ is a maximal witness block, since it would not be maximal with respect to the $\prec$ relation; therefore, $h_w(w)$ must be induced; however, this cannot be the case, either, because of Proposition A.2;

(*c*) $|vars(w)| > |vars(h_w(w))|$ – this is clearly not possible, since by hypothesis w is maximal with respect to compacting homomorphisms.

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

Proof of Theorem 3.4.1 *Given a $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st})$, a source instance I, call J a canonical universal solution of Σ_{st} over I. If J is isomorphism–free, consider the following set:*

$$\mathcal{E}^J_{\text{REW-I}} = \bigcup_{\epsilon \in \text{EXPAN}(\mathcal{M})} \{a(\chi^l(\overline{x}_1, \overline{y}_1)) \mid a \text{ s.t. } J \models a(\text{REW-I}_\epsilon(\overline{x}_1, \overline{y}_1))\}$$

then it is the case that:

$$\mathcal{E}^J_{\text{REW-I}} = \text{REDUCE}(\text{MAX-INFORMATIVE}(\text{MAX-COMPACT}(\mathcal{W}^{<I,J>})))$$

*P*roof:

Before getting to the actual proof, 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 1 ***[Image of a Variable]*** *Given a formula homomorphism $h^f : \varphi(\overline{x}, \overline{y}) \rightarrow \varphi'(\overline{x}', \overline{y}')$; 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 A.3 *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 $v' \in \mathcal{V}_{h^f}(v)$.*

*P*roof: 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. ◇

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 2 ***[Compatible Formula Homomorphism]*** *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 an 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 it is the case 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 $h(a(v)) = a'(v')$.*

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.*

Definition 3 ***[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 A.4 *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 an 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, h is a surjection;*
- *if h^f is proper, and a' is invertible, h is proper.*

*P*roof: 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 A.3, we know that this is a well defined mapping. Note also that, by construction, if h is indeed an homomorphism, h^f is compatible with h, a, a'.

We shall now prove that h is an 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:

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

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

On the contrary, assume h^f is proper. Then, 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 A.4 has a dual aspect, as stated in Lemma A.5. 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 4 ***[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, 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$(t)$ as any atom $R_i(\overline{x}_i, \overline{y}_i)$ of $\varphi(\overline{x}, \overline{y})$ such that $t = a(R_i(\overline{x}_i, \overline{y}_i))$. Note that there may be in general more than one atom of this kind, in which case we pick exactly one.*

Lemma A.5 *Given two conjunctive formulas $\varphi(\overline{x}, \overline{y})$, $\varphi'(\overline{x}', \overline{y}')$, suppose there are assignments a, a' such that there exists an 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, h^f is a surjection;*
- *if h is proper, h^f is proper;*

*P*roof: Let's 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 $atom'(h(a(R_i(\overline{x}_i, \overline{y}_i)))) \in \varphi'(\overline{x}', \overline{y}')$;

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

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 the formula $match_{h,a,a'}(R_i(\overline{x}_i, \overline{y}_i))$, called $occ_{A_j}(match_{h,a,a'}(R_i(\overline{x}_i, \overline{y}_i)))$, i.e.:

$$h^f(R_i.A_j : v_k) = occ_{A_j}(match_{h,a,a'}(R_i(\overline{x}_i, \overline{y}_i))) \tag{A.2}$$

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:

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

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

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

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, h^f is a surjection;
- if h is proper, h^f is proper;

This follows immediately from the definition of $atom'$. In fact, assume h is a surjection. In this case, we note that $match_{h,a,a'}$ is obtained by the composition of three surjective mappings – a, h, and $atom'$, and therefore it is itself surjective.

On the contrary, assume h is proper. In this case, $h(a(\varphi(\overline{x}, \overline{y}))$ 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 A.5 has a direct impact on the way in which our rewritings are evaluated, as stated by the following Lemma.

Lemma A.6 *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 an homomorphism: $h : a(\varphi(\overline{x},\overline{y})) \rightarrow a'(\varphi'(\overline{x}',\overline{y}'))$. 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.*

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

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

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 $\textsc{joins}_{h^f}(\overline{x}')$. 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 $\textsc{joins}_{h^f}(a'(\overline{x}'))$ evaluates to true. This proves the claim. ◇

We are now ready to prove the main claim. We introduce the following additional sets:

$$\begin{array}{lcl}
\mathcal{E}^J & = & \bigcup_{\epsilon \in \text{EXPAN}(\mathcal{M})} \{a(\chi^l(\overline{x}_1,\overline{y}_1)) \mid a \text{ s.t. } J \models a(\epsilon(\overline{x}_1,\overline{y}_1))\} \\
\mathcal{E}^J_{\text{REW-C}} & = & \bigcup_{\epsilon \in \text{EXPAN}(\mathcal{M})} \{a(\chi^l(\overline{x}_1,\overline{y}_1)) \mid a \text{ s.t. } J \models a(\textsc{rew-c}_\epsilon(\overline{x}_1,\overline{y}_1))\}
\end{array}$$

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

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

Part 2. $\mathcal{E}^J_{\text{REW-C}} = \text{MAX-COMPACT}(\mathcal{W}^{<I,J>})$

Part 3. $\mathcal{E}^J_{\text{REW-I}} = \text{MAX-INFORMATIVE}(\text{MAX-COMPACT}(\mathcal{W}^{<I,J>}))$

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 $\mathcal{E}^J_{\text{REW-I}}$ is a singleton. Therefore, REDUCE() is the identity mapping on MAX-INFORMATIVE(MAX-COMPACT($\mathcal{W}^{<I,J>}$)), 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 $w_e \in \mathcal{E}^J$. We need to show that w_e 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 $w_e \in \mathcal{W}^{<I,J>}_{m,\overline{a}_w}$. 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_e 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_e belongs to some $\mathcal{E}^J_m$, 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., $J \models a_1(\chi^l(\overline{x}_1,\overline{y}_1)) = w_e$; call a_2 the assignment such that $J \models a_2(\psi(\overline{x}_2,\overline{y}_2))$. We 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 A.4 that there exists a surjective homomorphism $h : a_2(\psi(\overline{x}_2,\overline{y}_2)) \rightarrow a_1(\chi(\overline{x}_1,\overline{y}_1))$. Since h is surjective, we have that:

$$w_e = a_1(\chi(\overline{x}_1,\overline{y}_1)) = h(a_2(\psi(\overline{x}_2,\overline{y}_2))) = \psi(h(a_2(\overline{x}_2)), h(a_2(\overline{y}_2)))$$

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

In order to complete the proof that w_e 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 $a_2(\psi(\overline{x}_2, \overline{y}_2))$ is a witness block for m in J, and $J \in \mathsf{USol}_{\mathcal{M}}(I)$, 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_e 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 $m : \forall \overline{x} : \phi(\overline{x}) \rightarrow \exists \overline{y}(\psi(\overline{x}, \overline{y}))$ 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 $\mathcal{E}_m^J$, we know that there exists an assignment a_w such that $w = \psi(a_w(\overline{x}_2), a_w(\overline{y}_2))$. We 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, $\text{PROVENANCE}(t_i)$, 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 A.5, 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 w is a canonical block for $\chi(\overline{x}_1, \overline{y}_1)$, Lemma A.5 holds. Moreover, since h_i is surjective, 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 $J \models a_\chi(\epsilon(\overline{x}_1, \overline{y}_1))$, i.e., $J \models a_\chi(\chi^l(\overline{x}_1, \overline{y}_1)) = w$, and 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.

In order to show that a_b exists, we notice that w has the form $a_w(\psi(\overline{x}_2, \overline{y}_2))$. As a consequence, by definition of a witness block, there exists a canonical

assignment a_b for $\overline{x}_2 \cup \overline{y}_2$ such that $a_b(\overline{x}_2) = a_w(\overline{x}_2)$ and: (a) $I \models \phi(a_b(\overline{x}_2))$; (b) m has been fired with assignment a_b to generate a canonical block $a_b(\psi(\overline{x}_2, \overline{y}_2))$.

From the discussion above, it follows that $J \models a_\chi(\chi^l(\overline{x}_1, \overline{y}_1)) = w$ and that $J \models a_b(\psi(\overline{x}_2, \overline{y}_2))$. In order to conclude the proof, we only need to prove that a_χ, a_b are such that $\text{EQUAL}_{h^f{}_\epsilon}(a_\chi(\overline{x}_1), a_b(\overline{x}_2))$ evaluates to true.

But this follows from Lemma A.6. In order to apply the Lemma, we need to show that there is a formula homomorphism $h : a_b(\psi(\overline{x}_2, \overline{y}_2)) \to a_\chi(\chi^l(\overline{x}_1, \overline{y}_1))$. To see, this, consider that we know that $w = a_\chi(\chi^l(\overline{x}_1, \overline{y}_1)) = a_w(\psi(\overline{x}_2, \overline{y}_2))$. For any variable $v_{2k} \in \overline{x}_2 \cup \overline{y}_2$, let's choose h in such a way that $h(a_b(v_{2k})) = a_w(v_{2k})$. This is a valid homomorphism of $a_b(\psi(\overline{x}_2, \overline{y}_2))$ into $a_w(\psi(\overline{x}_2, \overline{y}_2))$. In fact, h maps atoms in $a_b(\psi(\overline{x}_2, \overline{y}_2))$ into atoms in $a_w(\psi(\overline{x}_2, \overline{y}_2))$ by construction. To prove that it is a valid homomorphism, it remains to show that it maps constants to themselves. To see this, notice that a_b is a canonical assignment. Therefore, it only associates constants with universal variables. Based on this, for every universal variable $x_{2k} \in \overline{x}_2$, it is the case that $h(a_b(x_{2k})) = a_w(x_{2k}) = a_b(x_{2k})$, since we know that $a_b(\overline{x}_2) = a_w(\overline{x}_2)$. We can conclude that h is the identity on constants, and therefore it is a valid homomorphism. Note also that h^f is compatible with h, a_χ, a_b by construction. This means that we can apply Lemma A.6. It follows that $\text{EQUAL}_{h^f{}_\epsilon}(a_\chi(\overline{x}_1), a_b(\overline{x}_2))$ 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. – $\mathcal{E}^J_{\text{REW-C}} = \text{MAX-COMPACT}(\mathcal{W}^{<I,J>})$

Part 2. (first half) – $\mathcal{E}^J_{\text{REW-C}} \subseteq \text{MAX-COMPACT}(\mathcal{W}^{<I,J>})$

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

Call ϵ the expansion such that, for assignment a, $J \models a(\text{REW-C}_\epsilon(\overline{x}_1, \overline{y}_1))$. Suppose there exists a witness block w' such that $w_e \prec w'$. By Part 1 of the proof, we know there exists an expansion ϵ' in $\text{EXPAN}(\mathcal{M})$ and an assignment a' such that $J \models a'(\epsilon'(\overline{x}'_1, \overline{y}'_1)) = w'$.

Since we assume that $w_e \prec w'$, i.e., there is a compacting homomorphism $h' : w_e \to w'$, we know by Lemma A.5 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:

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

We also know that $h^{f'}$ is a surjection, since h' is a surjection. 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 $|\chi'(\overline{x}'_1, \overline{y}'_1)| \leq |\chi(\overline{x}_1, \overline{y}_1)|$ (otherwise $h^{f'}$ would be proper). 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 $|vars(w')| < |vars(w_e)|$. It is possible to see that this may only happen if at least one of the following cases occurs:

(a) h' maps some null $N_i \in vars(w_e)$ to a constant;

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

Let us consider the two cases separately. In case (a), call y_i the existential variable such that $a(y_i) = N_i$. If $h'(a(y_i))$ is a constant, all occurrences of the form $occ_{A_j}(atom'(R_k(\ldots A_j : h'(a(y_i)) \ldots)))$ 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_i are mapped to universal occurrences, and it is the case that $\mathcal{V}_{h^{f'}}(y_i)$ 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 $\mathcal{I}(y_i) \in \mathcal{V}_{y_i}$, 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. 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.

Since 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 REW-C$_\epsilon$ is of the following form:

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

but this in turn implies that it is not possible that $w_e \in \mathcal{E}^J_{\text{REW-C}}$. In fact, it must be the case that $J \not\models a(\text{REW-C}_\epsilon(\overline{x}_1, \overline{y}_1))$. To see this, notice that there are assignments a, a' such that $w_e = a(\chi(\overline{x}_1, \overline{y}_1)) \subseteq J$, $w_e{}' = a'(\chi'(\overline{x}'_1, \overline{y}'_1)) \subseteq J$, and an homomorphism $h'_\epsilon : w_e \rightarrow w_e{}'$.

Therefore, by Lemma A.6, $\text{EQUAL}_{h^{f'}}(a(\overline{x}_1), a'(\overline{x}'_1))$ evaluates to true, and the existentially quantified subformula evaluates to true. This means that we have reached a contradiction, since w_e cannot belong to $\mathcal{E}^J_{\text{REW-C}}$, and the claim is proven.

This proves the first half of Part 2.

Part 2. (second half) – $\text{MAX-COMPACT}(\mathcal{W}^{<I,J>}) \subseteq \mathcal{E}^J_{\text{REW-C}}$

In order to prove the claim, we need to show that, for any witness block $w \in \text{MAX-COMPACT}(\mathcal{W}^{<I,J>})$, it is the case that $w \in \mathcal{E}^J_{\text{REW-C}}$. 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 $J \models a(\text{REW-C}_\epsilon(\overline{x}_1, \overline{y}_1))$.

We shall prove the claim by way of contradiction. More specifically, suppose that $J \not\models a(\text{REW-C}_\epsilon(\overline{x}_1, \overline{y}_1))$. Since $J \models a(\epsilon(\overline{x}_1, \overline{y}_1)) = w$, by definition of REW-C_ϵ, 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 $\text{EQUAL}_{h^{f'}}(a(\overline{x}_1), a'(\overline{x}'_1))$ evaluates to true.

Consider now $w' = a'(\chi'(\overline{x}'_1, \overline{y}'_1))$. Based on Lemma A.4, we know that there exists an homomorphism $h' : w \to w'$. 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:

- at least one labeled null in $R(t)$ or in $R(t')$ is mapped by h' to a constant;
- 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 the first case, 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 the second case, 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 $w \in \text{MAX-COMPACT}(\mathcal{W}^{<I,J>})$, and therefore w is maximal with respect to $\prec$. Therefore, we have proven the claim.

This concludes the proof of Part 2.

Part 3. – $\mathcal{E}^J_{\text{REW-I}} = \text{MAX-INFORMATIVE}(\text{MAX-COMPACT}(\mathcal{W}^{<I,J>}))$

Part 3. (first half) – $\mathcal{E}^J_{\text{REW-I}} \subseteq \text{MAX-INFORMATIVE}(\text{MAX-COMPACT}(\mathcal{W}^{<I,J>}))$

In order to prove the claim, we need to prove that any block of facts $w_e \in \mathcal{E}^J_{\text{REW-I}}$ belongs also to $\text{MAX-INFORMATIVE}(\text{MAX-COMPACT}(\mathcal{W}^{<I,J>}))$. This amounts to prove that w_e is maximal with respect to $<$, i.e., there exists no witness block w' such that there is a proper homomorphism $h' : w_e \to w'$.

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 $J \models a(\text{REW-I}_\epsilon(\overline{x}_1, \overline{y}_1)) = w_e$, for some assignment a. Assume that there exists a witness block w' in $\text{MAX-COMPACT}(\mathcal{W}^{<I,J>})$ such that there exists a proper homomorphism $h' : w_e \to w'$. In this case, we know by Part 2 of the proof that there exists an expansion ϵ' and assignment a' such that $J \models a'(\text{REW-C}'_\epsilon(\overline{x}'_1, \overline{y}'_1))$. Also, by Lemma A.5, 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 REW-I_ϵ has the following form:

$$\text{REW-I}_\epsilon \quad = \quad \text{REW-C}_\epsilon \wedge \neg\exists \overline{x}'_1, \overline{y}'_1 : (\text{REW-C}'_\epsilon \wedge \text{EQUAL}_{h^{f'}}(\overline{x}_1, \overline{x}'_1)) \wedge ...$$

It follows that $J \not\models a(\text{REW-I}_\epsilon(\overline{x}_1, \overline{y}_1))$. In fact, $\text{EQUAL}_{h^{f'}}(a(\overline{x}_1), a'(\overline{x}'_1))$ evaluates to true by Lemma A.6. This means that it is not possible that $w_e \in \mathcal{E}^J_{\text{REW-I}}$, i.e., we have reached a contradiction. This proves the claim.

Part 3. (second half) – $\text{MAX-INFORMATIVE}(\text{MAX-COMPACT}(\mathcal{W}^{<I,J>})) \subseteq \mathcal{E}^J_{\text{REW-I}}$

In order to prove the claim, we need to show that, for any witness block $w \in \text{MAX-INFORMATIVE}(\text{MAX-COMPACT}(\mathcal{W}^{<I,J>}))$, it is the case that $w \in \mathcal{E}^J_{\text{REW-I}}$. 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 $J \models a(\text{REW-I}_\epsilon(\overline{x}_1, \overline{y}_1))$.

Again, this is done by way of contradiction. Assume $J \not\models a(\text{REW-I}_\epsilon(\overline{x}_1, \overline{y}_1))$; 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', $J \models \text{REW-C}_{\epsilon'}(a'(\overline{x}'_1, \overline{y}'_1)$, and a, a' are such that $\text{EQUAL}_{h^{f'}}(a(\overline{x}_1), a'(\overline{x}'_1))$ evaluates to true.

Consider now $w' = a'(\chi'(\overline{x}'_1, \overline{y}'_1))$. Based on Lemma A.4, we know there must be an homomorphism $h' : w \to w'$. We want to show that h' is proper. In order to do this, we first show that a' is invertible.

But any assignment such that $J \models \text{REW-C}_{\epsilon'}(a'(\overline{x}'_1, \overline{y}'_1))$ must be invertible. Suppose, in fact, that a' is not invertible. This means that there exist two different atoms $R_i(\ldots), R_j(\ldots)$ 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 atom $R_j(\ldots)$. Call a'' the restriction of a' to $\overline{x}''_1, \overline{y}''_2$. Notice that $a'(\chi^{'l}(\overline{x}'_1, \overline{y}'_1)) = a''(\chi^{''l}(\overline{x}''_1, \overline{y}''_1)) = w'$.

Based on Lemma A.4, we know that, since there is a surjective homomorphism (the identity), from $a'(\chi^{'l}(\overline{x}'_1, \overline{y}'_1))$ to $a''(\chi^{''l}(\overline{x}''_1, \overline{y}''_1))$, there must be a surjective formula homomorphism $h^{f''}$ of $a'(\chi^{'l}(\overline{x}'_1, \overline{y}'_1))$ into $a''(\chi^{''l}(\overline{x}''_1, \overline{y}''_1))$.

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 $h^f{}_{e'} : \psi^l(\overline{x}_2, \overline{y}_2) \rightarrow \chi^{l'}(\overline{x}_1', \overline{y}_1')$, and therefore there exists a surjection $h^f{}_{e''} : \psi^l(\overline{x}_2, \overline{y}_2) \rightarrow \chi^{''l}(\overline{x}_1'', \overline{y}_1'')$, obtained by the composition of the two surjective formula homomorphisms $h^f{}_{e'}$ and $h^{f''}$.

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, REW-C$'_\epsilon$ has the following form:

$$\text{REW-C}'_\epsilon \quad = \quad \epsilon' \wedge \neg \exists \overline{x}_1'', \overline{y}_1'' : (\epsilon'' \wedge \text{EQUAL}_{h^{f''}}(\overline{x}_1', \overline{x}_1'')) \wedge ...$$

But then, by Lemma A.6, it is not possible that $J \models \text{REW-C}_{\epsilon'}(a'(\overline{x}_1', \overline{y}_1'))$, which contradicts our hypothesis. Therefore, a' must be invertible.

Since a' is invertible, by Lemma A.4, 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. ◇

Proof of Theorem 3.5.4 *Given a $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st})$. Given an isomorphism-invariant skolemization strategy,* SKOL*, $\Sigma^{FO,core}_{\mathcal{M},\text{SKOL}}$ is a core schema mapping for $\mathcal{M}$.*

*P*roof: Recall that $\Sigma^{FO,core}_{\mathcal{M},\text{SKOL}} = \text{REW-EI}(\text{NORMALIZE}(\Sigma^{FO,exp}_{\mathcal{M},\text{SKOL}}))$. In order to prove the claim, we need to show that, given a source instance I, the result of the chase of $\Sigma^{FO,core}_{\mathcal{M},\text{SKOL}}$ over I is the core universal solution for $\mathcal{M}$ over I, J_0, i.e.:

$$J_{chase} = \Sigma^{FO}_{\mathcal{M},\text{SKOL}}(I) \cong J_0$$

In order to prove the claim, 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 $\Sigma^{FO,core}_{\mathcal{M},\text{SKOL}}$, we might generate the core following a two step process. Based on Theorem 3.4.1, 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, $\Sigma^{FO,full}_{\mathcal{M}}$, one for each expansion $\epsilon \in \text{EXPAN}(\mathcal{M})$:

$$\Sigma^{FO,full}_{\mathcal{M}} = \{r^{full}_\epsilon . \forall \overline{x}_1, \overline{y}_1 : \text{REW-I}_\epsilon(\overline{x}_1, \overline{y}_1) \rightarrow \chi(\overline{x}_1, \overline{y}_1) \mid \epsilon \in \text{EXPAN}(\mathcal{M})\}$$

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 $\Sigma^{FO,core}_{\mathcal{M},\text{SKOL}}$ 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 $\Sigma^{FO,core}_{\mathcal{M},\text{SKOL}}$ are the product of a normalization step and of a further rewriting step, according to REW-EI.

We shall therefore apply the same transformations also to $\Sigma^{FO,full}_{\mathcal{M}}$; 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:

$$\Sigma^{FO,norm}_{\mathcal{M}} = \{\text{REW-EI}_r(\overline{x}_1, \overline{y}_1) \mid r \in \text{NORMALIZE}(\text{EXPAN}(\Sigma^{FO,full}_{\mathcal{M}})\}$$

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 A.7 *Given an isomorphism-free solution J, the result of chasing the two sets of rules $\Sigma^{FO,full}_{\mathcal{M}}$ and $\Sigma^{FO,norm}_{\mathcal{M}}$ over J is the same.*

*P*roof: Let's call J_0 the result of chasing $\Sigma^{FO,full}_{\mathcal{M}}$, and J_* the result of chasing $\Sigma^{FO,norm}_{\mathcal{M}}$ over J.

Being $\Sigma^{FO,full}_{\mathcal{M}}$ a set of full dependencies, the normalization procedure generates a set of logically equivalent new dependencies. Since REW-EI 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 $r^{full}_{\epsilon}.\forall \overline{x}_1, \overline{y}_1 : \text{REW-I}_\epsilon(\overline{x}_1, \overline{y}_1) \rightarrow \chi(\overline{x}_1, \overline{y}_1)$ and assignment a.

Consider $a(\text{REW-I}_\epsilon(\overline{x}_1, \overline{y}_1))$. If $\chi^l(\overline{x}_1, \overline{y}_1)$ is normalized, w also belongs to J_*. Assume $\chi^l(\overline{x}_1, \overline{y}_1)$ is not normalized. Then, let's consider each of its connected components. Each component $\varphi_i(\overline{x}_i, \overline{y}_i)$ generates a rule of the form $r^{full}_{\epsilon,i}.\text{REW-I}_\epsilon(\overline{x}_1, \overline{y}_1) \rightarrow \varphi_i(\overline{x}_i, \overline{y}_i)$. We want to prove that all sets of facts corresponding to instances of the connected components, $\bigcup_i \{a(\varphi_i(\overline{x}_i, \overline{y}_i))\}$ belongs to J_*. There are two possible cases:

(*a*) $J \models a(\text{REW-EI}_{r^{full}_{\epsilon,i}}(\overline{x}_1, \overline{y}_1))$, and therefore $a(\varphi_i(\overline{x}_i, \overline{y}_i))$ belongs to J_* as well;

(*b*) $J \not\models a(\text{REW-EI}_{r^{full}_{\epsilon,i}}(\overline{x}_1, \overline{y}_1))$; this means that there must be a different rule $r^{full}_{\epsilon,j}$ 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

$J \models a'(\text{REW-I}_{\epsilon,j}(\overline{x}_1, \overline{y}_1))$ and that $\text{EQUAL}_{hf}(a(\overline{x}_i), a'(\overline{x}'))$ 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 $J \models a'(\text{REW-I}_{\epsilon,j}(\overline{x}_1, \overline{y}_1))$, 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, it also belongs to J_*.

In this case, by Lemma A.4, we know there exists an 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's 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 an 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, $\Sigma_{\mathcal{M},\text{SKOL}}^{FO,core}$, and $\Sigma_{\mathcal{M}}^{FO,norm}$. Based on the semantics of FO-rules, for each expansion ϵ we concentrate on the two queries $Q_{\text{REW-EI}_{\text{REW-SI}_\epsilon}}(I)$ and $Q_{\text{REW-EI}_{\text{REW-I}_\epsilon}}(I)$. We now want to prove the following Lemma.

Lemma A.8 *Consider an expansion $\epsilon \in \text{EXPAN}(\mathcal{M})$, a source instance I and a canonical universal solution $J \in \mathsf{USol}_{\mathcal{M}}(I)$. There is a block of facts of the form*

$$\text{REW-EI}_{\text{REW-I}_\epsilon}(a(\overline{x}_1), b(\overline{y}_1)) \in Q_{\text{REW-EI}_{\text{REW-I}_\epsilon}}(J)$$

if and only if there is a block of facts

$$\text{REW-EI}_{\text{REW-SI}_\epsilon}(a(\overline{x}_1)) \in Q_{\text{REW-EI}_{\text{REW-SI}_\epsilon}}(I)$$

*P*roof: 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

of the form $\text{REW-S}_\epsilon(a(\overline{x}_1))$ exists in I. In fact, recall that

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

Let's 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 $\text{PREM}_{\chi^l(\overline{x}_1, \overline{y}_1)}(a(\overline{x}_1))$ is contained in I.

Consider now the existentially quantified subformula. There, obviously there exists some 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 $\text{REW-S}_\epsilon(a(\overline{x}_1))$ exists in I.

A very similar argument holds for REW-C_ϵ and REW-SC_ϵ, and for REW-I_ϵ and REW-SI_ϵ. Since the two sets of rules are normalized in the same way, the claim also holds for the final rewritings, i.e., $\text{REW-EI}(\text{REW-I}_\epsilon)$ and $\text{REW-EI}(\text{REW-SI}_\epsilon)$. $\diamond$

Based on Lemma A.8, 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:

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

We can show that the two instances are equal up to isomorphisms. In fact, we know that for each rule $r \in \Sigma^{FO,core}_{\mathcal{M},\text{SKOL}}$:

$$r : \text{REW-EI}_{\text{REW-SI}_\epsilon}(\overline{x}_1) \rightarrow \varphi_{\text{SKOL}}(\overline{x})$$

there is a corresponding rule $r^n \in \Sigma^{FO,norm}_{\mathcal{M}}$:

$$r^n \text{REW-EI}_{\text{REW-I}_\epsilon}(\overline{x}_1, \overline{y}_1) \rightarrow \varphi(\overline{x}, \overline{y})$$

According to Lemma A.8, the premise of r is satisfied by I for an assignment a if and only if the premise of r^n 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 $\varphi(a(\overline{x}), b(\overline{y}))$ is fact block in J_0, while $\varphi_{\text{SKOL}}(a(\overline{x}))$ is a block of facts generated by properly assigning values to Skolem terms. However, the two blocks must be isomorphic. In fact, we know that $\varphi(a(\overline{x}), b(\overline{y}))$ is a canonical block, and therefore it has been generated by the standard skolemization strategy over

$\varphi(a(\overline{x}), b(\overline{y}))$. 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. $\diamond$

Proof of Theorem 3.6.1 *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 belongs 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.*

*P*roof: 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 that $k' \leq k$. 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. ◇

Bibliography

[1] Altova MapForce. http://www.altova.com/MapForce.

[2] Microsoft BizTalk Server 2006 R2. http://www.microsoft.com/biztalk.

[3] Stylus Studio 2008, XML Enterprise Suite. http://www.stylusstudio.com.

[4] The Ontology Alignment Evaluation Initiative – 2007. http://oaei.ontologymatching.org/2007/.

[5] Data Engineering Bullettin. Special Issue on Data Transformations, 1999.

[6] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[7] A. Adya, J.A. Blakeley, S. Melnik, S. Muralidhar, and the ADO.NET Team. Anatomy of the ADO.NET entity framework. In *Proc. of ACM SIGMOD*, pages 877–888, 2007.

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

[9] 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.

[10] V. V. Anshelevich. A Hierarchical Approach to Computer Hex. *Artif. Intell.*, 134(1–2):101–120, 2002.

[11] M. Arenas and L. Libkin. XML Data Exchange: Consistency and Query Answering. *J. of the ACM*, 55(2):1–72, 2008.

[12] P. Atzeni, G. Ausiello, C. Batini, and M. Moscarini. Inclusion and equivalence between relational database schemata. *Theor. Comput. Sci.*, 19:267–285, 1982.

[13] P. Atzeni, P. Cappellari, and P.A. Bernstein. Model-Independent Schema and Data Translations. In *Proc. of EDBT*, 2006.

[14] P. Atzeni and R. Torlone. Management of multiple models in an extensible database design tool. In *Proc. of EDBT*, pages 79–95, London, UK, 1996. Springer-Verlag.

[15] D. Aumueller, H. Do, Massmann S., and E. Rahm. Schema and Ontology Matching with COMA++. In *Proc. of ACM SIGMOD*, pages 906–908, 2005.

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

[17] P. A. Bernstein, A. Y. Halevy, and R. A. Pottinger. A Vision for Management of Complex Models. *ACM SIGMOD Record*, 29(4):55–63, 2000.

[18] P. A. Bernstein and S. Melnik. Model Management 2.0: Manipulating Richer Mappings. In *Proc. of ACM SIGMOD*, pages 1–12, 2007.

[19] P.A. Bernstein. Applying Model Management to Classical Metadata Problems. In *Proc. of CIDR*, 2003.

[20] P.A. Bernstein, S. Melnik, and P. Mork. Interactive schema translation with instance-level mappings. In *Proc. of VLDB*, pages 1283–1286. VLDB Endowment, 2005.

[21] A. Bilke and F. Naumann. Schema Matching using Duplicates. In *Proc. of ICDE*, pages 69–80, 2005.

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

[23] A. Bonifati, E. Q. Chang, T. Ho, L. Lakshmanan, and R. Pottinger. HePToX: Marrying XML and Heterogeneity in Your P2P Databases. In *Proc. of VLDB*, pages 1267–1270, 2005.

[24] 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.

[25] S. Bowers and L. Delcambre. The uni-level description: A uniform framework for representing information in multiple data models. In *Proc. of the 22 nd International Conference on Conceptual Modeling (ER 2003*, pages 45–58. Springer-Verlag, 2003.

[26] L. Bravo, W. Fan, and S. Ma. Extending Dependencies with Conditions. In *Proc. of VLDB*, pages 243–254, 2007.

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

[28] D. Calvanese, G. De Giacomo, M. Lenzerini, and R. Rosati. Logical Foundations fo Peer-to-Peer Data Integration. In *Proc. of ACM PODS*, pages 241–251, 2004.

[29] 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.

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

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

[32] P. R. Clayton. *Fundamentals of Electric Circuit Analysis*. John Wiley & Sons, 2001.

[33] S. Cluet, C. Delobel, J. Simeon, and K. Smaga. Your Mediators Need Data Conversion! In *Proc. of ACM SIGMOD*, pages 177–188, 1998.

[34] E. F. Codd. *A Relational Model of Data for Large Shared Data Banks*, volume 13. ACM, New York, NY, USA, 1970.

[35] S. Davidson and A. Kosky. WOL: A Language for Database Transformations and Constraints. In *Proc. of ICDE*, pages 55–65, 1997.

[36] A. Deutsch, A. Nash, and J. Remmel. The chase revisited. In *Proc. of ACM PODS*, pages 149–158, New York, NY, USA, 2008. ACM.

[37] R. Dhamankar, Y. Lee, A. H. Doan, A. Halevy, and P. Domingos. iMAP: Discovering Complex Semantic Matches between Database Schemas. In *Proc. of ACM SIGMOD*, pages 383–394, 2004.

[38] H. H. Do, S. Melnik, and E. Rahm. Comparison of Schema Matching Evaluations. In *Proc. of the 2nd GI Workshop on Web Databases*, pages 221–237, 2002.

[39] H. H. Do and E. Rahm. COMA - A System for Flexible Combination of Schema Matching Approaches. In *Proc. of VLDB*, pages 610–621, 2002.

[40] A. H. Doan, P. Domingos, and A. Halevy. Reconciling Schemas of Disparate Data Sources: A Machine-Learning Approach. In *Proc. of ACM SIGMOD*, pages 509–520, 2001.

[41] P. G. Doyle and J. L. Snell. Random Walks and Electric Networks. In *Proc. of the Mathematical Associations of America*, 1984.

[42] 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.

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

[44] C. Faloutsos. Indexing multimedia databases. In *Proc. of ACM SIGMOD*, page 467, New York, NY, USA, 1995. ACM Press.

[45] M. Friedman, A. Levy, and T. Millstein. Navigational plans for data integration. In *AAAI '99/IAAI '99: Proceedings of the sixteenth national conference on Artificial intelligence and the eleventh Innovative applications of artificial intelligence conference innovative applications of artificial intelligence*, pages 67–73, Menlo Park, CA, USA, 1999. American Association for Artificial Intelligence.

[46] 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.

[47] A. Fuxman, P.G. Kolaitis, R.J. Miller, and W. Tan. Peer data exchange. In *Proc. of ACM PODS*, pages 160–171, New York, NY, USA, 2005. ACM.

[48] A. Gal. Managing Uncertainty in Schema Matching with Top-K Schema Mappings. *J. of Data Semantics*, VI:90–114, 2006.

[49] A. Gal. Why is Schema Matching Tough and What We Can Do About It. *Sigmod Record*, 35(4):2–5, 2006.

[50] A. Gal. The Generation Y of XML Schema Matching (Panel Description). In *Proceedings of XML Database Symposium*, pages 137–139, 2007.

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

[52] A. Y. Halevy. Answering Queries Using Views: A Survey. *VLDB J.*, 10(4):270–294, 2001.

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

[54] R. Hull and R. King. Semantic database modeling: survey, applications, and research issues. *ACM Comp. Surv.*, 19(3):201–260, 1987.

[55] J. Kang and J. F. Naughton. On Schema Matching with Opaque Column Names and Data Values. In *Proc. of ACM SIGMOD*, pages 205–216, 2003.

[56] M. Lenzerini. Data integration: a Theoretical Perspective. In *Proc. of ACM PODS*, pages 233–246, 2002.

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

[58] W. S. Li and C. Clifton. SEMINT: A Tool for Identifying Attribute Correspondences in Heterogeneous Databases using Neural Networks. *Data and Know. Eng.*, 33(1):49–84, 2000.

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

[60] S. Melnik, A. Adya, and P.A. Bernstein. Compiling mappings to bridge applications and databases. In *Proc. of ACM SIGMOD*, pages 461–472, 2007.

[61] S. Melnik, P.A. Bernstein, A. Halevy, and E. Rahm. Supporting executable mappings in model management. In *Proc. of ACM SIGMOD*, pages 167–178, 2005.

[62] S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity Flooding: A Versatile Graph Matching Algorithm and Its Application to Schema Matching. In *Proc. of ICDE*, pages 117–128, 2002.

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

[64] R.J. Miller, Y.E. Ioannidis, and R. Ramakrishnan. The use of information capacity in schema integration and translation. In *Proc. of VLDB*, pages 120–133, San Francisco, CA, USA, 1993. Morgan Kaufmann Publishers Inc.

[65] R.J. Miller, Y.E. Ioannidis, and R. Ramakrishnan. Schema equivalence in heterogeneous systems: bridging theory and practice. *Inf. Syst.*, 19(1):3–31, 1994.

[66] T. Milo and S. Zohar. Using Schema Matching to Simplify Heterogeneous Data Translation. In *Proc. of VLDB*, pages 122–133, 1998.

[67] F. Naumann, C.-T. Ho, X. Tian, L. M. Haas, and N. Megiddo. Attribute Classification Using Feature Analysis. In *Proc. of ICDE*, page 271, 2002.

[68] C. R. Palmer and C. Faloutsos. Electricity-Based External Similarity of Categorical Attributes. In *Proc. of PAKDD*, pages 486–500, 2003.

[69] R. Pierce. *An Introduction to Information Theory.* Dover Publications, 1980.

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

[71] R. Pottinger and A. Halevy. MiniCon: A Scalable Algorithm for Answering Queries using Views. *VLDB J.*, 10:182–198, 2001.

[72] 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.

[73] E. Rahm and P. A. Bernstein. A Survey of Approaches to Automatic Schema Matching. *VLDB J.*, 10:334–350, 2001.

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

[75] N. C. Shu, B. C. Housel, R. W. Taylor, S. P. Ghosh, and V. Y. Lum. Express: a data extraction, processing, and restructuring system. *ACM Trans. Database Syst.*, 2(2):134–174, 1977.

[76] P. Shvaiko and J. Euzenat. A Survey of Schema Based Matching Approaches. *J. of Data Semantics*, IV - LNCS 3730:146–171, 2005.

[77] W. Su, J. Wang, and F. Lochovsky. Holistic Schema Matching for Web Query Interfaces. In *Proc. of EDBT*, pages 77–94, 2006.

[78] 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.

[79] D.C. Tsichritzis and F.H. Lochovsky. *Data Models*. Prentice Hall Professional Technical Reference, 1982.

[80] J.D. Ullman. Information integration using logical views. *Theor. Comput. Sci.*, 239(2):189–210, 2000.

[81] 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.