The chase procedure and its applications

Adrian Constantin Onet

A Thesis
In the Department
of
Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements
For the Degree of
Doctor of Philosophy(Computer Science)
Concordia University
Montreal, Quebec, Canada

June 2012

Abstract

The chase procedure and its applications

Adrian Constantin Onet, Ph.D.
Concodria University, 2012

The goal of this thesis is not only to introduce and present new chase-based algorithms, but also to investigate the differences between the main existing chase procedures. In order to achieve this, first we will investigate and do a clear delimitation between the existing chase algorithms based on their termination criteria. This will give a better picture of which chase algorithm can be used for different dependency classes. Next, we will investigate the data exchange, data repair and data correspondence problems and show how the chase algorithm can be used to characterize different types of solutions. For the later two problems, we will also investigate the data complexity of *solution-existence* and *solution-check* problems. Further, we will introduce a new chase based algorithm which computes representative solutions under constructible models, a new closed world semantics. This new semantics is, in our view, appropriate to be used as a closed world semantics in data exchange. We will also show that the conditional table computed by this chase algorithm can help to get both possible and certain answers for general queries. And finally, we will investigate strong representation systems and strong data exchange representation system. We will prove, by introducing a new chase based algorithm, that mappings specified by source-to-target second order dependencies and target richly acyclic TGD's are strong data exchange representation systems for the class of first order queries.

I would like to dedicate this thesis to Ralu

Acknowledgements

I would like to acknowledge the important contribution and generous support of the people who helped me achieving this research. I want first to address my sincere thanks to my supervisor, Prof. Gösta Grahne, for his stimulating remarks and suggestions and attentive guidance. My special thanks also goes to the following people for the exchanged views and inspiring discussions, in person or by email: Phokion Kolaitis, Sergio Greco, Jorge Pérez, Bruno Marnette, Alin Deutsch, André Hernich, Maurizio Lenzerini, Slawek Staworko, Georg Gottlob. This thesis would not have been possible without the financial support from National Research Council and Concordia University.

Contents

List of Figures

Chapter 1

Introduction

The key notion of this research, the chase, is an algebraic proof procedure that repeatedly applies a series of chase steps in order to repair inconsistent database instances. This general definition needs a more accurate specification. Therefore, it is known that each chase step takes a dependency that is not satisfied by the database instance, a set of tuples that violates the dependency and changes the database instance so that the resulting instance satisfies the dependency for the given set of tuples. Consider, for example, a database instance I containing tuples $\{R(a,b), S(b,b)\}$ and a dependency specified by the following formula $\forall x\ \forall y\ (R(x,y) \rightarrow S(y,x))$ (mentioning that for each tuple $R(x,y)$ there needs to be a corresponding tuple of the form $S(y,x)$). For the tuple $R(a,b)$, the given dependency is not satisfied because there does not exist a corresponding tuple $S(b,a)$ in I as specified by the dependency. In this case, the chase step will simply add the missing tuple $S(b,a)$ to I and the resulting instance will be $I' = \{R(a,b), S(b,b), S(b,a)\}$. It is easy to note that the resulting instance satisfies the dependency for the tuple $R(a,b)$, and that it actually satisfies the dependency for all sets of tuples in I'.

This procedure was originally developed for testing logical implication between sets of embedded dependencies [58]. In their work, Maier, Mendelzon and Sagiv used the chase procedure as a tool able to check if a set of embedded dependencies Σ, specified by a set of *functional dependencies* (FDs) and *join dependencies* (JDs), logically implies a given join dependency ξ. For this, Maier et al. represented ξ as a *tableau* chased with Σ and finally checked if the resulting tableau represents the identity mapping for all instances that satisfy Σ. For a better view of this process, consider the following trivial example with named attributes. Let ξ be the join dependency $*[ABC, BCD]$ and Σ specified by only one join dependency $*[ABC, CD]$. First, ξ is transformed into the following tableau representation:

A	B	C	D
x_1	x_2	x_3	y_1
y_2	x_2	x_3	x_4

where x_1,x_2,x_3 and x_4 are *distinguished variables* and y_1,y_2 are *nondistinguished variables* (note the presence of the repeated variables x_2 and x_3 in the columns B and C respectively). By applying the chase procedure to this tableau with Σ, the following tableau is obtained:

A	B	C	D
x_1	x_2	x_3	y_1
y_2	x_2	x_3	x_4
x_1	x_2	x_3	x_4

The resulting tableau contains the tuple (x_1, x_2, x_3, x_4) only with distinguished variables. This tells us that the tableau has an identity mapping on all instances. Thus, the dependency ξ is logically implied by Σ. This result is useful, for example, if one needs to remove redundant dependencies from a set of join and functional dependencies over a database. Later on, the chase was reformulated for other types of dependencies as well [74]. In [13] a unified treatment was proposed for the implication problem by extending the chase procedure for classes of tuple-generating and equality generating dependencies. These two classes of dependencies were shown to be expressible enough to capture all the previous classes.

The chase procedure was also shown to be useful in determining the equivalence of database states known to satisfy a set of dependencies [64]. For this, Mendelzon used the chase procedure to compute a *weak instance* associated with database state. A weak instance represents all possible *universal relations* that satisfy all integrity constraints and whose projections on the given relation symbols contain each of the relation in the database. Similarly to the logical implication problem for embedded dependencies, the chase proved to be efficient in determining query-equivalence and containment-under-database constraints [6; 50]. Johnson and Klug [50] showed that for two conjunctive queries Q, Q' and Σ, a set of functional and inclusion dependencies, $Q(I) \subseteq Q'(I)$ holds for all instances I satisfying Σ if and only if there exists a homomorphism from $I_{Q'}$ to the instance resulting by applying the chase procedure on I_Q with Σ (where I_Q denotes the instance containing all conjuncts of Q as tuples).

More recently, the chase procedure has gained a lot of attention due to its usefulness in: data integration [15; 53], ontologies [17; 18], inconsistent databases and data repairs [4; 8; 37], data exchange [27], query optimization [24; 63], peer data exchange [11], and data correspondence [37]. In this thesis we will review the chase procedure and its application in data exchange, data repair and data correspondence problems.

1.1 The Application Areas

1.1.1 Data Exchange

Data Exchange is an important concept harking way back to federated and heterogeneous databases. The problem was finally put on a sound formal basis by the fundamental work of Fagin, Kolaitis, Miller and Popa [27]. The data exchange problem can be described as follows: given a "source" database schema, a "target" database schema, a database instance over the source schema and a set of statements describing the relationship between the source schema and the target schema, find an instance on the target schema such that together with the source instance satisfies those statements. In most cases the relationship between the source schema and target schema is specified by a set of logical formulae of a special format called dependencies. In data exchange these dependencies are classified as *source-to-target* dependencies, the ones representing the relationship between the source and the target schema, and *target* dependencies, specifying the set of constraints that needs to be satisfied by the target instance.

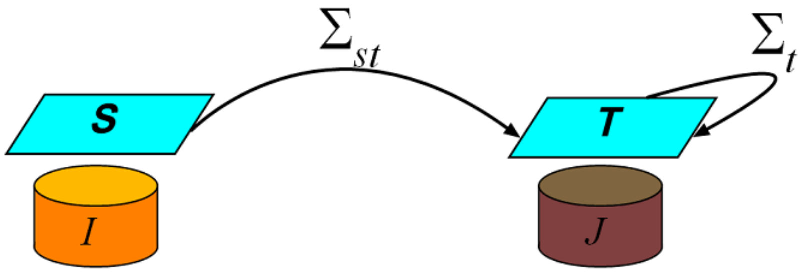

Figure 1.1: Data Exchange

Figure 1.1 is a graphical representation of the data exchange problem, where

S and **T** represent the source and target schemata, the input source instance is represented by I, Σ_{st} - the set of source-to-target dependencies, Σ_t - the set target dependencies, and finally J is the target instance that needs to be computed. Such target instance J is called a *solution* for the data exchange problem. However, the solution J that needs to be found is not guaranteed to be unique. There may exist instances, different than J, that are also solutions for the same problem. Actually, it may be that there are infinitely many such solutions. For example, let us consider the source schema **S** containing a single unary relation symbol *Emp* corresponding to the employees names; and the target schema **T** with a single binary relation symbol *EmpDept* denoting the employees and their department. The relationship between the source and target schemata states that each employee belongs to at least one department. This relationship can be specified by the following logical expression:

$$\forall x\ (Emp(x) \rightarrow \exists y\ EmpDept(x, y))$$

Consider instance $I = \{Emp(john), Emp(mary)\}$. For this setting the following instances over the target schema represent solutions for the given problem:

J_1	J_2	J_3
$EmpDept(john, hr)$	$EmpDept(john, hr)$	$EmpDept(john, hr)$
$EmpDept(mary, qa)$	$EmpDept(mary, hr)$	$EmpDept(mary, qa)$
		$EmpDept(mike, hr)$
		$EmpDept(john, prod)$

On the other hand, the target instance $J' = \{EmpDept(john, hr)\}$ is not a solution since it does not contain the information that *mary* also is part of at least one department. One may observe that if in the instance J_1 we replace the departments with any two other departments, the resulting instance will still be a solution. Abstracting even more, we may replace the departments values in J_1 with two null values X and Y as place holders for the department names. The instance J_4 obtained this way is also a solution of the given problem:

J_4
$EmpDept(john, X)$
$EmpDept(mary, Y)$

The previous instance has the property that, for any solution K, we may construct instance J_K by replacing nulls X and Y in J_4 with some constants or nulls such that $J_K \subseteq K$. This makes, in some aspects, J_4 a more *general solution* than the others and, consequently, it is called a *universal solution*. In [27] it is shown that in order to obtain the certain answers for conjunctive queries over the target database, one have to materialize only a universal solution. Furthermore, in the same paper, it is also noted that the universal solutions are not unique in general, not even up to variable renaming. Even more, there is a data exchange configuration that does not have a universal solution. As it was shown later [23], it is undecidable if there exists a universal solution for a data exchange problem. In case there exist universal solutions, it would be preferable to materialize the *smallest* such universal solution on the target. In [28] it is proved that in case there exist universal solutions for a data exchange problem, then there is a *smallest* universal solution that is unique up to variable renaming. Such a solution was called *the core solution* or simply *the core*. These results are fundamental for our research, since the chase procedure plays an important role in finding universal solutions in data exchange [27]. In [23] it was noted that there is a direct correlation between the chase termination and the existence of the universal solutions (see Section 5).

1.1.2 Data Repair

Data repair is the transformation process applied to an inconsistent database instance such that the resulting instance is consistent and it differs ”minimally” from the initial instance. In this case, the consistency of a database instance is considered against a set of integrity constraints over the database schema. Such constraints may be: key, foreign-key, join, or more generally, any constraints specified by a set of embedded dependencies. Let us denote by Σ the set of constraints. Based on the ”minimality” restrictions used when comparing the repair with the initial instance, we differentiate the following types of repairs:

- *Superset repair.* An instance J is said to be a superset repair of an instance I with respect of Σ, if instance J contains all tuples from I, satisfies all constraints in Σ, and there is no other instance J' properly contained in J with these properties.
- *Subset repair.* An instance J is said to be a subset repair of an instance I with respect of Σ, if instance J contains only tuples from I, satisfies all constraints in Σ, and there is no other instance J' properly containing J with these properties.
- *Symmetric-difference repair.* An instance J is said to be a symmetric-difference repair of an instance I with respect to Σ, if instance J is obtained by adding and removing tuples to/from I, satisfies all constraints in Σ, and there is no other instance J' obtained from a subset of steps used to obtain instance J.

Figure 1.2 gives a visual representation of the aforementioned repair problems, where Σ represents the set of constraints, $\mathbf{S}$ represents the database schema, I the initial instance and instance J represents a superset, subset and symmetric-difference repairs respectively.

A general point of view of the literature introducing the types of repairs is, of course, necessary. The research makes it obvious that data repair represents a major interest

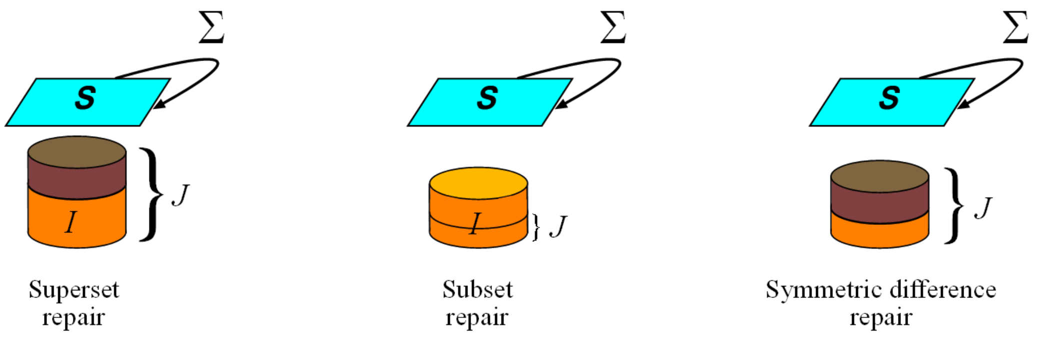

Figure 1.2: Data Repair

in this area. The symmetric-difference repair was introduced by Arenas et al. in [8]. The subset repair as studied by Chomicki and Marcinkowski in [21] requires the repair to be a maximal consistent subset of the inconsistent instance. In [37] the superset repair problem is tackled. Lopatenko and Bertossi, in [57], also consider *cardinality repairs*, where the repair is to be a subset of maximal cardinality. Afrati and Kolaitis, in [4], recently introduced the *component-cardinality repair* which is similar to the cardinality repair except that it considers the cardinality of each relation separately. The data complexity of checking if a given instance is a cardinality repair for a given set of constraints is coNP-hard even for simple sets of constraints. For this reason we will not cover this type of repairs in our thesis. As we will see in section 5.2, the chase process plays an important role in determining the existence and checking if an instance is a subset/superset/symmetric-difference repairs for a given instance under a set of constraints.

To exemplify the difference between the three data repairs mentioned before let us consider the following trivial example. The database schema **S** consisting of two relation names $\{Emp, EmpDept\}$, similarly to the data exchange example, considers also instance the I with tuples $I = \{Emp(ben), Emp(john), Emp(mary), EmpDept(ben, hr)\}$. Consider the constraints on schema **S** specified by the following formula:

$$\forall x\ (Emp(x) \rightarrow \exists y\ EmpDept(x, y)).$$

Clearly I is inconsistent as there are no departments assigned for employee "*john*" and "*mary*". The only subset repair for I under the given constraint is the instance $J^{\mathbf{sub}} = \{Emp(ben), EmpDept(ben, hr)\}$. The three instances below represent a few superset repairs from the infinitely many such repairs. Observe that each of the superset repairs in this example has exactly 6 tuples and that, by removing any of the added tuples, the instance will become inconsistent with respect to the given constraint, while by adding tuples, the repairs are no longer minimal:

$J_1^{\mathbf{sup}}$	$J_2^{\mathbf{sup}}$	$J_3^{\mathbf{sup}}$
$Emp(ben)$	$Emp(ben)$	$Emp(ben)$
$Emp(john)$	$Emp(john)$	$Emp(john)$
$Emp(mary)$	$Emp(mary)$	$Emp(mary)$
$EmpDept(ben, hr)$	$EmpDept(ben, hr)$	$EmpDept(ben, hr)$
$EmpDept(john, hr)$	$EmpDept(john, prod)$	$EmpDept(john, qa)$
$EmpDept(mary, hr)$	$EmpDept(mary, qa)$	$EmpDept(mary, hr)$

Finally, below are represented another three instances which are symmetric-difference repairs for I with the given constraint. Note that any subset and superset repair is also a symmetric-difference repair:

$J_1^{\mathbf{sym}}$	$J_2^{\mathbf{sym}}$	$J_3^{\mathbf{sym}}$
$Emp(ben)$	$Emp(ben)$	$Emp(ben)$
$Emp(mary)$	$Emp(john)$	$Emp(john)$
$EmpDept(ben, hr)$	$EmpDept(ben, hr)$	$Emp(mary)$
$EmpDept(mary, hr)$	$EmpDept(john, prod)$	$EmpDept(ben, hr)$
		$EmpDept(john, qa)$
		$EmpDept(mary, hr)$

1.2 Thesis Structure

Our thesis is structured in two main parts. The first one, covered by chapters 3 and 4, presents the chase algorithm. Based on its applicability and performance in different cases, the chase algorithm comes in different variations. Chapter 3 is dedicated to the reviewing of the most used of these chase variations and to the highlighting of their difference. It is well known that the chase algorithm does not always terminate. Even more, it is also known that testing if the chase algorithm terminates is undecidable in general [23]. In Chapter 4 we revisit the most recent work related to the chase termination and present the main known classes of dependencies for which the *standard-chase* algorithm termination is guaranteed on all instances. In the same chapter we also investigate if these classes of dependencies ensure termination for different chase variations already introduced in Chapter 3.

The second part of the thesis is reserved to the some of the applications of the chase procedure. More precisely, chapter 5 shows why the chase algorithm plays an important role in data exchange, data repair and the newly introduced *data correspondence* problems. This is followed by Chapter 6 where we introduce a chase variation that relies on a new *closed world* semantics, the *constructible models* semantics. This new chase variation helps, in data exchange, to materialize a *conditional table* over the target schema used to get the certain answer for general first order queries (FO) over the target instance compared to the standard chase that materializes an instance over the target schema used only to get the certain answer for union of conjunctive queries (UCQ). In Chapter 7 we extend the standard-chase algorithm for second-order tuple-generating-dependencies under a closed world semantics and show that st-SO dependencies with a target richly acyclic set of TGD's represent a strong data exchange system. Finally, Chapter 8 is allocated for conclusions and further work related to the chase procedure and its applications.

1.3 Contributions

This dissertation covers the results published by the author in [37; 38; 39; 40; 67] (first four being a joint work with prof. Gösta Grahne). In this thesis we review the most prominently used chase variations and make a clear distinction between them based on the followings:

(a) the result computed by the chase and

(b) the chase algorithm termination.

Part of this study was first presented in [67]. It is well known that the problem of deciding if the standard-chase procedure terminates[1] for a given instance is undecidable [23]. In Chapter 4 we show that this undecidability result can be extended to the oblivious-chase algorithm. The chase termination undecidability result motivated the research community to find classes of dependencies that ensure the chase termination [23; 27; 59; 61; 63; 70]. Most of these classes guarantee the chase termination for the standard chase or parallelized standard chase. Unfortunately, none of these classes of dependencies ensure the termination for the oblivious-chase algorithm. In this thesis we extend this set of classes of dependencies by unveiling a class of dependencies that ensures the oblivious-chase termination for all input instances. As it will be shown in Chapter 6, this class of dependencies lays the foundation of the more restrictive class of dependencies ensuring that the newly introduced conditional-chase terminates.

Next we show how the chase procedure plays an important role in data repair by focusing on the following two problems:

1. Solution-Existence: given an instance I and a set of dependencies Σ, *is there an instance J that is a subset repair for I with Σ?*

[1]As we will see in Chapters 3 and 4, the chase termination is defined over two aspects: 1. the chase procedure terminates for any non-deterministic choices made in the algorithm; 2. there exists a series of non-deterministic choices that makes the algorithm to terminate.

2. Solution-Check: given the instances I,J and a the set of dependencies Σ, *is J a superset/subset/symmetric-difference repair for I with* Σ?

We evaluate these problems when Σ is specified by a set of weakly acyclic tuple generating dependencies, a class of dependencies that ensures the standard-chase algorithm termination for any input instance. For this class of dependencies, we show that the subset repair existence problem can be solved in polynomial time. On the other hand, testing if an instance is a subset or symmetric-difference solution was proved [4] to be an NP-complete problem (data complexity) for the same class of dependencies. More recently, this work was also extended by computing consistent answers over the data repairs [72]. In this thesis we extend the data complexity results by showing that the NP-completeness result also holds for the superset repair. Next, we introduce a new large class of dependencies, called *semi-LAV*, that properly contains both *full* and weakly acyclic LAV[1] dependencies. For this new class of dependencies we show that there exists a polynomial time algorithm able to decide all the previously presented problems.

In Chapter 5 we introduce and investigate a new problem with large practical implications, the *data correspondence* problem. Part of this work was first presented in [37]. Data correspondence is a generalization of data exchange [27] and peer data exchange [11], and a special case of the data repair problem. More specifically, data correspondence is the constructive testing between two (or more) database instances in order to verify if they represent the same information. The challenge arises when the two databases may be structured according to different schemata. The prototypical example is to compare a decomposed, normalized instance to the initial universal instance.

As an example, consider the following concrete scenario from a virtual financial brokerage firm where the employees enter their working hours into a database with the

[1]We allow duplicated variables in the body for LAV dependencies.

following schema:

$$\begin{aligned}&EmpHours(\mathit{EmpId}, \mathit{ProjId}, \mathit{TotHours})\\&HourlyRate(\mathit{EmpId}, \mathit{Rate})\\&Sponsor(\mathit{MgrId}, \mathit{ProjId})\\&ExpensePlan(\mathit{PlanId}, \mathit{Rate})\end{aligned}$$

The relation *EmpHours* specifies that an employee with $EmpId$ worked on the project $ProjId$ for a total of $TotHours$. Then, the relation *HourlyRate* records the hourly salary of the employee on a project. We have a tuple associated with each project in the relation *Sponsor*, with the meaning that the project $ProjId$ is sponsored from the funds of the manager with $MgrId$. The relation *ExpensePlan* represents, of course, the hourly rate used for different expense plans. On the other hand, the Managers have to justify their use of funds by entering data in a different database with the schema:

$$Contribution(\mathit{MgrId}, \mathit{EmpId}, \mathit{TotHours}, \mathit{Rate})$$

meaning that the manager with $MgrId$ has paid the employee with *EmpId* for $TotHours$ hours at the rate of *Rate*. In order to verify that funds have been appropriately dispersed, the company relies on the constraints specified by the following logical expressions:

$$\begin{aligned}&\forall ei\ \forall pi\ \forall th\ \forall mi\ \forall r\ (EmpHours(ei, pi, th) \wedge Sponsor(mi, pi) \wedge HourlyRate(ei, r)\\&\qquad\qquad \longrightarrow Contribution(mi, ei, th, r)),\\&\forall mi\ \forall ei\ \forall th\ \forall r\ (Contribution(mi, ei, th, r)\\&\qquad\qquad \longrightarrow \exists pl\ HourlyRate(ei, r) \wedge ExpensePlan(pl, r)).\end{aligned}$$

If the two instances do not satisfy the given set of dependencies, the company needs to bring the two database instances up to date, by entering some missing tuples in the employee database (assuming that the managers financial reports are correct).

Consider, for example, the instance I_1 over the first schema and the instance I_2 over the second schema, described by the following tuples:

I_1	I_2
$EmpHours(john, "issuers", 100)$	$Contribution(rico, john, 100, 20\$)$
$EmpHours(ben, "issuers", 50)$	$Contribution(rico, ben, 50, 25\$)$
$EmpHours(anne, "capital", 50)$	$Contribution(rico, anne, 50, 20\$)$
$HourlyRate(john, 20\$)$	
$HourlyRate(ben, 25\$)$	
$HourlyRate(anne, 25\$)$	
$Sponsor(rico, "issuers")$	
$Sponsor(rico, "capital")$	
$ExpensePlan("A", 20)$	
$ExpensePlan("B", 30)$	

In this case, for the instances I_1, I_2 to satisfy the given constraints, the followings are a few of the minimal changes that may be applied:

- replace $Contribution(rico, anne, 50, 20\$)$ with $Contribution(rico, anne, 50, 25\$)$ in I_2, in order to satisfy the first formula, and add tuple $ExpensePlan(X, 25)$, where X may be replaced with any constant value, in order for the second formula to be satisfied, or

- remove tuples $EmpHours(anne, "capital", 50)$, $ExpensePlan(X, 25)$ from I_1, and remove tuple $Contribution(rico, anne, 50, 20\$)$ from I_2, or

- remove tuples $HourlyRate(ben, 25\$)$, $HourlyRate(anne, 25\$)$ from I_1 and remove tuples $Contribution(rico, ben, 50, 25\$)$, $Contribution(rico, anne, 50, 20\$)$ from instance I_2.

The resulting instances are called *solutions* for the correspondence problem. In case we allow in data correspondence problem both instances to be changed, we talk about *the uniform-data correspondence problem*, which is graphically depicted in figure 1.3, where

$\mathbf{S_1}$ and $\mathbf{S_2}$ represent the two database schemata; Σ_{12} and Σ_{21} the set of constraints; I_1 and I_2 the initial instances; and I_1', I_2' represent some superset solutions for the uniform-data correspondence problem. Note that the uniform-data correspondence problem can be viewed as a special case of data repair problem [4; 8; 21].

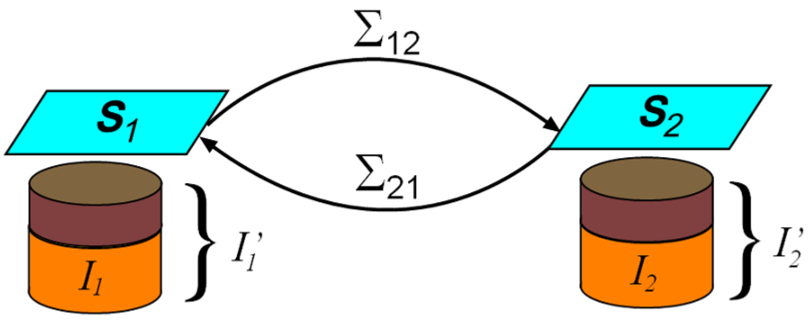

Figure 1.3: Uniform-Data Correspondence

In some cases, one of the sources needs to be authoritative, meaning that it is sound and complete and can not therefore be modified. By considering in the previous example, for the second source to be authoritative, we can apply the following minimal transformation to I_1 (the non-authoritative instance) in order to satisfy the given constraints: remove tuple $HourlyRate(anne, 25\$)$, add tuples $HourlyRate(anne, 20\$)$ and $ExpensePlan(X, 25)$. This correspondence problem is called *the non-uniform-data correspondence problem* and it is graphically depicted in figure 1.4. Even if the non-uniform-data correspondence problem looks similar to the *peer data exchange* problem [11] they are different in the sense that for the non-uniform correspondence problem we are looking for "minimal" solution rather than superset solutions. In the following, by mentioning *data correspondence problem* we refer to both uniform-data and non-uniform-data cases.

Similarly to the data repair problem, for the data correspondence problem we consider the superset, subset and symmetric-difference solutions cases. From the previous

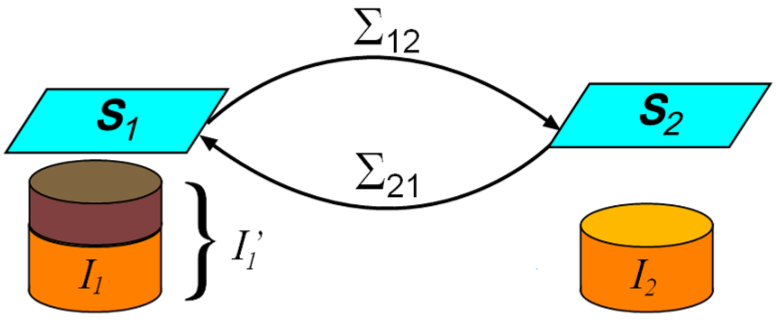

Figure 1.4: Non-Uniform-Data Correspondence

example we observe that there may be several solutions to the correspondence problem; however it may also be the case that there does not exist any solution. We will show in Section 5.3 that the chase procedure plays an important role in checking the existence of solutions and also in testing if a pair of instances is a solution for a data correspondence problem.

Similarly to data repair, we investigate the problem of deciding if there exists a solution and the problem of testing if a given pair of ground instances is a solution for both uniform and non-uniform data correspondence settings. Tables 1.1 and 1.2 summarize the data complexity results found for these problems under weakly acyclic and semi-LAV classes of dependencies. The coNP-completeness data complexity result occurs for the symmetric-difference solution-check for the non-uniform-data correspondence problem under a weakly acyclic set of dependencies. This result is rather surprising considering that the subset and superset solution-check problems are both polynomial under same settings. Also, from the complexity results table, it can be noted that the new class of semi-LAV dependencies is tractable for all previously mentioned problems.

The data exchange problem, as first presented in [27], relies on an open world semantics. Following that, one may get only certain answers for union of conjunctive queries. As we can see in the following example, the open world semantics is not suitable

Table 1.1: The data complexity of solution-existence for the correspondence problem

Existence Σ	unif. corr. sol. weakly acyc.	non-unif. corr. sol. weakly acyc.	non-unif. corr. sol. semi-LAV
subset	P	NPC	P
superset	-	NPC	P
⊕	-	NPC	P

Table 1.2: The data complexity of solution-check for the correspondence problem

Solution-Check Σ	uniform weakly acyc.	uniform semi-LAV	non-uniform weakly acyc.	non-uniform semi-LAV
subset	coNPC	P	P	P
superset	coNPC	P	P	P
⊕	coNPC	P	coNPC	P

for general first order queries. Consider, for example, a data exchange mapping over the source schema $\mathbf{S} = \{EmpCostCenterLocation\}$ and the target schema specified by $\mathbf{T} = \{EmpCostCenter, LocationCostCenter\}$. Consider also the relationship between the source and the target schema specified by the following formula, which normalizes the relation from the source schema into two relations over the target schema:

$$\begin{aligned}\forall ei\ \forall cc\ \forall loc\ \big(&EmpCostCenterLocation(ei, cc, loc)\\ &\rightarrow EmpCostCenter(ei, cc) \wedge LocationCostCenter(cc, loc)\big)\end{aligned} \tag{1.1}$$

Consider now the source instance I with the following tuples:

I
$EmpCostCenterLocation(joe, c1, new\ york)$
$EmpCostCenterLocation(bill, c2, montreal)$
$EmpCostCenterLocation(ann, c1, new\ york)$

Let us now consider the following non-monotone query over the target instance: *Give all employees allocated to a cost center different than New York.* Using notations from [1], this query can be expressed as:

$$ans(ei) \leftarrow EmpCostCenter(ei, cc), \neg LocationCostCenter(cc, \text{``}New\,York\text{``}) \tag{1.2}$$

The certain answer to this query will be empty under the OWA semantics . On the other hand, as each employee may belong to a single cost center and each cost center is uniquely located, we may expect that query 1.2 returns as certain answer "bill". Realizing that the open world semantics had some anomalies, Libkin [54] proposed a first CWA semantics for data exchange; his work was followed by [44; 45; 47; 56]. In this thesis we continue this line of work by proposing a new CWA semantics, called constructible solution semantics, which we argue is a natural fit for the data exchange problem. Intuitively constructible solution semantics only considers those tuples in the target instance that can be *constructed* from the source instance. We will show that the instance based representation is not enough to capture the new CWA semantics and we extend this by using *conditional tables* as representation of the possible solutions. Where the conditional tables are structures that extend tabular instances by associating local conditions for each tuple, such that a tuple exists under some valuation if the local condition is evaluated to **true**. Next, we extend the chase procedure to properly capture this new semantics. To illustrate this, consider the relationship between source and target schema specified by the following source-to-target dependency:

$$\forall ei\ \big(Employee(ei) \rightarrow \exists mi\ EmpMgr(ei, mi)\big) \tag{1.3}$$

and the target dependency:

$$\forall ei\ \big(EmpMgr(ei, ei) \rightarrow SelfMgr(ei)\big) \tag{1.4}$$

The following two instances correspond to the target representation of the source instance $I = \{employee(john), employee(ann)\}$ when chasing I with the standard chase and with new conditional chase respectively:

J
$EmpMgr(john, X)$
$EmpMgr(ann, Y)$

t	$\varphi(t)$
$EmpMgr(john, X)$	**true**
$EmpMgr(ann, Y)$	**true**
$SelfMgr(john)$	$X = john$
$SelfMgr(ann)$	$Y = ann$

As it can be seen, the standard-chase algorithm with the input dependencies 1.3 and 1.4 can not capture the possibility that the employees "john" and "ann" can be their own manager. We introduce a new class of dependencies that ensures the conditional-chase termination for any input instances and then relates the conditional-chase termination to the oblivious-chase termination.

Finally, in the last chapter we focus on the representation systems for data exchange. The notion of *representation systems* describes structures that are algebraically closed under queries. We extend the notion of representation system to encompass data exchange mappings. Seen through this lens, two major classes of representation systems emerge, namely *homomorphic* data exchange systems and *strong* data exchange systems. The monotone systems encompass the classical OWA data exchange semantics, in which reasoning is modulo homomorphic equivalence and only union of conjunctive queries are supported. We develop new technical tools that allow us to prove that there is a class of CWA strong representation systems in which reasoning is modulo isomorphic equivalence. These systems are based on conditional tables and support first-order queries and data exchange mappings specified by a large class of second-order dependencies. We achieve this by showing that under the constructible models closed world semantics, conditional tables are chaseable with the aforementioned class of second-order dependencies.

Chapter 2

Notations and Preliminaries

This chapter covers the notations and notions used in our thesis. Let $\mathbb{N}$ denote the set of natural numbers. We use as placeholders for natural numbers lowercase letters, possibly subscripted and/or superscripted (for example i,k^n,m_j^l). For any two natural numbers n,m with $m \leq n$ let $[m,n]$ be the set of all natural numbers p with $m \leq p \leq n$. The set $[1,n]$ is also denoted by $[n]$. For a set A, by $|A|$ we denote its cardinality, and by $\mathcal{P}(A)$ we identify its powerset, that is all subsets of A. By "$\oplus$" we represent the symmetric difference operator between sets. Thus for two sets A and B by $A \oplus B$ we mean the set $(A \setminus B) \cup (B \setminus A)$. For a given set I, we say that $A \leq_I B$ if $A \oplus I \subseteq B \oplus I$. Clearly $\leq_I$, for a given set I, is a partial order over sets. We say that sets A and B are incomparable, denoted $A \nparallel B$, if $A \nsubseteq B$ and $B \nsubseteq A$. A *tuple* $\bar{a}$ over a set A is a sequence of the form $(a_1, \ldots, a_n)$, with $a_i \in A$ for all $i \in [n]$. The length of a tuple $\bar{a}$, denoted $|\bar{a}|$, is the number of elements in $\bar{a}$. By abusing the notation, we sometimes treat tuples as sets, that is we write $a \in \bar{a}$ to denote that element a is part of the tuple, and we write $\bar{a} \subseteq A$ to denote that all elements in $\bar{a}$ are in set A. Given two sets A, B and the *mapping* $f : A \to B$, we define $Dom(f) = A$ (the domain of f) and $Im(f) = \{f(a) \mid a \in Dom(f)\}$ (the image of f), clearly $Im(f) \subseteq B$. We sometimes specify a mapping f as $\{a_1/f(a_1),\ldots,a_n/f(a_n)\}$, where $\{a_1,\ldots,a_n\} \subseteq \mathit{Dom}(f)$ and for all $a \in \mathit{Dom}(f) \setminus \{a_1,\ldots,a_n\}$ we have $f(a) = a$. A mapping f is said to extend mapping g, denoted $g \Subset f$, if $\mathit{Dom}(g) \subseteq \mathit{Dom}(f)$ and $f(a) = g(a)$ for all $a \in \mathit{Dom}(g)$. For two mappings f and g such that $\mathit{Dom}(f) \cap \mathit{Dom}(g) = \emptyset$, by $f \sqcup g$, we denote the smallest mapping on the domain $\mathit{Dom}(f) \cup \mathit{Dom}(g)$ such that $f \Subset f \sqcup g$ and $g \Subset f \sqcup g$. For a set D by $\mathsf{Id}(D)$ we denote the identity mapping on D, that is $\mathsf{Id}(D)(a) = a$ for all $a \in D$. By Id we denote the set of all identity mappings. Mappings are extended to tuples as follows: let f be a mapping and $\bar{a} = (a_1,\ldots,a_n) \subseteq \mathit{Dom}(f)$. By $f(\bar{a})$ we denote the tuple $(f(a_1),\ldots,f(a_n))$. In the natural way, we extend a mapping f to a set A as follow, $f(A) = \{f(a) \mid a \in A\}$, where $f(a)$ is defined for all $a \in A$ (note that A may be a set of tuples, not only elements from $\mathit{Dom}(f)$).

2.1 Relational Databases

A *database schema* $\mathbf{R}$, or simply *schema*, is a finite set $\{R_1,\ldots,R_n\}$ of relation symbols. For each relation symbol $R \in \mathbf{R}$, we assign a positive integer $\mathit{arity}(R)$ representing the relation arity. Let Δ_{C} and Δ_{N} be two disjoint countable infinite sets of constants and nulls respectively. In this thesis, we will denote the constants by lowercase letters, possibly subscripted/superscripted, from the beginning of the alphabet (eg. a, b_1, $\ldots$). The nulls are denoted by possibly subscripted/superscripted uppercase letters from the end of the alphabet (eg. X, Y_1, $\ldots$). Let $\mathsf{D} = \Delta_{\mathsf{C}} \cup \Delta_{\mathsf{N}}$. A finite instance for a schema $\mathbf{R}$ is a mapping I that assigns for each relational symbol R a finite set of tuples from $\mathsf{D}^{\mathit{arity}(R)}$ (i.e. $I(R) \subseteq \mathsf{D}^{\mathit{arity}(R)}$). In some cases we will also allow instances to assign infinite sets of tuples, in this case we talk about infinite instances. In this dissertation, if not mentioned otherwise, by instance we mean a finite instance. If a tuple $\bar{a}$ belongs to $I(R)$, we say that $R(\bar{a})$ is a *fact* for I. For ease of notation we will also denote by I the set of facts for mapping I. We use two distinct tabular representations to specify instances. For example, instance $I = \{R(a,b,X), R(b,Y,Z), S(c,d)\}$ will be represented in one of the following formats:

I
R(a,b,X)
R(b,Y,Z)
S(c,d,a)

or

R^I		
a	b	X
b	Y	Z

S^I	
c	d

where, by R^I and S^I we denote the sets $I(R)$ and $I(S)$ respectively. The set $Inst(\mathbf{R})$ contains all the instances over schema $\mathbf{R}$. Let $\mathbf{R}_1$, $\mathbf{R}_2$ be two database schemata, such that $\mathbf{R}_2 \subseteq \mathbf{R}_1$, for any instance $I \in Inst(\mathbf{R}_1)$ by $I|_{\mathbf{R}_2}$ we denote the instance over schema $\mathbf{R}_2$ that contains only the facts from I over relational symbols in $\mathbf{R}_2$. For two instances $I, J \in Inst(\mathbf{R})$ we may write $I \subseteq J$ to denote that all the facts in instance I are also in instance J. The set of all elements in an instance I is denoted with $dom(I)$, thus $dom(I)$ will contain all the constants and nulls occurring in I. By $\Delta_{\mathsf{N}}(I)$ we denote the set $\Delta_{\mathsf{N}} \cap dom(I)$. Similarly by $\Delta_{\mathsf{C}}(I)$ we denote the set $\Delta_{\mathsf{C}} \cap dom(I)$. An instance I is called a *ground* (or complete) , if $dom(I) \subseteq \Delta_{\mathsf{C}}$, or equivalently $\Delta_{\mathsf{C}}(I) = dom(I)$. By $Inst^*(R)$ we denote the set of all ground instances over schema $\mathbf{R}$. The presence of null values in an instance makes the instance incomplete. Intuitively, the nulls values in an instance I are placeholders for *unknown* constant values. Sometimes, to make a more clear distinction between a ground instance and an instance that may contain null values we will call the later *tableau*.

Let $I, J \in Inst(\mathbf{R})$, a mapping $h : dom(I) \to dom(J)$ is said be a *homomorphism* between I and J if $h(I) \subseteq J$ and $h(c) = c$ for all constants $c \in \Delta_{\mathsf{C}}(I)$. We write $I \xrightarrow{h} J$ to denote that there exists a homomorphism h between I and J. When the homomorphism is not relevant in the context we simply write $I \to J$. Instances I and J are said to be *homomorphically equivalent*, denoted $I \leftrightarrow J$, if $I \to J$ and $J \to I$. A homomorphism h from I to J is said to be *full* if $h(I) = J$. A full injective homomorphism is called *embedding* . An embedding h from I to J such that h^{-1} is also an embedding from J to I is called *isomorphism*. Two instances I and J are said to be *isomorphically equivalent* if there exists an isomorphism between I and J. This is denoted by $I \cong J$. Note that not all embeddings are isomorphisms. For example, consider the two instances $I = \{R(a, X), R(b, Y)\}$ and $J = \{R(a, Z), R(b, c)\}$. Clearly the embedding $e = \{X/Z, Y/c\}$ is not an isomorphism as $e^{-1}(c) = Z$, that is e^{-1} is not an embedding. A homomorphism $\hat{h}$ is said to be an *extension* of homomorphism h if $h \Subset \hat{h}$.

A homomorphism h from I to I is said to be an *endomorphism* . A *retraction* is a endomorphism h from I to $J = h(I)$ such that h is identity on $dom(J)$, in this case J is called a *retract* of I. An instance J is said to be a *proper retract* of an instance I if J is a retract of I and $J \subsetneq I$. An instance I is said to be a *core* if it does not have any proper retract. An instance J is said to be a *core* of I, if J is a retract of I and it is also a core. The cores of an instance I are unique up to isomorphism [27; 43] and therefore we can talk about *the core* of an instance I and denote it $core(I)$.

2.2 Queries

Consider Δ_{V} to be a countable infinite set of variables, such that it is disjoint of Δ_{C} and Δ_{N}. In this dissertation we will denote the variables with lowercase letters, possibly subscripted/superscripted, from the end of the alphabet (eg. x, z_1, y^2, ...). A *relational atom* , or simply atom, over schema $\mathbf{R}$ is an expression of the form $R(\bar{x})$, where $R \in \mathbf{R}$ and $\bar{x}$ is a tuple from $(\Delta_{\mathsf{C}} \cup \Delta_{\mathsf{V}})^{arity(R)}$. A conjunctive query q over schema $\mathbf{R}$ is an expression of the form:

$$q(\bar{x}) \leftarrow R_1(\bar{x}_1), \ldots, R_n(\bar{x}_n) \tag{2.1}$$

where for all $i \in [n]$ we have $R_i(\bar{x}_i)$ which is an atom over $\mathbf{R}$; the elements in $\bar{x}$ are from the set $\Delta_{\mathsf{C}} \cup \Delta_{\mathsf{V}}$, and all variables in $\bar{x}$ appear in at least one tuple $\bar{x}_i$, $i \in [n]$. The size $|\bar{x}|$ represents the arity of the query q, denoted with $arity(q)$. By CQ we denote the set of all conjunctive queries over any schema. The semantics for a conjunctive query q on a ground instance $I \in Inst^*(R)$ is defined as, where a valuation is a mapping identity on constants and its image is over constants:

$$q(I) = \{v(\bar{x}) \mid v \text{ - valuation, and } v(\bar{x}_i) \in I(R_i),\ \forall i \in [n]\} \tag{2.2}$$

In case $|\bar{x}| = 0$, then we say that q is a *boolean query* . The semantics for a boolean query q on a ground instance I is:

$$q(I) = \begin{cases} \textbf{true} & \text{if } \exists v \text{ - valuation, and } v(\bar{x}_i) \in I(R_i),\ \forall i \in [n] \\ \textbf{false} & \text{otherwise} \end{cases} \tag{2.3}$$

For a conjunctive query $q \in \mathsf{CQ}$, we denote by $db(q)$ the instance containing the facts $R_i(\bar{X}_i)$, $i \in [n]$, where $\bar{X}_i$ is obtained from $\bar{x}_i$ by replacing each variable with a fresh new null from Δ_{N}. Consider the following query:

$$q(x_1, x_3, x_5) \leftarrow R(a, x_1, x_2), S(x_2, x_3), R(x_3, x_4, x_5), T(x_5, b) \tag{2.4}$$

For this query we have $db(q) = \{R(a, X_1, X_2), S(X_2, X_3), R(X_3, X_4, X_5), T(X_5, b)\}$. Recall that the variables are denoted with lowercase letters and the nulls with uppercase letters.

Consider q_1,..., q_n be n conjunctive queries with the same arity. Then a query Q specified as $Q \leftarrow q_1 \vee \ldots \vee q_n$ is called a *union of conjunctive queries* and has the following semantics: $Q(I) = q_1(I) \cup \ldots \cup q_n(I)$. By UCQ we denote the set of all union of conjunctive queries. For a better distinction, in this thesis, we will denote conjunctive queries with lowercase letters and union of conjunctive queries with upper case letters. For a query $Q \in \mathsf{UCQ}$, $Q \leftarrow q_1 \vee \ldots \vee q_n$, we define $db(Q) = \{db(q_1), \ldots, db(q_n)\}$.

By $\mathsf{CQ}^{\neq}$ is denoted the set of all conjunctive queries that also allow the unequality atom. The extension of the previous classes of queries by allowing negated atoms gives $\mathsf{CQ}^{\neg}$,$\mathsf{UCQ}^{\neg}$,$\mathsf{CQ}^{\neg,\neq}$ and $\mathsf{UCQ}^{\neg,\neq}$. The semantics of these types of queries is naturally extended from the semantics of conjunctive queries. A more detailed description of these semantics can be found in [1].

Note that the query semantics previously defined mentions only ground instances, that is complete databases. An *incomplete database* (over a schema **R**) is conceptually a set $\mathcal{I}$ of ground instances (over schema **R**), or *possible worlds* I. Given a query q and an incomplete database $\mathcal{I}$, the result of q on $\mathcal{I}$ is $q(\mathcal{I}) = \{q(I)| \ I \in \mathcal{I}\}$. To this *exact answer* [36] there are two approximations [48], namely:

- *certain answer* evaluation: $certain(q,\mathcal{I}) = \bigcap_{I\in\mathcal{I}} q(I)$, and
- *possible answer* evaluation: $possible(q,\mathcal{I}) = \bigcup_{I\in\mathcal{I}} q(I)$.

In some cases an incomplete database $\mathcal{I}$ over schema **R** can be specified by a single, not necessarily ground, instance I such that

$$\mathcal{I} = \{J \mid v \text{ is a valuation, } v(I) \subseteq J,\ J \in Inst^{*}(R)\}$$

Where a *valuation* v is mapping with domain $\Delta_{\mathsf{C}} \cup \Delta_{\mathsf{N}}$ and image Δ_{C}. In this case, for any query $q \in \mathsf{UCQ}$, the certain answer $certain(q,\mathcal{I})$ can be computed by executing these two steps [27]: a) evaluate q on instance I by treating each null values as a new constant; b) the certain answer result will contain all the tuples from the previous evaluation that contains only constants. Libkin showed in [55] that UCQ is the largest class of queries for which the certain answer can be evaluated in this way. In [1; 35] it is also shown that if $\mathcal{I}$ is specified by instance I, $\mathcal{J}$ is specified by instance J and $I \leftrightarrow J$, then $certain(q,\mathcal{I}) = certain(q,\mathcal{J})$ for any $q \in \mathsf{UCQ}$. Thus, if two incomplete instances are represented by two homomorphically equivalent instances, then the certain answers for any union of conjunctive query will be the same for both incomplete databases.

Given $Q \in \mathsf{UCQ}$ over database schema $\mathbf{R}$ and instance $I \in Inst(\mathbf{R})$ we say that I is a model for Q, denoted $I \vDash Q$, if there exists instance $J \in db(Q)$ such that $J \to I$.

2.3 Dependencies

In most real life applications, database schemata have attached a set of constraints that needs to be satisfied by each instance over the given schema. Some of the most frequent constraints of this type are: primary-key, foreign-key and functional dependency. It is very common to use first order logic as a representation language for database constraints.

In this dissertation we will focus mainly on constraints specified as *embedded dependencies* (for a survey on database dependencies see [31]). An *embedded dependency* over schema $\mathbf{R}$ is a first order sentence ξ of the form:

$$\forall \bar{x}, \forall \bar{y} \left(\alpha(\bar{x}, \bar{y}) \to \exists \bar{z}\, \beta(\bar{x}, \bar{z})\right) \tag{2.5}$$

where all variables in $\bar{x}$ appear both in α and β. The expression α is a conjunction of possible negated relational atoms over $\mathbf{R}$ and unequalities atoms. The expression β is a first order expression over relational atoms over $\mathbf{R}$, unequalities and equalities atoms. We usually refer to the formula given by α as the *body* of the embedded dependency, and the formula given by β as the head of the dependency.

The following subclasses of embedded dependencies play an important role in representing database constraints:

- A *tuple-generating-dependency* (TGD) is an embedded dependency where both the body and the head are formulae logically equivalent with a conjunction of relational atoms;

- An *equality-generating-dependency* (EGD) is an embedded dependency where the body is a formula logically equivalent with a conjunction of relational atoms and the head is represented by an equality atom;

- A *tuple-generating-dependency with disjunctions* (TGD$^{\vee}$) is an embedded dependency where the body is a formula logically equivalent with a conjunction of relational atoms and the head is equivalent with a disjunction of the form $\beta_1 \vee \beta_2 \vee \dots \beta_n$ where, for all $i \in [n]$, β_i is a conjunction of relational atoms;
- A *tuple-generating-dependency with negations* (TGD$^{\neg}$) is a tuple generating dependency where we allow negated relation atoms both in the body and the head;
- A *tuple-generating-dependency with negations and disjunctions* (TGD$^{\vee,\neg}$) is a tuple-generating-dependency with disjunctions where we also allow negated relation atoms in the body and the head.

The following three subclasses of TGD are also of importance in our work:

- A *full* TGD is a TGD without any existentially quantified variables;
- A *local-as-view dependency* (LAV) is a TGD with only one relational atom in the body of the dependency;
- A *true-local-as-view dependency* (LAV*) is a LAV dependency without repeating variables in the body;

Sometimes, when referring to a generic tuple-generating-dependency, we may also use the notation $\alpha \to \beta$; and when referring to an equality-generating-dependency, we use the notation $\alpha \to x = y$. For simplicity we will omit the universal quantifiers; also the conjunction between atoms will be denoted by comma. For example the embedded dependency:

$$\forall x \forall y \forall z \left(R(x,y,z) \wedge S(y,z) \to \exists v \; (R(v,y,z) \wedge S(v,z)) \vee (R(x,y,v) \wedge \neg S(v,z)) \right)$$

will be simply denoted as:

$$R(x,y,z), S(y,z) \rightarrow \exists v\ R(v,y,z), S(v,z) \vee R(x,y,v), \neg S(v,z)$$

Given two distinct schemata $\mathbf{R}_1$ and $\mathbf{R}_2$, a set Σ of TGD's is said to be *source-to-target* TGD's if the relational symbols in the body of each dependency are from schema $\mathbf{R}_1$ and the relational symbols in the head of each dependency are from schema $\mathbf{R}_2$. In this case we say that Σ is a set of source-to-target dependencies over schema $(\mathbf{R}_1, \mathbf{R}_2)$.

For a schema $\mathbf{R}$, by TGD($\mathbf{R}$) we denote the set of all TGD's over schema $\mathbf{R}$. Similarly are defined the dependency class restrictions to a database schema. An instance I is said to satisfy a set of embedded dependencies Σ, denoted $I \vDash \Sigma$, if I satisfies all dependencies in Σ in the standard model theoretic sense.

Finally, given a tuple-generating-dependency ξ of the form $\alpha(\bar{x}, \bar{y}) \rightarrow \exists \bar{z}\ \beta(\bar{x}, \bar{z})$, by *body*($\xi$) we denote the instance containing the tuples obtained by replacing all variables in the relational atoms from the body of the dependency with fresh new null values. Similarly, by *head*(ξ) we denote the instance obtained from the head of the dependency. In case $\xi \in$TGD$^{\vee}$, *head*(ξ) represents a set of instances corresponding to each disjunct in the head of the dependency. For ease of reference, we consider that each variable x from ξ is replaced by null X in *body*(ξ) and *head*(ξ) (that is each variable name from the dependency is kept with the same as null in the instance but upper case, thus the same variable is mapped to the same null both in the body and the head of the dependency).

Chapter 3

The Chase Procedure

3.1 The Chase Procedure

The chase procedure is an iteration a chase steps that either adds a new tuple to satisfy a TGD, either changes the instance to model some equality-generating-dependency, or fails when the instance could not be changed to satisfy an equality-generating-dependency. Depending on when or how the chase step is applied, different chase variations have been considered lately [16; 23; 27; 38; 59; 63]. To differentiate between the variations of the chase procedure, we will call the *the standard chase* the chase procedure considered by Fagin et al [27] for the data exchange problem. Most of the practical constraints in databases can be represented as a set of tuple-generating (TGD) and equality-generating (EGD) dependencies. The first part of this chapter is devoted to present the chase procedure applied on an instance over a set of of TGD's and EGD's. Later on, in Section 3.2, we will also consider the chase under other types of dependencies such as TGD$^{\vee}$, TGD$^{\neg}$ and TGD$^{\vee,\neg}$ (as defined in the preliminaries). For ease of notation, through this section, if not mentioned otherwise, we will use the notation I to represent an arbitrary instance and Σ to represent an arbitrary set of

TGD's and EGD's over the same schema. Also, the database schema will be explicitly mentioned if it is not obvious from the context. Part of this section was also published in [67].

3.1.1 The Chase Step

As stipulated at the beginning of this chapter, the chase procedure is a repetitive application of a chase step. Each chase step "applies" a dependency (in this case TGD or EGD) on a subset of the instance.

3.1.1.1 The TGD Chase Step

Let I be an instance over schema $\mathbf{R}$ and let ξ be the TGD: $\alpha(\bar{x},\bar{y}) \to \exists \bar{z}\ \beta(\bar{y},\bar{z})$ over the same schema $\mathbf{R}$. In this case, it is said that the TGD ξ is *applicable* to instance I with homomorphism h if the following two conditions hold:

1. h is a homomorphism between instances $body(\xi)$ and I, i.e. $body(\xi) \xrightarrow{h} I$ and
2. there is no extension $\hat{h}$ of h such that $head(\xi) \xrightarrow{\hat{h}} I$.

If TGD ξ is applicable to I with h, we say that the pair (ξ, h) is a *standard TGD trigger* for instance I. In addition, if (ξ, h) is a trigger for I, construct an extension $\hat{h}$ of h, such that $\hat{h}(Z) = Z'$, for all nulls $Z \in \bar{Z}$ (recall that each variable x corresponds to null X in $body(\xi)$ and $head(\xi)$), with Z' a fresh new labeled null from Δ_{N}. The instance J obtained as $J = I \cup \hat{h}(head(\xi))$ is called the *result* of applying a *standard TGD chase step* on I with trigger (ξ, h). The notation $I \xrightarrow{(\xi,h)} J$ represents a standard TGD chase step applied on I with trigger (ξ, h).

Example 1 *Consider instance $I = \{R(a,b), R(b,a), S(b,c)\}$ over schema $\mathbf{R} = \{R, S\}$, and consider TGD ξ: $R(x,y), R(y,x) \to \exists z\ S(x,z)$. For this dependency we have $body(\xi) = \{R(X,Y), R(Y,X)\}$ and $head(\xi) = \{S(X,Z)\}$. For these settings there exists* *homomorphism $h = \{X/a, Y/b\}$ that maps body(ξ) to I; and there is no extension of h that maps head(ξ) to I. This makes ξ applicable to I with homomorphism h, yielding the chase step $I \xrightarrow{(\xi,h)} J$, where $J = I \cup \{S(a, Z')\}$ and where the extension $\hat{h}$ of homomorphism h maps Z to Z'. Note that the homomorphism $h_1 = \{X/b, Y/a\}$ together with dependency ξ does not form a trigger for I because there exists $\hat{h}_1$ extension of h_1, namely $\hat{h}_1 = \{X/b, Y/a, Z/c\}$, such that $\hat{h}_1(head(\xi)) \subseteq I$.*

3.1.1.2 The EGD Chase Step

Let I be an instance for schema $\mathbf{R}$, and ξ be the EGD: $\alpha(\bar{x}) \to x_i = x_j$, where $x_i, x_j \in \bar{x}$. The EGD ξ is said to be *applicable* to I with homomorphism h, if the following two conditions hold:

1. *body*$(\xi) \xrightarrow{h} I$ and
2. $h(x_i) \neq h(x_j)$.

Similarly to the TGD case, the pair (ξ, h) is called an *EGD trigger* for I, or simply a trigger for I. For a trigger (ξ, h) over instance I, if h maps both nulls X_i and X_j to constants, then we say that the EGD chase step *fails*, and this is denoted as $I \xrightarrow{(\xi,h)} \bot$. In case the homomorphism h does not map both nulls X_i and X_j to (distinct) constants, then we say that the chase step does not fail. This is denoted with $I \xrightarrow{(\xi,h)} J$, where the instance J is computed as follows:

1. If $h(X_i)$ and $h(X_j)$ are both mapped to labeled nulls, then construct instance J from instance I by replacing either all occurrences of $h(X_i)$ with $h(X_j)$, or all occurrences of $h(X_j)$ with $h(X_i)$.

2. If one of $h(X_i)$, $h(X_j)$ is a constant and the other is a labeled null, then J is constructed by replacing in I all occurrences of the labeled null with the constant.

It can also be noted that the choice in the first condition makes the step nondeterministic, unless we assume an enumeration of the variables.

Example 2 *Let us consider instance $I = \{R(a,b), R(c,Z_1), R(Z_1,Z_2)\}$ and EGD ξ: $R(x,y) \to x = y$. For this dependency we have body(ξ) $= \{R(X,Y)\}$. There are three distinct homomorphisms that map body(ξ) to I: $h_1 = \{X/a, Y/b\}$, $h_2 = \{X/c, Y/Z_1\}$ and $h_3 = \{X/Z_1, Y/Z_2\}$. All these homomorphisms map X and Y to different values, meaning that the EGD ξ is applicable for all. As both $h_1(x)$ and $h_1(y)$ are constants, it follows that $I \xrightarrow{(\xi,h_1)} \bot$. On the other hand, for homomorphism h_2, we have $h_2(X) \in \Delta_{\mathsf{C}}$ and $h_2(Y) \in \Delta_{\mathsf{N}}$. Thus $I \xrightarrow{(\xi,h_2)} J_2$, where $J_2 = \{R(a,b), R(c,c), R(c,Y)\}$ is obtained by replacing all occurrences of X in I with constant c. Finally, h_3 maps both nulls X and Y to distinct labeled nulls, making the EGD ξ applicable on I with homomorphism h_3. Thus $I \xrightarrow{(\xi,h_3)} J_3$, where $J_3 = \{R(a,b), R(c,Y), R(Y,Y)\}$ is obtained from I by replacing X with Y. Another correct EGD chase step would have been $I \xrightarrow{(\xi,h_3)} J_3'$, where $J_3' = \{R(a,b), R(c,X), R(X,X)\}$ is obtained from I by replacing Y with X. Observe that J_3 and J_3' are equivalent up to variable renaming.*

3.1.2 The Chase Algorithm

Using the previously introduced standard-chase steps, we are now ready to present the standard-chase algorithm. This algorithm can be described as an iterative application of the standard-chase steps. In case one of these EGD chase steps fails, then the chase algorithm is said to fail. If the algorithm does not fail due to an EGD, it chooses nondeterministically another trigger and proceeds with the corresponding standard-chase step. The algorithm terminates either when one of the standard EGD chase step fails, or when there are no other EGD or TGD triggers. Figure 3.1 depicts the pseudo-code for the standard-chase algorithm.

```
STANDARD-CHASE($I$,$\Sigma$)
1  $I_0 \leftarrow I$; $i \leftarrow 0$;
2  if exists standard trigger $(\xi, h)$ with $\xi \in \Sigma$ for $I_i$
3      then
4          if $I_i \xrightarrow{(\xi,h)} \perp$
5              then return FAIL
6              else $I_i \xrightarrow{(\xi,h)} I_{i+1}$; $i \leftarrow i + 1$
7      else return $I_i$
8  goto 2
```

Figure 3.1: The standard-chase algorithm

The presented algorithm has a nondeterministic step at line 2 induced by the trigger choice. With this the chase process to be viewed as a tree, also called *chase execution tree*, where level i in the tree represents the i-th step in the chase algorithm, and where to each node a new edge is added for each of the applicable trigger. Each path from the root of the tree to a leaf node represents an *execution branch*, or simply a branch, similarly to a nondeterministic finite automata. Thus the algorithm may return different instances depending on the branch considered. Figure 3.2 depicts a chase execution tree for the algorithm 3.1, where the highlighted edges represent a possible execution branch for the algorithm (note that if selecting trigger (ξ_2^0, h_2^0) in the presented chase tree the algorithm fails).

There are cases when, for some execution branches, the algorithm fails while it does not fail for other execution branches, as it is shown in Example 4. This may happen if one exhaustively chooses triggers for the same dependencies in the nondeterministic step. As we will see in subsection 3.1.3.3, this does not occur when we consider the

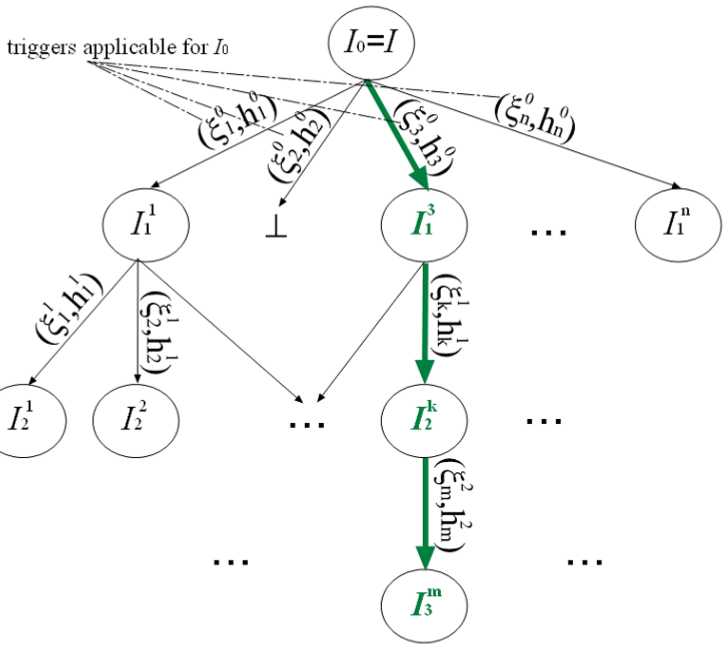

Figure 3.2: A chase execution tree

restricted-chase variation.

Moreover, the standard-chase algorithm stops if either it fails, due to an EGD trigger at step 4, or there are no other triggers to be applied. As the TGD's are adding new tuples to the instance, it may be that the chase algorithm never terminates as shown in example 3.

The following proposition ensures that if the standard-chase algorithm fails on one execution branch, then for any other execution branch for which the algorithm terminates in a finite number of steps, the algorithm will also fail.

Proposition 1 [27] *Let I be an instance and Σ a set of TGD's and EGD's. If for some nondeterministic choice the standard-chase algorithm fails, then it will fail for any nondeterministic choice for which the algorithm terminates in a finite number of steps.*

For each execution branch, for which the algorithm does not fail, define the *chase sequence* associated with that branch as a finite or infinite sequence $(I_0, I_1, I_2, \ldots, I_n, \ldots)$, such that $I_0 = I$ and $I_i \xrightarrow{(\xi,h)} I_{i+1}$, for all $i \geq 0$ and some trigger (ξ, h). In the example depicted in Figure 3.2 the chase sequence associated with the highlighted execution branch has the first four instances: I_0, I_1^3, I_2^k and I_3^m. For ease of notation, in this dissertation, we identify the execution branch by its associated chase sequence. If for some execution branch $(I_0, I_1, I_2, \ldots)$ the algorithm terminates in the finite, then there exists a positive integer n such that there is no trigger for I_n.

As shown in the following example, the standard-chase sequence may be finite or infinite for the same set of TGD's and for the same input instance.

Example 3 *Consider instance $I = \{R(a,b)\}$ and TGD's:*

$$\begin{aligned} \xi_1 &: R(x,y) \rightarrow R(y,x) \\ \xi_2 &: R(x,y) \rightarrow \exists z\, R(y,z) \end{aligned}$$

If in the chase tree we first chose the TGD trigger $(\xi_1, \{X/a, Y/b\})$, the tuple $R(b,a)$ is added to instance I resulting in instance I_1. It can be noticed that, after this choice, the standard-chase step can't be applied on I_1 with ξ_2 and with homomorphism $h = \{X/a, Y/b\}$ as there exists the extension $\hat{h} = \{X/a, Y/b, Z/a\}$ of h, such that $\hat{h}$ maps head(ξ_2) into I_1. Similarly, the standard-chase step can't be applied on I_1 with dependency ξ_1. From this it follows that sequence $(I_0,\ I_1)$, with $I_0 = I$, is a finite standard-chase sequence. On the other hand, if in the algorithm we first choose the trigger $(\xi_2, \{X/a, Y/b\})$, and from there on only chose triggers over dependency ξ_2, the following infinite chase sequence is obtained:

$$\begin{array}{cc} \multicolumn{2}{c}{R^{I_0}} \\ \hline a & b \end{array} \xrightarrow{(\xi_2,h_1)} \begin{array}{cc} \multicolumn{2}{c}{R^{I_1}} \\ \hline a & b \\ b & X_1 \end{array} \xrightarrow{(\xi_2,h_2)} \dots \xrightarrow{(\xi_2,h_n)} \begin{array}{cc} \multicolumn{2}{c}{R^{I_n}} \\ \hline a & b \\ b & X_1 \\ X_1 & X_2 \\ \multicolumn{2}{c}{\dots} \\ X_{n-1} & X_n \end{array} \xrightarrow{(\xi_2,h_{n+1})} \dots$$

The following example shows a case when the standard-chase algorithm fails for some execution branches and does not terminate (implicitly does not fail) for others. Note that this does not contradict Proposition 1 as the non-failing execution branch is infinite.

Example 4 *Consider a slightly changed set of dependencies from the previous example:*

$$\begin{aligned} \xi_1 &: R(x,y) \rightarrow T(y,x) \\ \xi_2 &: T(x,y) \rightarrow x = y \\ \xi_3 &: R(x,y) \rightarrow \exists z\, R(y,z) \end{aligned}$$

Let $I = \{R(a,b)\}$ be an instance. By applying TGD trigger $(\xi_1, \{X/a, Y/b\})$ it will add the tuple $T(b,a)$ to I. Next, if applied EGD trigger $(\xi_2, \{X/a, Y/b\})$, the standard-chase algorithm will fail. On the other hand, if the chosen execution branch would have only used the triggers over ξ_3, the standard-chase algorithm would not have terminated, as shown in the previous example.

The next proposition shows the relationship between all the finite instances resulting by following different finite execution branches of the standard-chase algorithm.

Proposition 2 [27] *Let K and J be two finite instances returned by the standard-chase algorithm on two distinct execution branches, with input I and Σ, then K and J are homomorphically equivalent, that is $K \leftrightarrow J$.*

From this proposition it follows that whatever execution branch we choose in the standard-chase algorithm, if it terminates, the result is indistinguishable using certain answer over union of conjunctive queries. Based on this result, if there exists an execution branch for which the standard-chase algorithm on input I and Σ terminates in the finite and does not fail, then we denote $\mathit{chase}^{\mathbf{std}}_{\Sigma}(I)$ to be one representative from the homomorphic equivalence class of the instances returned by the standard-chase algorithm. If the standard-chase algorithm fails or if it does not terminate in the finite on all execution branches, then we set $\mathit{chase}^{\mathbf{std}}_{\Sigma}(I) = \perp$.

Using a notation similar to [61] we denote by $\mathsf{CT}^{\mathbf{std}}_{\forall\forall}$ the class of all sets of dependencies for which the standard-chase algorithm terminates for all instances on all execution branches. Then, $\mathsf{CT}^{\mathbf{std}}_{\forall\exists}$ symbolizes the class of all sets of dependencies for which the standard-chase algorithm terminates for all instances on at least one execution branch. Given an instance I, define $\mathsf{CT}^{\mathbf{std}}_{I,\forall}$ and $\mathsf{CT}^{\mathbf{std}}_{I,\exists}$ to be the class of sets of dependencies for which the standard-chase algorithm terminates for instance I on all execution branches and on at least one execution branch respectively.

The following theorem, obtained by Fagin, Kolaitis, Miller and Popa, ensures that the finite instances returned by the standard-chase algorithm are actually models for the set of input dependencies and input instance.

Theorem 1 [27] *Let Σ be a set of TGD's and EGD's and I be an instance, then for all instances J returned by the standard-chase algorithm with input I and Σ we have: $J \models \Sigma$ and $I \to J$.*

The previous theorem does not hold in case the standard-chase algorithm does not terminate. Consider, for example, the infinite standard-chase sequence $(I_0, I_1, \ldots, I_n, \ldots)$, from Example 3. It is easy to verify that $I_i \not\models \xi_1$, for any positive i, as the tuple $R(b, a)$ is not added in a finite number of steps to the computed instance. From the previous theorem we get the following corollary:

Corollary 1 *If chase*$^{\mathbf{std}}_{\Sigma}(I) \neq \bot$*, then chase*$^{\mathbf{std}}_{\Sigma}(I) \vDash \Sigma$ *and* $I \to$ *chase*$^{\mathbf{std}}_{\Sigma}(I)$*.*

Deutsch, Nash and Remmel showed in [23] that given I and Σ the problems of testing whether the standard-chase terminates on all execution branches or if it terminates for some execution branches are, in general, undecidable.

Theorem 2 [23] *Let* Σ *be a set of TGD's and* I *an instance. Then:*

1. *It is undecidable if* $\Sigma \in \mathsf{CT}^{\mathbf{std}}_{I,\exists}$*, and*
2. *It is undecidable if* $\Sigma \in \mathsf{CT}^{\mathbf{std}}_{I,\forall}$*.*

The next best hope is to either find classes of dependencies for which it is decidable if the standard-chase algorithm terminates for a given instance on some branches (i.e. data dependent chase termination), or to find classes of dependencies that guarantee the standard-chase termination on all execution branches for all instances (i.e. data independent chase termination). Even if most of the research focused on data independent chase termination (see Chapter 4), there was some work done on data dependent chase termination as well, first by Meier et al. [61] and more recently by Hernich [46], who showed that for guarded dependencies (a class that properly contain LAV dependencies) it is decidable if the core chase (a variation of the standard chase, see Section 3.1.3.4) terminates for a given instance. One such class of dependencies, that ensures the standard-chase termination for all instances, is the full TGD, that is TGD's without existential quantifiers. For full TGD's, it is not only known that the standard chase always terminates, but it is also known that all instances returned by the nondeterministic standard-chase algorithm are identical. In Chapter 4 we review other larger classes of dependencies for which it is known that they belong to either $\mathsf{CT}^{\mathbf{std}}_{\forall\forall}$ or $\mathsf{CT}^{\mathbf{std}}_{\forall\exists}$.

3.1.3 Chase Variations

Since its revival, the standard-chase algorithm proved to have some weak points. One of them represents the complexity of testing if an instance satisfies a TGD, for this one needs to find all the sub-instances that satisfy the body of the dependency and also check if the corresponding head of the dependency satisfies the given instance. Another weak point is that, by using the standard-chase algorithm, we may get two different instances for two distinct execution branches. Even more, as shown in example 3, we may have that one execution branch of the algorithm is finite, thus the algorithm terminates returning an instance, while another execution branch with the same input is not finite, thus the algorithm does not terminate. After the standard chase was proposed as a method of computing "general" solutions in data exchange [27], many variations of the standard-chase algorithm were introduced in the literature [16; 23; 27; 38; 59; 63] . In the remaining part of this chapter we present each of the main chase variations and highlight the differences from the standard chase based on their termination criteria, instances returned by different finite execution branches and complexity of the algorithm implementation.

3.1.3.1 The Oblivious Chase

In this subsection we focus on one of the simplest, based on the complexity of its implementation, variation of the standard chase named the *oblivious chase* (also known as the naive chase). The oblivious-chase algorithm counts on the relaxation of the chase step. The oblivious chase presented here differs from the one described by Cali et al. [16] by not relying on any order of the constants, nulls or tuples. As it will be shown in Proposition 4, this does not affect the instance returned by a terminating chase sequence.

The oblivious-chase step is defined by the removal of the second applicability condition from the standard-chase step (see page 30). We say that a pair (ξ, h) is an *oblivious-chase trigger* for instance I if the homomorphism h maps *body*(ξ) into I. If ξ is a TGD: $\alpha(\bar{x}, \bar{y}) \to \exists \bar{z}\ \beta(\bar{x}, \bar{z})$, and $\hat{h}$ is an extension of h that assigns a new fresh labeled null for each null $Z \in \bar{Z}$, then instance $J = I \cup \hat{h}(head(\xi))$ is said to be the result of *obliviously applying a TGD chase step* on instance I with trigger (ξ, h). This oblivious-chase step is denoted with $I \xrightarrow{*(\xi,h)} J$. Clearly if (ξ, h) is a standard TGD chase trigger for instance I, then (ξ, h) is also an oblivious-chase trigger for I. The converse, on the other hand, does not hold in general. In case of EGD's, the oblivious-chase step is identical with the standard-chase step. In order to avoid confusion, we will denote the oblivious EGD chase step as $I \xrightarrow{*(\xi,h)} J$.

Example 5 *Consider instance $I = \{R(a,b), R(b,a), S(b,c)\}$ and the tuple-generating-dependency ξ: $R(x,y), R(y,x) \to \exists z\ S(x,z)$ as in Example 1. The homomorphism $h = \{X/a, Y/b\}$, maps body(ξ) to I, and there is no extension of h that maps head(ξ) to I. In this case (ξ, h) is both a standard and an oblivious-chase trigger. On the other hand, the homomorphism $h_1 = \{X/b, Y/a\}$ also maps body(ξ) to I, but in this case there exists an extension $\hat{h}_1 = \{X/b, Y/a, Z/c\}$ of h_1, such that $\hat{h}_1$ maps head(ξ) into I. That is (ξ, h_1) is an oblivious-chase trigger but not a standard-chase trigger. The instance J is obtained by applying the oblivious-chase step $I \xrightarrow{*(\xi,h_1)} J$, where $J = I \cup \{S(b, Z')\}$ and Z' is a new fresh labeled null.*

The oblivious-chase algorithm is based on the repeated application of the oblivious-chase step, see Figure 3.3. As mentioned, the oblivious step does not check for the existence of the homomorphism extension, it just blindly applies all dependencies. Moreover, the algorithm applies each oblivious trigger exactly once.

The oblivious-chase sequence (finite and infinite) is defined similarly to the standard-chase sequence. The same way are defined the classes: $\mathsf{CT}^{\mathsf{obl}}_{\forall\forall}$ - containing all sets of dependencies for which the oblivious-chase terminates for all instances on all execution

```
OBLIVIOUS-CHASE($I$,$\Sigma$)
1  $I_0 \leftarrow I$;   $i \leftarrow 0$;
2  if exists oblivious trigger $(\xi, h)$ with $\xi \in \Sigma$ for $I_i$ not applied before
3      then
4            if $I_i \xrightarrow{*(\xi,h)} \bot$
5               then return FAIL
6               else  $I_i \xrightarrow{*(\xi,h)} I_{i+1}$;   $i \leftarrow i+1$
7      else  return $I_i$
8  goto 2
```

Figure 3.3: The oblivious-chase algorithm

branches; $\mathsf{CT}^{\mathbf{obl}}_{\forall\exists}$ - containing all sets of dependencies for which the oblivious chase terminates for all instances on at least one execution branch; $\mathsf{CT}^{\mathbf{obl}}_{I,\forall}$ - containing all sets of dependencies for which the oblivious chase terminates for instance I on all execution branches; and finally $\mathsf{CT}^{\mathbf{obl}}_{I,\exists}$ - containing all sets of dependencies for which the oblivious chase terminates for instance I on at least one execution branch.

The nondeterminism introduced at step 2 of the oblivious-chase algorithm may return different results for two distinct execution branches even with the same input. The following proposition states that, in the oblivious chase case, the notions of termination on all branches and of termination on at least one branch are interchangeable.

Proposition 3 $\mathsf{CT}^{\mathbf{obl}}_{\forall\forall} = \mathsf{CT}^{\mathbf{obl}}_{\forall\exists}$ *and* $\mathsf{CT}^{\mathbf{obl}}_{I,\forall} = \mathsf{CT}^{\mathbf{obl}}_{I,\exists}$ *for any instance* I.

Proof: Let us denote by $Trigg_\Sigma(I)$ the set of all oblivious triggers over instance I and dependencies Σ. With this notation, the statement from the proposition derives directly from the following observation: for the oblivious-chase algorithm, with input I and Σ, at any step i if there exists a terminating execution branch in the oblivious-chase tree, then all the triggers from $Trigg_\Sigma(I_i)$ are applied on that branch. Thus if the branch is terminating all execution branches will terminate as well, as those will apply the same set of triggers, possibly in another order ∎

This result adds a benefit to the oblivious-chase algorithm when compared with the standard-chase algorithm as one may not need to worry that the nondeterministically chosen execution branch does not terminate when another may terminate.

The following proposition extends the previous result by showing that if the oblivious-chase algorithm terminates on one execution branch for input I and Σ, then it will terminate on any execution branch and the instances resulting in this case for any two distinct execution branches are equivalent up to variable renaming. This is a stronger result than the one shown for the standard chase (Proposition 2), since in the standard chase case it may be that two instances returned by two finite execution branches are not isomorphically equivalent even if they are homomorphically equivalent.

Proposition 4 *If the oblivious chase terminates with input I and Σ, then the instances returned by any nondeterministic choice of the execution branch are isomorphically equivalent.*

Proof: Let J_1 and J_2 be two instances returned by running the oblivious-chase algorithm, with input I and Σ, on two distinct execution branches. Let $\mathcal{T}_0$ be the set of oblivious triggers for I. First we need to define the notion of isomorphically equivalent triggers. Two triggers (ξ, h_1) and (ξ, h_2) are said to be *isomorphically equivalent* if there exists a bijection f, identity on constants, such that $h_1 = f \circ h_2$ and $h_2 = f^{-1} \circ h_1$. We denote this relation by $(\xi, h_1) \cong (\xi, h_2)$. Assume that the order of triggers applied by the oblivious-chase algorithm to return instance J_1 is $\gamma_1 = ((\xi_1^1, h_1^1), (\xi_1^2, h_1^2), \dots, (\xi_1^n, h_1^n))$ such that $J_1^{i-1} \xrightarrow{*(\xi_1^i, h_1^i)} J_1^i$ for all $i \in [n]$ and $I = J_1^0$, $J_1^n = J_1$. Similarly, consider sequence $\gamma_2 = ((\xi_2^1, h_2^1), (\xi_2^2, h_2^2), \dots, (\xi_2^m, h_2^m))$ such that $J_2^{i-1} \xrightarrow{*(\xi_2^i, h_2^i)} J_2^i$ for all $i \in [m]$ and $I = J_2^0$, $J_2^m = J_2$. We will inductively construct a sequence of injections $(f_1, f_2, \dots, f_n)$

such that $f_i \Subset f_{i_1}$, for all $i \in [n-1]$, and for all $i \in [n]$ there exists an integer $j \in [m]$ such that $f_i(J_1^i) \subseteq J_2^j$, as follows:

- if $i = 1$, as trigger $(\xi_1^1, h_1^1) \in \mathcal{T}_0$ and $\mathcal{T}_0 \subseteq \gamma_2$, it follows that $(\xi_1^1, h_1^1) \in \gamma_2$. Let j_1 be the position of (ξ_1^1, h_1^1) in γ_2; also let $\hat{h}_1^*$ and $\hat{h}_2^*$ be the extensions of h_1^1 used by the oblivious-chase step on the first and second run of the algorithm. Define $f_1 = \{\hat{h}_1^*(Z_1)/\hat{h}_2^*(Z_1), \hat{h}_1^*(Z_2)/\hat{h}_2^*(Z_2), \dots, \hat{h}_1^*(Z_t)/\hat{h}_2^*(Z_t)\} \sqcup \mathsf{Id}(dom(I))$, where $(z_1, z_2, \dots, z_t)$ is the sequence of existentially quantified variables in ξ^1. Clearly f_1 is an injection as both extensions $\hat{h}_1^*$ and $\hat{h}_2^*$ map the nulls associated with the existentially quantified variables to new fresh nulls. Also, from the definition of the oblivious-chase step, it directly follows that $f_1(J_1^1) \subseteq J_2^{j_1}$;

- if $i \in [2, n]$, then, from the induction assumption and the assumption that the oblivious chase terminates, it follows that trigger $(\xi_1^i, f_{i-1} \circ h_1^i) \in \gamma_2$, and let j_i be the position of $(\xi_1^i, f_{i-1} \circ h_1^i)$ in γ_2. Let $\hat{h}_1^*$ and $\hat{h}_2^*$ be the extensions of homomorphisms h_1^i and $f_{i-1} \circ h_1^i$ respectively used by the oblivious-chase step on the first and second run of the algorithm respectively. Let us now define mapping $f_i = \{\hat{h}_1^*(Z_1)/\hat{h}_2^*(Z_1), \hat{h}_1^*(Z_2)/\hat{h}_2^*(Z_2), \dots, \hat{h}_1^*(Z_t)/\hat{h}_2^*(Z_t)\} \sqcup f_{i-1}$ where $(z_1, z_2, \dots, z_t)$ is the sequence of existentially quantified variables in ξ_1^i. Clearly f_i is an injection as both extensions $\hat{h}_1^*$ and $\hat{h}_2^*$ map the nulls associated with the existentially quantified variables to new fresh nulls. Also from the definition of the oblivious-chase step and the induction assumption, it directly follows that $f_1(J_1^1) \subseteq J_2^{j^*}$, where j^* is the maximum from the set $\{j_1, j_2, \dots, j_i\}$.

From this inductive definition, from the definition of the oblivious-chase algorithm that it applies each trigger only once, and also from the assumption that both γ_1 and γ_2 are finite, it implies that there is a one-to-one correspondence between the triggers from γ_1 and triggers in γ_2. From this it follows that $|\gamma_1| = |\gamma_2|$. This means, from the way f_n was defined, that $f_n(J_1) \subseteq J_2$, even more because $|\gamma_1| = |\gamma_2|$, it means that $f_n(J_1) = J_2$

(otherwise it would mean that there exists at least one trigger in γ_2 that generates the extra tuples in J_2 without a corresponding trigger in γ_1). As f_n is an injection, it also follows that $J_1 = f_n^{-1}(J_1)$. This means that J_1 and J_2 are isomorphically equivalent ■

Similarly to the standard chase case, if $\Sigma \in \mathsf{CT}^{\mathbf{obl}}_{I,\forall}$, then we denote by $chase^{\mathbf{obl}}_{\Sigma}(I)$ one representative instance of the isomorphic equivalence class. If the oblivious chase fails or if it does not terminate, we set $chase^{\mathbf{obl}}_{\Sigma}(I) = \bot$. Clearly if $chase^{\mathbf{obl}}_{\Sigma}(I) \neq \bot$, then $chase^{\mathbf{obl}}_{\Sigma}(I) \vDash \Sigma$.

The next corollary establishes an order between the dependency classes defined for standard and oblivious-chase algorithms termination.

Corollary 2 $\mathsf{CT}^{\mathbf{obl}}_{\forall\forall} \subset \mathsf{CT}^{\mathbf{std}}_{\forall\forall}$ *and* $\mathsf{CT}^{\mathbf{obl}}_{I,\forall} \subset \mathsf{CT}^{\mathbf{std}}_{I,\forall}$ *for any instance* I*.*

Proof: Both statements follow directly from the observation that a standard trigger for an instance I is also an oblivious trigger for I. The strict inclusion for the first statement is proved in 6. For the second statement of the corollary, consider $I = \{R(\bar{a})\}$, then for $\Sigma = \{R(\bar{x}) \to \exists \bar{y}\ R(\bar{y})\}$ the oblivious-chase algorithm will not terminate with I and Σ, but clearly the standard chase will terminate as there is no standard trigger for instance I and set of dependencies Σ ■

As shown in the following example there are sets of dependencies $\Sigma \in \mathsf{CT}^{\mathbf{std}}_{\forall\forall} \setminus \mathsf{CT}^{\mathbf{obl}}_{\forall\forall}$.

Example 6 *Consider* Σ *containing a single TGD* ξ*:* $R(x,y) \to \exists z R(x,z)$*. It can easily be noted that for any instance* I*, the standard-chase algorithm terminates on input* I *and* Σ*. On the other hand, for instance* $I = \{R(a,b)\}$*, the oblivious-chase algorithm creates the following infinite chase sequence:*

$$\begin{array}{cc} \multicolumn{2}{c}{R^{I_0}} \\ \hline a & b \end{array} \xrightarrow{*(\xi,h_1)} \begin{array}{cc} \multicolumn{2}{c}{R^{I_1}} \\ \hline a & b \\ a & X_1 \end{array} \xrightarrow{*(\xi,h_2)} \cdots \xrightarrow{*(\xi,h_n)} \begin{array}{cc} \multicolumn{2}{c}{R^{I_n}} \\ \hline a & b \\ a & X_1 \\ a & X_2 \\ \multicolumn{2}{c}{\cdots} \\ a & X_n \end{array} \xrightarrow{*(\xi,h_{n+1})} \cdots$$

In order to relate termination of the standard and the oblivious chase, we introduce a transformation called *enrichment* that takes a TGD $\xi = \alpha(\bar{x}, \mathbf{y}) \rightarrow \exists \bar{z}\ \beta(\bar{x}, \bar{z})$ over schema $\mathbf{R}$ and converts it into the TGD $\hat{\xi} = \alpha(\bar{x}, \bar{y}) \rightarrow \exists \bar{z}\ \beta(\bar{x}, \bar{z}), H(\bar{x}, \bar{y})$, where H is a new relational symbol that does not appear in $\mathbf{R}$. For a set Σ of TGD's defined on schema $\mathbf{R}$, the transformed set is $\widehat{\Sigma} = \{\hat{\xi} : \xi \in \Sigma\}$. Using the enrichment notion, we can present the relation between the standard and oblivious-chase terminations.

Theorem 3 $\Sigma \in \mathsf{CT}^{\mathbf{obl}}_{\forall\forall}$ *iff* $\widehat{\Sigma} \in \mathsf{CT}^{\mathbf{std}}_{\forall\forall}$.

Proof: Let $\Sigma \in$TGD($\mathbf{R}$), and suppose that there is instance I for which the standard chase with $\widehat{\Sigma}$ does not terminate. This means that there is an infinite standard-chase sequence

$$I = I_0 \xrightarrow{(\hat{\xi}_0,h_0)} I_1 \xrightarrow{(\hat{\xi}_1,h_1)} \ldots\ldots \xrightarrow{(\hat{\xi}_{n-1},h_{n-1})} I_n \xrightarrow{(\hat{\xi}_n,h_n)} \ldots. \tag{3.1}$$

Thus $h_i(body(\hat{\xi}_i)) \subseteq I_i$, for all positive i. Since $body(\hat{\xi}_i) = body(\xi_i)$, we have that

$$I = J_0 \xrightarrow{(\xi_0,h_0)} J_1 \xrightarrow{(\xi_1,h_1)} \ldots\ldots \xrightarrow{(\xi_{n-1},h_{n-1})} J_n \xrightarrow{(\xi_n,h_n)} \ldots. \tag{3.2}$$

where $J_i = I_i|_{\mathbf{R}}$, is an infinite oblivious-chase sequence with Σ on I. From this follows by contraposition that $\Sigma \in \mathsf{CT}^{\mathbf{obl}}_{\forall\forall}$ implies $\widehat{\Sigma} \in \mathsf{CT}^{\mathbf{std}}_{\forall\forall}$.

For the second part, let us suppose that there is an instance I for which the oblivious chase with Σ does not terminate. Then there is an infinite oblivious-chase sequence

$$I = I_0 \xrightarrow{(\xi_0,h_0)} I_1 \xrightarrow{(\xi_1,h_1)} \ldots\ldots \xrightarrow{(\xi_{n-1},h_{n-1})} I_n \xrightarrow{(\xi_n,h_n)} \ldots. \tag{3.3}$$

Let $J_0 = I_0$, and for all $i \in \omega$, let $J_{i+1} = I_{i+1} \cup \{H(h_i(\bar{x}), h_i(\bar{y}))\}$. We claim that

$$I = J_0 \xrightarrow{(\hat{\xi}_0,h_0)} J_1 \xrightarrow{(\hat{\xi}_1,h_1)} \ldots\ldots \xrightarrow{(\hat{\xi}_{n-1},h_{n-1})} J_n \xrightarrow{(\hat{\xi}_n,h_n)} \ldots. \tag{3.4}$$

is an infinite standard-chase sequence with $\widehat{\Sigma}$ and I. Toward a contradiction, suppose it is not. Then there must be an $i \in \omega$, such that the standard-chase step cannot be applied with h_i and $\hat{\xi}_i$ on J_i. Let $\xi_i = \alpha(\bar{x}, \bar{y}) \rightarrow \beta(\bar{x}, \bar{z})$. Then $\hat{\xi}_i = \alpha(\bar{x}, \bar{y}) \rightarrow \beta(\bar{x}, \bar{z}), H(\bar{x}, \bar{y})$. If $(\hat{\xi}_i, h_i)$ is not a trigger for J_i, then there exists an extension $\hat{h}_i$ of h_i such that $\hat{h}_i(body(\hat{\xi}_i)) \subseteq J_i$. Since $\hat{h}_i$ is an extension of h_i, it follows that $\hat{h}_i(\bar{X}) = h_i(\bar{X})$ and $\hat{h}_i(\bar{Y}) = h_i(\bar{Y})$, meaning that $H(h_i(\bar{x}), h_i(\bar{y}))) \in J_i$. Because the facts over H are only introduced by the standard chase, it follows that the homomorphism h_i has already been applied with $\hat{\xi}_i$ earlier in the standard-chase sequence. But then h_i must also have been applied with ξ_i at the same earlier stage in the oblivious-chase sequence. This is a contradiction, since it entails that trigger (ξ_i, h_i) would have been applied twice in the oblivious-chase sequence ■

This characterization of the oblivious-chase termination based on the standard-chase termination was used in [39] to find a sufficient condition for the termination of another chase variation called the conditional chase.

The following corollary shows that the undecidability result from Theorem 2 also holds for the oblivious chase case.

Corollary 3 *Let a I be an instance and Σ a set of TGD's. Then it is undecidable if $\Sigma \in \mathsf{CT}^{\mathbf{obl}}_{I,\forall}$.*

Proof: Toward a contradiction, let us suppose that it is decidable if the oblivious-chase algorithm terminates for a given instance I with dependencies Σ. Thus, one may decide if, for a given instance J, it is that $J = chase^{\mathbf{obl}}_{\Sigma}(I)$, contradicting with the undecidability result given by Theorem 15 in [16] ∎

We close the presentation of the oblivious-chase algorithm by comparing the instances returned by the different execution branches of the oblivious-chase algorithm and the standard-chase algorithm. This result is a particularization of the result presented by Cali et al. in [16].

Theorem 4 [16] *Let I be and instance and let Σ be a set of TGD's and EGD's, such that $chase^{\mathbf{obl}}_{\Sigma}(I) \neq \bot$. Then $chase^{\mathbf{std}}_{\Sigma}(I) \leftrightarrow chase^{\mathbf{obl}}_{\Sigma}(I)$ and $chase^{\mathbf{obl}}_{\Sigma}(I) \vDash \Sigma$.*

3.1.3.2 The Semi-Oblivious Chase

Another variation of the standard-chase algorithm is the semi-oblivious chase described by Marnette [59][1]. In order to be consistent with the other chase algorithms presented, we will now define the semi-oblivious chase without the Skolem terms (used in [59]). Intuitively, the semi-oblivious chase differs from the oblivious one by not distinguishing between two triggers (ξ, h_1) and (ξ, h_2), if h_1 and h_2 differ only on nulls corresponding with universally quantified variables that occur only in the body of ξ.

Before presenting the semi-oblivious-chase algorithm, let us introduce a few notations:

- for a TGD ξ of the form $\alpha(\bar{x}, \bar{y}) \to \exists z\ \beta(\bar{x}, \bar{z})$, we denote X_{ξ} to be the set $\Delta_{\mathsf{N}}(body(\xi)) \cap \Delta_{\mathsf{N}}(head(\xi))$, that is the null values from $body(\xi)$ that also occur in $head(\xi)$;
- for homomorphism h and dependency ξ, we denote by $h|_{\mathsf{X}_{\xi}}$ the homomorphism h restricted to the nulls in X_{ξ}.

[1] In his paper, Marnette called the chase algorithm *Oblivious Skolem Chase*. In order to maintain a uniform description of the chase algorithms, we removed the Skolem terms and renamed the algorithm semi-oblivious.

Figure 3.4 presents the semi-oblivious-chase algorithm. Note that the algorithm uses the oblivious-chase step and not the standard-chase one.

```
SEMI-OBLIVIOUS-CHASE($I$,$\Sigma$)
1 $I_0 \leftarrow I$;  $i \leftarrow 0$;
2 if exists oblivious trigger $(\xi, h)$ for $I_i$, such that
     no trigger $(\xi, h')$ with $h|_{X_\xi} = h'|_{X_\xi}$ was applied before
3    then
4          if $I_i \xrightarrow{*(\xi,h)} \bot$
5             then return FAIL
6             else  $I_i \xrightarrow{*(\xi,h)} I_{i+1}$;   $i \leftarrow i + 1$
7    else  return $I_i$
8 goto 2
```

Figure 3.4: The semi-oblivious-chase algorithm

The following example presents the difference between the presented chase algorithms based on their output instances.

Example 7 *Let Σ contain a single TGD $\xi : R(x,y) \rightarrow \exists z\, T(x,z)$, and consider instance $I = \{R(a,b), R(a,c), R(d,e), T(a,a)\}$. For this configuration there exists only one standard TGD trigger $(\xi, \{X/d, Y/e\})$. By applying this trigger, it will add a new tuple $T(d, X_1)$ to I resulting instance $J^{\mathbf{std}}$. In the semi-oblivious chase, besides generating the tuple $T(d, X_1)$, similarly to the standard chase, it will also apply the oblivious trigger $(\xi, \{X/a, Y/b\})$, generating the tuple $T(a, X_2)$. Still the oblivious trigger $(\xi, \{X/a, Y/c\})$ will not be applied in step 2 of the algorithm 3.4 as the homomorphism restricted to $\{X\}$ was already applied with dependency ξ. The instance returned by the semi-oblivious-chase algorithm is denoted with $J^{\mathbf{sobl}}$. The oblivious-chase algorithm applies all three oblivious triggers, returning instance $J^{\mathbf{obl}}$. Below are the tabular representations of the instances returned, restricted to relation symbol T:*

$J^{\mathbf{std}}$

T	
a	a
d	X_1

$J^{\mathbf{sobl}}$

T	
a	a
d	X_1
a	X_2

$J^{\mathbf{obl}}$

T	
a	a
d	X_1
a	X_2
a	X_3

Similarly to the previous chase algorithm, we define $\mathsf{CT}^{\mathbf{sobl}}_{\forall\forall}$ to be the class of dependency sets for which the semi-oblivious chase terminates for all instances for all execution branches; $\mathsf{CT}^{\mathbf{sobl}}_{\forall\exists}$ - to be the class of dependency sets for which the semi-oblivious chase terminates for all instances on at least one execution branch; $\mathsf{CT}^{\mathbf{sobl}}_{I,\forall}$ - to be the class of dependency sets for which the semi-oblivious chase terminates for instance I on all execution branches; $\mathsf{CT}^{\mathbf{sobl}}_{I,\exists}$ - to be the class of dependency sets for which the semi-oblivious chase terminates for instance I on at least one execution branch.

Similarly to the oblivious chase case, the classes $\mathsf{CT}^{\mathbf{sobl}}_{\forall\forall}$ and $\mathsf{CT}^{\mathbf{sobl}}_{\forall\exists}$ are indistinguishable:

Proposition 5 $\mathsf{CT}^{\mathbf{sobl}}_{\forall\forall} = \mathsf{CT}^{\mathbf{sobl}}_{\forall\exists}$ *and* $\mathsf{CT}^{\mathbf{sobl}}_{I,\forall} = \mathsf{CT}^{\mathbf{sobl}}_{I,\exists}$ *for any instance* I.

Proof: It follows directly from the observation that with the semi-oblivious-chase algorithm at any step i, with $\mathcal{T}_i$ being the set of all oblivious triggers for instance I_i, if there exists a terminating execution branch for the oblivious-chase tree, then for any trigger $(\xi, h) \in \mathcal{T}_i$ there exists a trigger $(\xi, h') \in \mathcal{T}_i$ with $h|_{\mathsf{X}_\xi} = h'|_{\mathsf{X}_\xi}$ such that the trigger (ξ, h') is applied on the terminating branch $\blacksquare$

Proposition 6 *If the semi-oblivious-chase algorithm terminates with input* I *and* Σ, *then any two instance, returned by the semi-oblivious-chase algorithm with input* I *and* Σ *by choosing different execution branches, are isomorphically equivalent.*

Proof: Similar to the proof from Proposition 4, but in this case the trigger being partitioned in equivalence classes such that two triggers (ξ, h), (ξ', h') are considered equivalent if $\xi = \xi'$ and $h|_{\mathsf{X}_\xi} = h'|_{\mathsf{X}_\xi}$ ∎

Let I be an instance over schema $\mathbf{R}$ and Σ a set of TGD's and EGD's over $\mathbf{R}$, if $\Sigma \in \mathsf{CT}^{\mathbf{sobl}}_{I,\forall}$, then by $chase^{\mathbf{sobl}}_{\Sigma}(I)$ we denote one representative of the isomorphic equivalence class mentioned in the previous proposition. In case the semi-oblivious-chase algorithm fails or is non-terminating with input instance I and dependencies Σ, then we set $chase^{\mathbf{sobl}}_{\Sigma}(I) = \bot$.

Similarly to the oblivious chase case, we can find a rewriting of the dependencies such that we can relate the semi-oblivious-chase algorithm termination to the standard-chase algorithm termination. For this, we introduce a transformation, called *semi-enrichment*, that takes a TGD $\xi = \alpha(\bar{x}, \bar{y}) \rightarrow \exists \bar{z}\ \beta(\bar{x}, \bar{z})$ over a schema $\mathbf{R}$ and converts it into the TGD $\tilde{\xi} = \alpha(\bar{x}, \bar{y}) \rightarrow \exists \bar{z}\ \beta(\bar{x}, \bar{z}), H(\bar{x})$, where H is a new relational symbol which does not appear in $\mathbf{R}$. For a set Σ of TGD's defined on schema $\mathbf{R}$, the transformed set is $\tilde{\Sigma} = \{\tilde{\xi} : \xi \in \Sigma\}$. Using the semi-enrichment notion, the relation between the standard and semi-oblivious chase terminations can be presented as follows.

Theorem 5 $\Sigma \in \mathsf{CT}^{\mathbf{sobl}}_{I,\forall}$ *iff* $\tilde{\Sigma} \in \mathsf{CT}^{\mathbf{std}}_{I,\forall}$.

Proof: The reasoning is similar to the one used in Theorem 3 ∎

The homomorphic equivalence between the instances returned by a terminating oblivious chase and the standard chase can be extended for semi-oblivious chase as well.

Theorem 6 *Let I be and instance and let Σ be a set of TGD's and EGD's, such that $chase^{\mathbf{sobl}}_{\Sigma}(I) \neq \bot$. Then $chase^{\mathbf{std}}_{\Sigma}(I) \leftrightarrow chase^{\mathbf{sobl}}_{\Sigma}(I)$ and $chase^{\mathbf{sobl}}_{\Sigma}(I) \models \Sigma$.*

Proof: The direction $chase_\Sigma^{\mathbf{std}}(I) \to chase_\Sigma^{\mathbf{sobl}}(I)$ follows directly from the fact that each standard trigger is also a semi-oblivious trigger. For the other direction, we can observe that the tuples generated by applying a semi-oblivious trigger which is not a standard trigger can be mapped to $chase_\Sigma^{\mathbf{sobl}}(I)$ ∎

Clearly $\mathsf{CT}_{\forall\forall}^{\mathbf{obl}} \subset \mathsf{CT}_{\forall\forall}^{\mathbf{sobl}}$. Also, for any non empty instance I we have $\mathsf{CT}_{I,\forall}^{\mathbf{obl}} \subset \mathsf{CT}_{I,\exists}^{\mathbf{sobl}}$. For this consider a non-empty instance I and the set Σ containing a set of TGD's defined as follows: for each n-ary relation R in the database schema the TGD below occurs in Σ:

$$R(x_1, x_2, \dots, x_{n-1}, x_n) \to \exists y\, R(x_1, x_2, \dots, x_{n-1}, y) \tag{3.5}$$

It is obvious to see that the oblivious-chase algorithm with input I and Σ will not terminate for any non-empty instance I. On the other hand, the semi-oblivious chase will terminate for any instance I and Σ. From this and Theorems 6 and 4, it follows that if $chase_\Sigma^{\mathbf{obl}}(I) \neq \bot$, then $chase_\Sigma^{\mathbf{obl}}(I)$, $chase_\Sigma^{\mathbf{sobl}}(I)$ and $chase_\Sigma^{\mathbf{std}}(I)$ are homomorphically equivalent.

The following proposition relates the termination classes for the three chase algorithms presented.

Proposition 7 $\mathsf{CT}_{\forall\forall}^{\mathbf{obl}} = \mathsf{CT}_{\forall\exists}^{\mathbf{obl}} \subset \mathsf{CT}_{\forall\forall}^{\mathbf{sobl}} = \mathsf{CT}_{\forall\exists}^{\mathbf{sobl}} \subset \mathsf{CT}_{\forall\forall}^{\mathbf{std}} \subset \mathsf{CT}_{\forall\exists}^{\mathbf{std}}$

Proof: To prove that $\mathsf{CT}_{\forall\exists}^{\mathbf{obl}} \subset \mathsf{CT}_{\forall\forall}^{\mathbf{sobl}}$ see Example 6. Example 3 shows a set of dependencies for which for any instance there exists a branch under the standard chase that terminates. Clearly, as shown in the example, not all the branches terminate proving that $\mathsf{CT}_{\forall\forall}^{\mathbf{std}} \subset \mathsf{CT}_{\forall\exists}^{\mathbf{std}}$. Finally, to prove that $\mathsf{CT}_{\forall\forall}^{\mathbf{sobl}} \subset \mathsf{CT}_{\forall\forall}^{\mathbf{std}}$, consider the following set of dependencies $\Sigma = \{\xi_1, \xi_2\}$ where:

$$\xi_1: \quad R(x) \rightarrow \exists z\, S(z), T(z, x) \tag{3.6}$$

$$\xi_2: \quad S(x) \rightarrow \exists w\, R(w), T(x, w) \tag{3.7}$$

Clearly, the semi-oblivious chase on instance $I = \{R\}$ does not terminate. To show that the standard chase terminates for any instance I on Σ, consider I to be an arbitrary instance with $|R^I| = n$ and $|S^I| = m$. We will show that the standard chase on I with Σ will terminate in maximum $n + m$ steps. To prove this, observe that the dependencies are triggered only by tuples over R or S. It can be noted that for each tuple $R(a)$ the dependencies will be triggered at most once by the standard chase. For example, for the tuple $R(a)$ the standard trigger $(\xi_1, \{X/a\})$ will generate the two tuples $S(X_1)$ and $T(X_1, a)$, which will not be applied by the standard-chase algorithm on any dependency as $\{R(a), S(X_1), T(X_1, a)\} \models \Sigma$ ∎

We conclude this subsection with the undecidability result for the semi-oblivious-chase algorithm given by Marnette [59]:

Theorem 7 [59] *Given an instance I and a set of TGD's Σ, it is undecidable if $\Sigma \in \mathsf{CT}^{\mathbf{sobl}}_{I,\forall}$.*

From this theorem and Proposition 5, we can conclude that, given an instance I and a set of TGD's Σ, it is undecidable if $\Sigma \in \mathsf{CT}^{\mathbf{sobl}}_{I,\exists}$ too.

3.1.3.3 The Restricted Chase

As shown in section 3.1.2 the standard-chase algorithm involves a non-deterministic trigger selection that introduces new problems related to the result of the chase algorithm (see Example 3). In this subsection, a variation of the standard chase is described that eliminates the non-deterministic behavior by ordering all the objects involved, thus applying the chase steps in a specific deterministic order. The resulting algorithm is called the restricted chase (introduced by Cali et al. in [16]).

The restricted chase presented here differs slightly from the algorithm described in [16], as in order to fully eliminate the non-deterministic choice we had to order the set of dependencies as well.

First let us consider a total order for the set $\Delta_\mathsf{C} \cup \Delta_\mathsf{N}$, such that all the symbols in Δ_N follow the symbols in Δ_C. Let us also consider a total order for the dependencies in Σ. And let (ξ, h) be a standard-chase trigger for I and Σ. The restricted-chase step behaves like the standard one, with the exception that, when applied to instance I, all the new labeled nulls introduced follow the ones existing in I. Consider I to be the initial instance. Then the *level* of a tuple is defined recursively as follows:

1. all atoms in $I_0 = I$ are considered of level 0.
2. if $I_i \xrightarrow{(\xi,h)} I_{i+1}$, then, if the highest level of an atom in $h(body(\xi))$ is k, all the tuples created by applying this trigger have level $k + 1$.

The notion of level is extended to an instance as follows: $level(I)$ is the highest level of any tuples in I. An instance I is said to lexicographically precede an instance J if, for all symbols a in I, there is a symbol b in J such that a precedes b. If I does not precede J and J does not precede I, then we say that instances I and J are lexicographically incomparable.

We are now ready to define the order $\lessdot$ between the set of triggers applicable for an instance I and for the set of dependencies Σ. A trigger (ξ, h) is said to precede trigger (ξ', h') denoted by $(\xi, h) \lessdot (\xi', h')$, if:

1. $level(h(body(\xi))) < level(h'(body(\xi')))$, or

2. $level(h(body(\xi))) = level(h'(body(\xi')))$ and $h(body(\xi))$ lexicographically precedes instance $h'(body(\xi'))$, or

3. $level(h(body(\xi))) = level(h'(body(\xi')))$ and $h(body(\xi))$, $h'(body(\xi'))$ are lexicographically incomparable and ξ precedes ξ'

Note that $<$ is a total order on the triggers. With the use of this order, Figure 3.5 summarizes the restricted-chase algorithm.

```
RESTRICTED-CHASE($I$,$\Sigma$)
1   $I_0 \leftarrow I$; $i \leftarrow 0$;
2   **if** exists a standard trigger for $I_i$
3       **then**
4           Let $(\xi, h)$ be the smallest "$<$" trigger for $I_i$
5           **if** $I_i \xrightarrow{(\xi,h)} \perp$
6               **then return** FAIL
7               **else** $I_i \xrightarrow{(\xi,h)} I_{i+1}$; $i \leftarrow i + 1$
8       **else**
9           **return** $I_i$
10  **goto** 2
```

Figure 3.5: The restricted-chase algorithm

If the restricted-chase algorithm terminates and does not fail for input I and Σ, we denote by $chase_{\Sigma}^{\mathbf{res}}(I)$ the result of this algorithm. In case the algorithm does not terminate or if it fails, we set $chase_{\Sigma}^{\mathbf{res}}(I) = \perp$. Similarly to the other chase algorithms, we introduce the classes $\mathsf{CT}_{\forall\forall}^{\mathbf{res}}$,$\mathsf{CT}_{\forall\exists}^{\mathbf{res}}$,$\mathsf{CT}_{I,\forall}^{\mathbf{res}}$ and $\mathsf{CT}_{I,\exists}^{\mathbf{res}}$ for an instance I. As the non-deterministic choice was eliminated, it follows that $\mathsf{CT}_{\forall\forall}^{\mathbf{res}} = \mathsf{CT}_{\forall\exists}^{\mathbf{res}}$ and $\mathsf{CT}_{I,\forall}^{\mathbf{res}} = \mathsf{CT}_{I,\exists}^{\mathbf{res}}$ for any instance I.

Proposition 8 $\mathsf{CT}_{I,\forall}^{\mathbf{std}} \subset \mathsf{CT}_{I,\forall}^{\mathbf{res}}$ *and* $\mathsf{CT}_{I,\forall}^{\mathbf{res}} \subset \mathsf{CT}_{I,\exists}^{\mathbf{std}}$ *for any non-empty instance I.*

Proof: The inclusion parts for both statements are trivial. Let us next prove the strict inclusion part for both inclusions. Without loss of generality, consider the schema containing a single binary relation R. In case it contains more relational symbols of different arities, we need to create a set of TGD's similarly to the one in formula 3.5. To prove the first inclusion, consider the set of dependencies from Example 3. For this example, it was shown that there are execution branches for which the standard chase does not terminate. On the other hand, if we consider ξ_2 to precede ξ_1, the restricted-chase algorithm will terminate. In order to prove the strict inclusion of the second formula, we may use the same example but with ξ_1 preceding ξ_2. For this set of dependencies, the restricted chase does not terminate but there exists a standard execution branch for which the standard-chase terminates$\blacksquare$

The following proposition positions the class of dependencies that ensures the restricted-chase algorithm termination compared to the other classes of dependencies.

Proposition 9 $\mathsf{CT}^{\mathbf{std}}_{\forall\forall} \subset \mathsf{CT}^{\mathbf{res}}_{\forall\forall} \subset \mathsf{CT}^{\mathbf{std}}_{\forall\exists}$.

Proof: The inclusion part follows directly from the definition of the standard and restricted-chase algorithms. The strict inclusion can be easily verified with the dependencies from Example 3 $\blacksquare$

As expected, it is undecidable if the restricted chase terminates on a given input.

Theorem 8 [16] *Let Σ be a set of TGD's and I an instance, then it is undecidable if* $\Sigma \in \mathsf{CT}^{\mathbf{std}}_{I,\exists}$

The results of Theorem 4 can be now extended to the restricted chase as follows:

Theorem 9 [16] *Let I be an instance and let Σ be a set of TGD's and EGD's such that $chase^{\mathbf{res}}_{\Sigma}(I) \neq \bot$. Then $chase^{\mathbf{std}}_{\Sigma}(I) \leftrightarrow chase^{\mathbf{res}}_{\Sigma}(I)$ and $chase^{\mathbf{res}}_{\Sigma}(I) \vDash \Sigma$.*

Cali et al. in [16] considered adding an ordering for the oblivious chase too. Still in our context, it is not necessary, as it was shown in Proposition 3, since the oblivious chase terminates, the instances returned by any execution branch are isomorphically equivalent. Thus, the instance returned by an ordered version of the oblivious-chase algorithm is isomorphically equivalent to any instance returned by the non-deterministic oblivious-chase algorithm.

3.1.3.4 The Core Chase

The class of chase algorithms is enriched by the core-chase algorithm introduced by Deutsch et al. [23]. We need to clarify from the very beginning that the core-chase differs from the other variations by the parallel execution of the standard TGD chase steps. Note that we may only apply the standard TGD chase steps in parallel and not the EGD ones, as the latter may modify the given instance by equating existing labeled nulls to constants or to other labeled nulls.

In this dissertation we slightly changed the algorithm from [23] by applying all the EGD triggers before applying in parallel the TGD triggers. This modification does not change the instance returned or the complexity of the given algorithm, see Figure 3.6.

Intuitively, the core-chase step does the following: first, it sequentially applies all EGD triggers; next, it applies in parallel all the standard TGD triggers; and finally, the core-chase step will compute the core of the instance computed before. By applying all the triggers in parallel the core-chase algorithm eliminates the nondeterministic choice introduced by the standard one. If the core-chase algorithm terminates and does not fail for input I and Σ, we denote the returned instance by $chase_{\Sigma}^{\mathbf{core}}(I)$. In case the core chase fails or it is non-terminating, we set $chase_{\Sigma}^{\mathbf{core}}(I) = \bot$.

Similar classes to the core-chase algorithm termination are introduced: $\mathsf{CT}_{\forall\forall}^{\mathbf{core}}$, $\mathsf{CT}_{\forall\exists}^{\mathbf{core}}$, $\mathsf{CT}_{I,\forall}^{\mathbf{core}}$ and $\mathsf{CT}_{I,\exists}^{\mathbf{core}}$ for some instance I. As the nondeterministic step is elim-

```
CORE-CHASE($I$,$\Sigma$)
1  $I_0 \leftarrow I$;   $i \leftarrow 0$;
2  **if** exists a standard EGD trigger $(\xi, h)$ for $I_i$
3      **then**
4            **if** $I_i \xrightarrow{(\xi,h)} \perp$
5               **then return** FAIL
6               **else** $I_i \xrightarrow{(\xi,h)} I_{i+1}$;   $i \leftarrow i+1$   **goto** 2
7  **if** exists a standard TGD trigger for $I_i$
8      **then**
9            For all triggers $(\xi, h)$ for $I_i$, compute in parallel $I_i \xrightarrow{(\xi,h)} J_j$
10           $I_{i+1} \leftarrow core(\bigcup_j J_j)$;   $i \leftarrow i+1$
11     **else**
12           **return** $I_i$
13 **goto** 2
```

Figure 3.6: The core-chase algorithm

inated for the core chase, we have: $\mathsf{CT}^{\mathbf{core}}_{\forall\forall} = \mathsf{CT}^{\mathbf{core}}_{\forall\exists}$ and $\mathsf{CT}^{\mathbf{core}}_{I,\forall} = \mathsf{CT}^{\mathbf{core}}_{I,\exists}$ for any instance I.

The following theorem gives us a convenient property of the core-chase algorithm that ensures that if the standard chase terminates on some execution branches, then the core-chase algorithm terminates.

Theorem 10 [23] *Let I be an instance and Σ a set of TGD's, then* $\mathsf{CT}^{\mathbf{std}}_{I,\exists} \subset \mathsf{CT}^{\mathbf{core}}_{I,\forall}$ *and* $\mathsf{CT}^{\mathbf{std}}_{\forall\exists} \subset \mathsf{CT}^{\mathbf{core}}_{\forall\forall}$.

Proof: We will only show the strict inclusion part of the theorem, the rest is proved by Deutsch et al. in [23]. Let us consider the set Σ containing a single dependency $\{R(x) \rightarrow \exists z\ R(z)S(x)\}$, and consider instance $I = \{R(a)\}$. For this instance and Σ, at each loop in the standard-chase algorithm, there exists exactly one trigger, and the algorithm will not terminate generating at step n the following instance I_n:

R^{I_n}
a
X_1
X_2
...
X_{n-1}
X_n

S^{I_n}
a
X_1
X_2
...
X_{n-1}

From this, it follows that $\Sigma \notin \mathsf{CT}^{\mathbf{std}}_{\forall\forall}$ and $\Sigma \notin \mathsf{CT}^{\mathbf{std}}_{\forall\exists}$. Note that for any integer n, the core of instance I_n is $core(I_n) = \{R(a), S(a)\}$. From this, it follows directly that the core-chase algorithm will stop after only one loop returning instance $J = \{R(a), S(a)\}$■

The following example shows a set of TGD's for which the standard-chase algorithm does not terminate on any execution branch but the core-chase algorithm does.

Example 8 *Consider instance $I = \{R(a,b)\}$ and the following set Σ of TGD's :*

$$\begin{aligned}\xi_1 &: \forall x,y\ R(x,y) \to \exists z\ R(y,z)\\ \xi_2 &: \forall x,y\ R(x,y), R(y,z) \to R(y,y)\end{aligned}$$

Clearly the standard-chase algorithm will not terminate for I and for Σ on any execution branch, as after the first iteration the algorithm will add the tuple $R(b,X_1)$ which will fire an infinite standard-chase sequence. On the other hand, let us not forget that, at the first iteration, the core chase will apply the trigger $(\xi_1, \{X/a, Y/b\})$, resulting the instance $I_1 = \{R(a,b), R(b,X_1)\}$. After that, the core chase will of course generate the instance $I_2 = \{R(a,b), R(b,b)\}$, which is obviously the core of the instance $J_1 = \{R(a,b), R(b,X_1), R(X_1,X_2), R(b,b)\}$. After this second iteration, there are no other triggers on I_2 and the core-chase algorithm terminates.

As expected, it is shown that the core-chase termination problem is undecidable.

Theorem 11 [23] *Let Σ be a set of TGD's and I an instance, then it is undecidable if* $\Sigma \in \mathsf{CT}^{\mathbf{core}}_{I,\forall}$.

Note that at line 10 the core-chase algorithm does not simply compute the union between all the instances computed at line 9, but it also computes its core. This gives the following result which links the result of the core-chase algorithm to the result of the standard-chase algorithm:

Theorem 12 [23] *Let I be and instance and let Σ be a set of TGD's and EGD's, such that $chase^{\mathbf{std}}_{\Sigma}(I) \neq \bot$. Then $chase^{\mathbf{std}}_{\Sigma}(I) \leftrightarrow chase^{\mathbf{core}}_{\Sigma}(I)$ and $chase^{\mathbf{core}}_{\Sigma}(I) \vDash \Sigma$, even more, $core(chase^{\mathbf{std}}_{\Sigma}(I)) = chase^{\mathbf{core}}_{\Sigma}(I)$.*

As a final remark, the core-chase algorithm computes the core of an instance at line 10 of the algorithm. Hell and Nesetril [42] showed that the decision problem *is a graph a core?*, or equivalently *is an instance a core?* is coNP-complete. This result was improved afterward by Fagin et al. in [28] by proving that the decision problem: *is instance I the core of J?* (the Core-Identification problem) is DP-complete. Where the class DP consists of all decision problems that can be written as the intersection of an NP-problem and a coNP-problem. Fagin et al. used a reduction from 3-Colorability/Non-3-Colorability. Armed with these results, we can now state the following proposition:

Proposition 10 *The core-identification problem in the CORE-CHASE algorithm is* DP*-complete.*

Proof: We will reduce the Core-Identification problem (known to be a DP-complete problem [28]) to the stated problem as follows. Given instances I and J over schema $\mathbf{R}$ consider $\Sigma = \{S(x) \rightarrow T(x)\}$, where S and T are two relations symbols not in $\mathbf{R}$.

Consider also the CORE-CHASE algorithm running with input $I \cup \{S(a)\}$ and Σ. It is easy to see that there is only one trigger to be applied, thus $J_1 = I \cup \{S(a), T(a)\}$ from line 10 of the algorithm. From this, it follows that the Core-Identification problem: *is J the core of I?* is equivalent with the following decision problem: *is $J = core(J_1)$?* from the CORE-CHASE algorithm.■

3.1.3.5 Summary

We are now ready to summarize the main properties of the different chase algorithm variations and compare the instances computed by these different algorithms.

1. for all the chase variation algorithms presented, in case they terminate for a given input, the instance returned is a model for the input instance and the input set of dependencies.

2. for all the chase algorithms, in case they terminate for a given input, the instances returned from each of these algorithms are homomorphically equivalent. On the other hand, if the instances are homomorphically equivalent, it follows that they preserve the certain answers to any UCQ query (see page 25). This means that one may use any of these chase algorithms to compute the certain answers over the "solution" instances for UCQ queries.

3. Figure 3.7 illustrates the relationship between the termination classes for all the chase algorithms presented in this chapter.

Finally, from a practical perspective, it is natural to observe that the easiest implementations are those of the oblivious and semi-oblivious chase algorithms, as they need only to apply each possible trigger only once, without testing if it satisfies or not the given dependency. In contrast, the core-chase algorithm is the hardest to implement as it needs to also implement a core computation module.

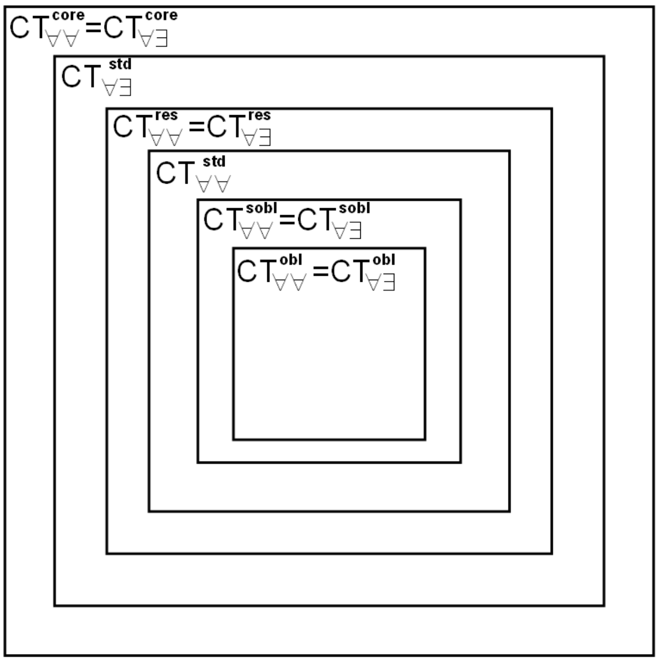
$CT^{core}_{\forall\forall} = CT^{core}_{\forall\exists}$
$CT^{std}_{\forall\exists}$
$CT^{res}_{\forall\forall} = CT^{res}_{\forall\exists}$
$CT^{std}_{\forall\forall}$
$CT^{sobl}_{\forall\forall} = CT^{sobl}_{\forall\exists}$
$CT^{obl}_{\forall\forall} = CT^{obl}_{\forall\exists}$

Figure 3.7: Termination classes for different chase variation

3.2 Chase Extensions

The chase algorithms presented in the previous sections considered only TGD's and EGD's as constraints. In this section, we will add some extensions to the core-chase algorithm in order to deal with the more general $\text{TGD}^{\vee,\neg}$ dependencies (also referred as negation disjunctive embedded dependencies). As we will see in Section 5.1, chasing $\text{TGD}^{\vee,\neg}$ dependencies helps in finding universal solution sets for the data exchange problem. These, afterwards, are used in computing certain answers for queries from the more general class $\mathsf{UCQ}^{\neg,\neq}$. To present this extension, we will use the chase algorithm introduced by Nash et al. in [23]. Recently, chasing disjunctive dependencies was also investigated by Marnette et al. in [60]. Also, chasing TGD's with unequalities are investigated by Karvounarakis and Tannen in [51], and chasing $\text{TGD}^{\vee,\neg}$ dependencies over queries were studied by Meier et al. in [63].

Let us review the extended chase step for disjunctive embedded dependencies. A disjunctive TGD$^\vee$ is an embedded dependency of the form:

$$\xi: \ \forall \bar{x}\, \alpha(\bar{x}) \to \bigvee_{1\le i\le n} \exists \bar{z}_i\, \beta_i(\bar{x}_i, \bar{z}_i) \tag{3.8}$$

where, $\bar{x}_i \subseteq \bar{x}$, for every $i \in [n]$. The formulae α and each β_i are conjunctions of relational and equality atoms. For each $i \in [n]$, let us denote by ξ_i the dependency: $\forall \bar{x}\ \alpha(\bar{x}) \to \exists \bar{z}_i\ \beta_i(\bar{x}_i, \bar{z}_i)$. The extended chase step is defined as follows: let I be an instance and h a homomorphism such that the pair (ξ_i, h) is a trigger for I. If for all $i \in [n]$, $I \xrightarrow{(\xi_i,h)} \perp$, then we say that the extended chase step on I with trigger (ξ, h) *failed*, and we denote it as $I \xrightarrow{(\xi,h)} \perp$. Otherwise, if it does not fail, let $\mathcal{J}$ be the set containing all instances J_i, such that $I \xrightarrow{(\xi_i,h)} J_i$. The set $\mathcal{J}$ is said to be obtained from I in one extended chase step with trigger (ξ, h) and it is denoted as $I \xrightarrow{(\xi,h)} \mathcal{J}$.

Example 9 *Let us consider the following TGD$^\vee$:*

$$\xi: \ \forall x, y, z\, R(x, y), R(y, z) \to R(x, z) \vee x = y \vee \exists v\, R(v, z) \tag{3.9}$$

Consider instance $I = \{R(a, b), R(b, c)\}$ and let $h = \{X/a, Y/b, Z/c\}$ be the homomorphism that maps body(ξ) to I. The three disjuncts from the head of ξ give the following dependencies:

$$\begin{aligned}
\xi_1 &: \ \forall x, y, z\, R(x, y), R(y, z) \to R(x, z)\\
\xi_2 &: \ \forall x, y, z\, R(x, y), R(y, z) \to x = y\\
\xi_3 &: \ \forall x, y, z\, R(x, y), R(y, z) \to \exists v\, R(v, z)
\end{aligned}$$

For these dependencies, we have $I \xrightarrow{(\xi_1,h)} J$, where $J = I \cup \{R(a, c)\}$, $I \xrightarrow{(\xi_2,h)} \perp$ and $I \vDash \xi_3$. Thus $I \xrightarrow{(\xi,h)} \mathcal{J}$, where $\mathcal{J} = \{J\}$.

A dependency ξ of the form $\forall \bar{x}\ \alpha(\bar{x}) \to \perp$, where α is a conjunction of atoms, is called *falsehood* [23]. If for instance I there exists homomorphism h, such that $h(\alpha(\bar{x})) \subseteq I$, then we say that the extended chase *fails* on I with (ξ, h) and it is denoted as $I \xrightarrow{(\xi,h)} \bot$.

If we allow unequalities both in the body and head of TGD$^\vee$, we obtain the class TGD$^{\vee,\neq}$. Let Σ be a set of TGD$^{\vee,\neq}$ over schema $\mathbf{R}$. In this case, the set of dependencies Σ will be replaced by the set $\Sigma^{\neq}$, in which each unequality of the from $x \neq y$ from Σ is replaced by atom $N(x,y)$, where N is a new relational symbol that does not occur in Σ. To this we add the following two new dependencies to $\Sigma^{\neq}$:

$$\forall x, y \;\; x = y \vee N(x,y) \tag{3.10}$$

$$\forall x, y \;\; x = y \wedge N(x,y) \rightarrow \bot \tag{3.11}$$

It may be noticed that the new schema for $\Sigma^{\neq}$ contains one extra relational symbol compared to the schema of Σ and that the set of dependencies also contains two new dependencies.

Finally, by considering the negation to the class TGD$^\vee$, we obtain the class TGD$^{\vee,\neg}$. Let Σ be a set of TGD$^{\vee,\neg}$ over schema $\mathbf{R}$. By $\Sigma^{\neq,\neg}$ is denoted the set of dependencies $\Sigma^{\neq}$ in which each negated atom of the form $\neg R(\bar{x})$ is replaced by a new atom $\hat{R}(\bar{x})$, and also for each relation $R \in \mathbf{R}$ the following two dependencies are added to $\Sigma^{\neq,\neg}$:

$$\forall \bar{x} \; R(\bar{x}) \vee \hat{R}(\bar{x}) \tag{3.12}$$

$$\forall \bar{x} \; R(\bar{x}) \wedge \hat{R}(\bar{x}) \rightarrow \bot \tag{3.13}$$

Note that, for any set Σ of TGD$^{\vee,\neg}$, the set of dependencies $\Sigma^{\neq,\neg}$ is in TGD$^\vee$.

Example 10 *Consider the following set of dependency $\Sigma = \{\xi_1, \xi_2\}$:*

$$\begin{array}{lcl} \xi_1 & : & \forall x, y \; R(x,y) \rightarrow x \neq y \\ \xi_2 & : & \forall x, y \; R(x,y), S(x) \rightarrow \neg S(y) \end{array}$$

The corresponding set $\Sigma^{\neq,\neg}$ will contain the following TGD$^\vee$'s:

$$
\begin{array}{lcl}
\xi_1 & : & \forall x,y\; R(x,y) \to N(x,y) \\
\xi_2 & : & \forall x,y\; R(x,y), S(x) \to \hat{S}(y) \\
\xi_3 & : & \forall x,y\; \big(x = y \vee N(x,y)\big) \\
\xi_4 & : & \forall x,y\; \big(x = y \wedge N(x,y) \to \bot\big) \\
\xi_3 & : & \forall x,y\; R(x,y) \vee \hat{R}(x,y) \\
\xi_4 & : & \forall x,y\; R(x,y) \wedge \hat{R}(x,y) \to \bot \\
\xi_5 & : & \forall x\; S(x) \vee \hat{S}(x) \\
\xi_6 & : & \forall x\; S(x) \wedge \hat{S}(x) \to \bot
\end{array}
$$

Using the previous notations, Figure 3.8 illustrates the *extended core-chase* algorithm introduced by Deutsch, Nash and Remmel in [23]. The algorithm has as input an instance $I \in Inst(\mathbf{R})$ and a set Σ of TGD$^{\vee,\neg}$ and returns a set of instances over schema $\mathbf{R}$.

```
EXTENDED-CORE-CHASE($I$,$\Sigma$)
1  $\mathcal{L}_0 \leftarrow \{I\}$;   $i \leftarrow 0$;
2  Compute in parallel for each instance $J \in \mathcal{L}_i$ the set $\mathcal{K}_J$
   where $K \in \mathcal{K}_J$ iff $J \xrightarrow{(\xi,h)} K$ for some $\xi \in \Sigma^{\neq,\neg}$ and homomorphism $h$
3  $\mathcal{L}' \leftarrow \bigcup_{J \in \mathcal{L}_i} \bigcup_{K \in \mathcal{K}_J} \{core(K)\}$
4  compute $\mathcal{L}_{i+1}$ by removing from $\mathcal{L}'$ all $K$ such that $\exists L \in \mathcal{L}'$, $L \to K$;
5  $i \leftarrow i + 1$;
6  **if** $\mathcal{L}_i = \mathcal{L}_{i-1}$
7      **then**
8          **return** the set of instances from $\mathcal{L}_i$ restricted to $\mathbf{R}$
9      **else**
10         **goto** 2
```

Figure 3.8: The extended core-chase algorithm

Note that at Step 2 of the algorithm the trigger (ξ, h) is either an EGD or a TGD trigger for J. Also observe that line 8 of the algorithm removes from the set of instances $\mathcal{L}_i$ all the tuples not in the initial schema of I, that is all the new extra relations added to the initial schema.

Example 11 *Consider $\Sigma = \{T(x) \rightarrow R(x)\}$ over schema $\{R, S, T\}$ and consider instance $I = \{T(a)\}$. Running the extended chase algorithm with input I and Σ the value of $\mathcal{L}_1$ after executing first time step 4 is:*

$$\mathcal{L}_1 = \{\{T(a), R(a), S(a)\}, \{T(a), R(a), \hat{S}(a)\}\},$$

thus the algorithm will return the following set, after eliminating all the relational symbols not in the initial schema: $\{\{T(a), R(a), S(a)\}, \{T(a), R(a)\}\}$.

Intuitively, the algorithm, if it terminates, returns an incomplete database with each of the instance representing possible worlds that model I and Σ. The semantic used for the result is a closed world semantics. That is if I is a possible world and $R(\bar{x})$ is an atom such that $I \not\models R(\bar{x})$, it follows that $I \models \neg R(\bar{x})$.

Proposition 11 [27] *Let $\mathcal{L}$ be the set of instances returned by the extended core-chase algorithm for input I and Σ a set of TGD$^{\vee,\neg}$'s. Then for all $J \in \mathcal{L}$ it holds that $J \models \Sigma$.*

We conclude this section with the corollary which follows directly from Theorem 11 and the fact that TGD $\subset$ TGD$^{\vee,\neg}$.

Corollary 4 *Let I be an instance and Σ a set of TGD$^{\vee,\neg}$. Then it is undecidable if the extended core-chase algorithm terminates with input I and Σ.*

Chapter 4

Sufficient Conditions for Chase Termination

4.1 The Chase Termination Problem

The undecidability results for the different chase variation algorithms presented in Chapter 3 motivated the research community to find classes of tuple-generating dependencies that ensure the termination for some of the chase variation algorithms for all input instances. In this chapter we focus on part of these classes of dependencies for which it is known that the standard chase terminates on all instances. Even if most of these classes were constructed to ensure the standard-chase termination, we will show in this chapter that all these classes guarantee the termination for a more easily implementable algorithm, that is the semi-oblivious-chase algorithm. Therefore, we will pairwise compare these classes of dependencies and conclude with a Hasse diagram for the subset relation between these classes which will give us a clear overview of these results.

4.2 Dependencies with Stratified-Witness

This first class of TGD's that guarantees the standard-chase termination was introduced [25] simultaneously with the class of weakly acyclic TGD's [27], the later one being a strict superset of the first one.

Definition 1 [16; 27] *For a given database schema* $\mathbf{R}$ *define a* position *in* $\mathbf{R}$ *to be a pair* (R,k)*, where* R *is a relation symbol from* $\mathbf{R}$ *and* $k \in [arity(R)]$.

Definition 2 [25] *Let* Σ *be a set of TGD's over schema* $\mathbf{R}$*. The* chase flow graph *associated with* Σ *is a directed edge-labeled graph* $G^F_\Sigma = (V,E)$*, such that each vertex represents a position in* $\mathbf{R}$ *and* $((R,i),(S,j)) \in E$ *if there exists a TGD* $\xi \in \Sigma$ *of the form* $\forall \bar{x},\bar{y}\ \alpha(\bar{x},\bar{y}) \to \exists \bar{z}\ \beta(\bar{x},\bar{z})$*, and if there exists a variable* $x \in \bar{x} \cup \bar{y}$ *occuring in position* (R,i) *in* α *and variable* $y \in \bar{x} \cup \bar{z}$ *that occurs in position* (S,j) *in* β*. In case* $y \in \bar{z}$*, the edge is labeled as* existential*, otherwise the label is considered* universal.

Example 12 *Consider schema* $\mathbf{R} = \{S,R\}$*, with* $arity(S) = 2$ *and* $arity(R) = 2$*. The positions in* $\mathbf{R}$ *are* $\{(S,2),(S,1),(R,1),(R,2)\}$*. Let* Σ_1 *contain the following TGD over* $\mathbf{R}$*:*

$$\xi_{11} \quad : \quad S(x,y) \to \exists z\ R(x,z)$$

let Σ_2 *be:*

$$\begin{aligned} \xi_{21} &\quad : \quad S(x,y) \to \exists z\ R(x,z) \\ \xi_{22} &\quad : \quad R(x,y) \to S(x,x) \end{aligned}$$

let Σ_3 *be:*

$$\begin{aligned} \xi_{31} &\quad : \quad S(x,y) \to \exists z\ R(x,z) \\ \xi_{32} &\quad : \quad R(x,y) \to \exists z\ R(x,z) \end{aligned}$$

and, finally, let Σ_4 be a slight modification of Σ_3:

$$\begin{aligned} \xi_{41} &: \quad S(x,y) \rightarrow \exists z\, R(x,z) \\ \xi_{42} &: \quad R(x,y) \rightarrow \exists z\, R(y,z) \end{aligned}$$

Figure 4.1 captures the chase flow graphs associated with Σ_1, Σ_2, Σ_3 and Σ_4 (the existential edges are represented as dotted lines in the graphs). Note that the graphs for Σ_3 and Σ_4 are the same because the flow graph does not distinguish the universal variables.

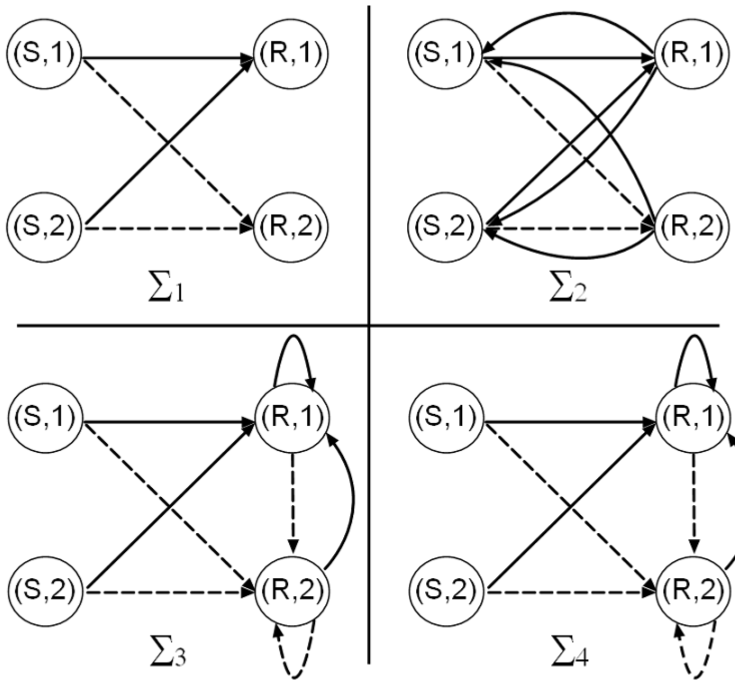

Figure 4.1: Chase flow graph

Definition 3 [25] *A set Σ of TGD's has* stratified-witness *if its corresponding flow graph has no cycles through an existential edge. By* SW *we denote the class of all sets of dependencies with stratified-witness.*

Returning to example 12, only Σ_1 has stratified-witness, all Σ_2,Σ_3 and Σ_4 have in their corresponding chase flow graph at least one cycle through an existential edge (see Figure 4.1). The following theorem allows us to know that the standard chase terminates for any set of TGD's with stratified-witness.

Theorem 13 [25] *If $\Sigma \in TGD(\mathbf{R})$ such that $\Sigma \in \mathsf{SW}$, then $\Sigma \in \mathsf{CT}^{\mathbf{std}}_{\forall\forall}$. Even more, if* i *denotes the maximum arity in* **R** *and* k *represents the number of existential edges in the flow graph, then the standard-chase algorithm terminates for instance I in $O(|I|^{i^{k+1}})$ time.*

From this theorem and the termination class hierarchy presented in Figure 3.7, it follows that if Σ is a set of TGD's with stratified-witness, then Σ ensures termination on all instances also for the core chase and restricted-chase algorithms. In the following section we will extend this result by showing that there exists a larger class of TGD's that guarantees the termination on all instances for the oblivious case following that any set of TGD's with stratified-witness are also in $\mathsf{CT}^{\mathbf{obl}}_{\forall\forall}$ and $\mathsf{CT}^{\mathbf{sobl}}_{\forall\forall}$.

4.3 Rich Acyclicity

The class of *richly acyclic* set of dependencies was introduced by Hernich and Schweikardt in [47] in a different context. We will show here that any set of dependencies from this new class ensures the oblivious-chase algorithm termination for all input instances.

Definition 4 [47] *Let Σ be a set of TGD's over schema* **R**. *The* extended dependency graph *associated with Σ is a directed edge-labeled graph $G^E_\Sigma = (V, E)$, such that each vertex represents a position in* **R** *and $((R,i),(S,j)) \in E$, if there exists a TGD $\xi \in \Sigma$ of the form $\forall \bar{x}, \bar{y}\ \alpha(\bar{x},\bar{y}) \to \exists \bar{z}\ \beta(\bar{x},\bar{z})$, and if one of the following holds:*

1. *$x \in \bar{x}$ and x occurs in α on position (R,i) and in β on position (S,j). In this case the edge is labeled as universal;*

2. $x \in \bar{x} \cup \bar{y}$ and x occurs in α on position (R, i) and variable $z \in \bar{z}$ that occurs in β on position (S, j). In this case the edge is labeled as existential.

Figure 4.2 illustrates the extended dependency graph for the sets of dependencies specified in Example 12 (the existential edges are represented as dotted lines in the graphs). Note that the extended dependency graph for Σ_3 and Σ_4 are not identical, as they were for the flow graph. This is due to the fact that the extended dependency graph distinguishes between the universally quantified variables.

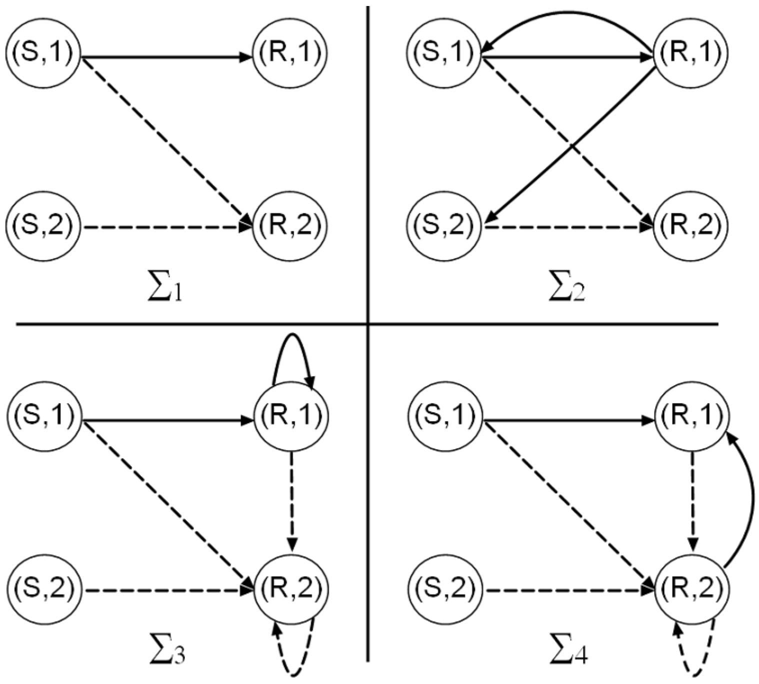

Figure 4.2: Extended dependency graph

Definition 5 [47] *A set Σ of TGD's is said to be richly acyclic if the corresponding extended dependency graph has no cycles through an existential edge. By* RA *we denote the class of all richly acyclic sets of dependencies.*

Returning to Example 12 from page 67, we may note that both Σ_1 and Σ_2 are richly acyclic but that Σ_3 and Σ_4 are not. It is obvious that the problem of testing if a given set of dependency Σ is in RA has polynomial complexity in the size of Σ. It also follows directly from the definitions that $\mathsf{SW} \subset \mathsf{RA}$. For the strict inclusion part see Σ_2 from the previous example.

The following theorem shows that rich acyclicity implies oblivious-chase algorithm termination on all instances in polynomial time in the size of the input instance.

Theorem 14 $\mathsf{RA} \subset \mathsf{CT}^{\mathbf{obl}}_{\forall\forall}$ *and for any* $\Sigma \in \mathsf{RA}$ *and any instance* I *there is a polynomial in the size of* I *that bounds the execution time of the oblivious-chase algorithm with* I *and* Σ.

Proof: The proof of this theorem uses some of the notions introduced in the following section, therefore the proof will be detailed in the next section at page 75∎

From this theorem and the class hierarchy illustrated in Figure 3.7, it follows that if Σ is a richly acyclic set of dependencies, then Σ ensures termination on all instances for all chase variations presented in Section 3.1.3.

4.4 Weak Acyclicity

Fagin et al. [27] introduced the class of *weakly acyclic* dependencies as a class that ensures termination of the standard-chase algorithm on all execution branches for all instances. This class of dependencies was initially introduced in the context of finding *universal solutions* for the data exchange problem.

Definition 6 [27] *Let* Σ *be a set of TGD's over schema* $\mathbf{R}$. *The* dependency graph *associated with* Σ *is a directed edge-labeled graph* $G^D_\Sigma = (V, E)$, *such that each vertex* *represents a position in* $\mathbf{R}$ *and* $((R,i),(S,j)) \in E$*, if there exists a TGD* $\xi \in \Sigma$ *of the form* $\forall \bar{x}, \bar{y}\ \alpha(\bar{x}, \bar{y}) \rightarrow \exists \bar{z}\ \beta(\bar{x}, \bar{z})$*, and if one of the following holds:*

1. $x \in \bar{x}$ *and* x *occurs in* α *on position* (R,i) *and in* β *on position* (S,j)*. In this case the edge is labeled as universal;*

2. $x \in \bar{x}$ *and* x *occurs in* α *on position* (R,i) *and variable* $z \in \bar{z}$ *that occurs in* β *on position* (S,j)*. In this case the edge is labeled as existential.*

The previous definition differs from Definition 4 only by considering existential edges only from positions occupied by universally quantified variables that occur both in the body and head of dependency.

Figure 4.3 illustrates the dependency graphs for the sets of dependencies described in Example 12 (the existential edges are represented as dotted lines in the graphs).

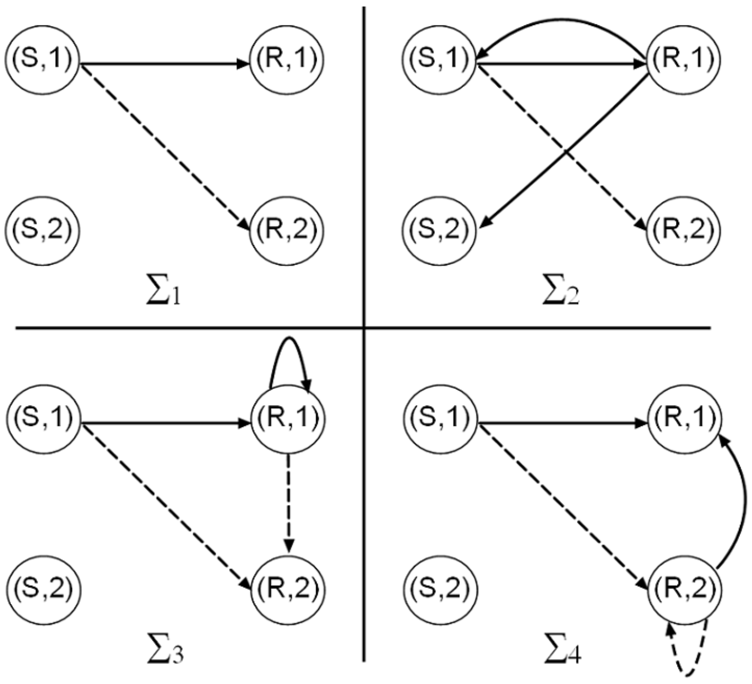

Figure 4.3: Dependency graph

Definition 7 [27] *A set of TGD's Σ is said to be* weakly acyclic *if the corresponding dependency graph does not have any cycle going trough an existential edge. By* WA *we denote the class of all weakly acyclic sets of dependencies.*

Returning to Example 12 (see page 67), the first three sets of dependencies are weakly acyclic, there are no edges through existentially labeled edges. The set Σ_3 was shown not to be richly acyclic but weakly acyclic, which gives us the strict inclusion $\mathsf{RA} \subset \mathsf{WA}$. On the other hand, Gottlob and Nash noticed [34] that any weakly acyclic set of dependency can be changed in a logically equivalent richly acyclic set of dependencies. To obtain this, each tgd of the form $\alpha(\bar{x}, \bar{y}) \rightarrow \exists \bar{z}\, \beta(\bar{x}, \bar{z})$, in case $|\bar{y}| > 0$, is replaced with the following two TGD's:

$$\alpha(\bar{x}, \bar{y}) \rightarrow B(\bar{x}) \tag{4.1}$$

$$B(\bar{x}) \rightarrow \exists \bar{z}\, \beta(\bar{x}, \bar{z}) \tag{4.2}$$

The following theorem by Fagin et al. [27] shows that, for a weakly acyclic set of dependencies, the standard-chase algorithm will terminate on all execution branches for all instances in a polynomial number of steps, in the size of the input instance.

Theorem 15 [27] $\mathsf{WA} \subset \mathsf{CT}^{\mathbf{std}}_{\forall\forall}$ *and for any $\Sigma \in \mathsf{WA}$ and any instance I there is a polynomial, in the size of the input instance, that bounds the execution time of the standard-chase algorithm with I and Σ.*

Because of this theorem and the class hierarchy shown in Figure 3.7, it follows that if Σ is a richly acyclic set of dependencies, then Σ ensures termination for standard, restricted and core-chase algorithms. For the strict inclusion part from the previous theorem, let us consider the following set of tgds $\Sigma = \{R(x, x) \rightarrow \exists y\, R(y, x)\}$. Clearly, Σ is not weakly acyclic. Still, for any instance I none of the tuples generated by the head of the dependency will trigger the body. Thus the standard-chase algorithm terminates on all branches for all input instances.

The following result extends the previous theorem by relating the WA class with the $\mathsf{CT}^{\mathbf{sobl}}_{\forall\forall}$ and $\mathsf{CT}^{\mathbf{obl}}_{\forall\forall}$ classes.

Proposition 12 $\mathsf{WA} \subset \mathsf{CT}^{\mathbf{sobl}}_{\forall\forall}$ *and* $\mathsf{WA} \nparallel \mathsf{CT}^{\mathbf{obl}}_{\forall\forall}$.

Proof: First it can be easily observed that if $\Sigma \in \mathsf{WA}$, then its semi-enrichment $\tilde{\Sigma} \in \mathsf{WA}$. From this and Theorems 5 and 15, it directly follows that $\mathsf{WA} \subseteq \mathsf{CT}^{\mathbf{sobl}}_{\forall\forall}$. For the strict inclusion part, consider $\Sigma_1 = \{S(y), R(x,y) \to \exists z\ R(x,z)\}$. It is clear that $\Sigma_1 \notin \mathsf{WA}$, but the standard-chase algorithm terminates on all branches for any instance I and Σ_1. For the second part, consider $\Sigma_2 = \{R(x,y) \to \exists z\ R(x,z)\}$, then clearly $\Sigma_2 \in \mathsf{WA}\dot{\mathsf{A}}$nd from Example 6 we have that $\Sigma_2 \notin \mathsf{CT}^{\mathbf{obl}}_{\forall\forall}$. On the other hand, the previous set $\Sigma_1 \in \mathsf{CT}^{\mathbf{obl}}_{\forall\forall}$ and as shown $\Sigma_1 \notin \mathsf{WA}$. From this, it follows that $\mathsf{WA} \nparallel \mathsf{CT}^{\mathbf{obl}}_{\forall\forall}$■

Lemma 1 *Let* Σ *be a richly acyclic set of TGD's then* $\widehat{\Sigma}$*, the enrichment of* Σ*, is a weakly acyclic set of TGD's.*

Proof: Let Σ be a set of TGD's with the corresponding extended dependency graph $G^E_\Sigma = (V^E, E^E)$. Let $\widehat{\Sigma}$ be the enrichment of Σ with the corresponding dependency graph $G^D_{\widehat{\Sigma}} = (V^D, E^D)$. We will show first that the graph $G^D_{\widehat{\Sigma}}$ restricted to the vertexes V^E is the same as G^E_Σ. For this, let us consider a tgd $\xi \in \Sigma$ of the form:

$$\forall \bar{x}, \forall \bar{y}\ R_1(\bar{x}_1, \bar{y}_1), R_2(\bar{x}_2, \bar{y}_2), \ldots, R_n(\bar{x}_n, \bar{y}_n) \to \exists \bar{z}\ S_1(\bar{x}'_1, \bar{z}_1), S_2(\bar{x}'_2, \bar{z}_2), \ldots, S_m(\bar{x}'_m, \bar{z}_n)$$

where $\bar{x} = \cup_{i\in[n]}\bar{x}_i = \cup_{j\in[m]}\bar{x}'_j$, $\bar{z} = \cup_{i\in[m]}\bar{z}_i$ and $\bar{y} = \cup_{i\in[n]}\bar{y}_i$. The corresponding enriched dependency $\hat{\xi}$ belonging to $\widehat{\Sigma}$ will have the following form:

$$\begin{aligned}&\forall \bar{x}, \forall \bar{y}\ R_1(\bar{x}_1, \bar{y}_1), R_2(\bar{x}_2, \bar{y}_2), \ldots, R_n(\bar{x}_n, \bar{y}_n) \to \\ &\qquad\qquad \exists \bar{z}\ S_1(\bar{x}'_1, \bar{z}_1), S_2(\bar{x}'_2, \bar{z}_2), \ldots, S_m(\bar{x}'_m, \bar{z}_n), H(\bar{x}, \bar{y})\end{aligned}$$

where H is a new relational symbol. Let us suppose that ξ creates the universal edge $((R_i,k),(S_j,l))$ in the extended dependency graph G^E_Σ. From this it follows that the same universal edge is also part of $G^D_{\widehat{\Sigma}}$ because both dependencies use the same set of universal variables that occurs in the same positions in both ξ and $\hat{\xi}$. Let us now suppose that ξ creates the existential edge $((R_i,k),(S_j,l))$ in the extended dependency graph G^E_Σ. This means that on position (R_i,k) occurs a variable either from $\bar{x}$ or $\bar{y}$ and that on position (S_j,l) from the head occurs a variable from $\bar{z}$. Clearly, $((R_i,k),(S_j,l))$ is also an existential edge in the dependency graph $G^D_{\widehat{\Sigma}}$. The only edges in $G^D_{\widehat{\Sigma}}$ not occurring in G^E_Σ are the universal edges oriented to the positions over relation H. As relation H does not appear in the body of any dependency in $\widehat{\Sigma}$, it follows that there can't be any cycle in $G^D_{\widehat{\Sigma}}$ through a position over H. But this means that any cycle in G^E_Σ is also a cycle in $G^D_{\widehat{\Sigma}}$ and vice versa. From this it directly follows that Σ is a richly acyclic set of TGD's if and only if $\widehat{\Sigma}$ is a weakly acyclic set of TGD's$_\blacksquare$

Now we are ready to prove Theorem 14 stated at page 71.

Theorem 14 $\mathsf{RA} \subset \mathsf{CT}^{\mathsf{obl}}_{\forall\forall}$ *and for any* $\Sigma \in \mathsf{RA}$ *and any instance* I *there is a polynomial, in the size of the instance* I*, that bounds the execution time of the oblivious-chase algorithm with* I *and* Σ*.*

Proof: Let Σ be a richly acyclic set of TGD's. From the previous theorem we have that $\widehat{\Sigma}$ is a weakly acyclic set of TGD's. On the other hand, from Theorem 15 it follows that the standard-chase procedure terminates with $\widehat{\Sigma}$ for any instance I, that is $\widehat{\Sigma} \in \mathsf{CT}^{\mathsf{std}}_{\forall\forall}$. From this and Theorem 3 follows that $\Sigma \in \mathsf{CT}^{\mathsf{obl}}_{\forall\forall}$. For the strict part inclusion it is easy to verify that the dependency $\Sigma = \{R(x,x) \rightarrow \exists y\ R(x,y), T(x)\}$ ensures the oblivious chase termination for any input instance, that is $\Sigma \in \mathsf{CT}^{\mathsf{obl}}_{\forall\forall}$, but $\Sigma \notin \mathsf{RA}$. For the second part of the theorem, let I and let k_I be the polynomial that bounds the execution time for the standard chase on input $\widehat{\Sigma}$ and I, as given by Theorem 15. But from the proof of

Theorem 3 we have that the number of steps executed by the standard-chase algorithm on input $\widehat{\Sigma}$ and I is the same with the number of steps executed by the oblivious-chase algorithm on input Σ and I. Thus k_I is also a polynomial that bounds the execution time for the oblivious chase on input Σ and I∎

4.5 Safe Dependencies

Meier, Schmidt and Lausen [61] observed that the weak acyclicity condition restricts also tgds that introduce new nulls that may not create infinite standard-chase sequences.

Example 13 *Let us consider an example from [61] with Σ containing only one dependency TGD:*

$$\xi \,:\, R(x,y,z), S(y) \rightarrow \exists w\; R(y,w,x)$$

Figure 4.4 represents the dependency graph corresponding to Σ. As it can be seen in the figure, the dependency graph contains a cycle going through an existential edge. Thus Σ is not weakly acyclic. On the other hand, the newly created null in position $(R,2)$ may create new null values only if the same null also appears in position $(S,1)$. Based on the given dependency, new nulls cannot be generated in position $(S,1)$. Thus this dependency can not cyclically create new nulls.

In order to be able to capture the type of dependencies presented in the previous example, Meier et al. observed that we need to change the dependency graph such that it does not contain as vertexes all the position in the schema, but only the ones that may contribute to an infinite cycle, that is positions in which new null values can occur.

Before introducing the notion of safe dependencies, let us first present the definition of the affected position introduced by Cali et al. in [16].

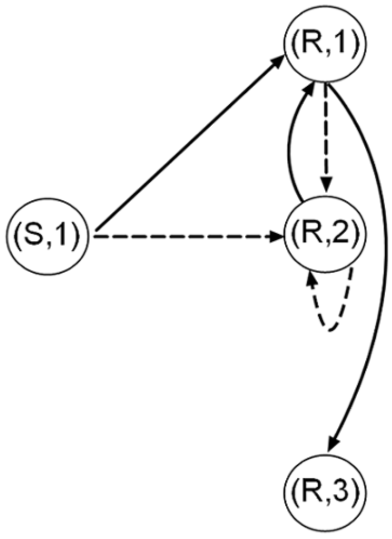

Figure 4.4: Dependency graph with cycles through existential labeled edges

Definition 8 [16] *The* affected positions *associated with a set of TGD's* Σ *is the set* $aff(\Sigma)$ *defined recursively as follows. Position* $(R, i) \in aff(\Sigma)$ *if:* $\xi \in \Sigma$, *then*

1. *if an existential variable appears in position* (R, i) *in* ξ, *or;*
2. *if universally quantified variable* x *appears in position* (R, i) *in the head of some dependency* $\xi \in \Sigma$ *and* x *occurs only in affected positions in the body of* ξ.

Returning to Example 13, it can be easily noted that $aff(\Sigma) = \{(R, 2)\}$. Similarly to the dependency graph, the *propagation graph* is defined as:

Definition 9 [61] *Let* Σ *be a set of TGD's . The* propagation graph *associated with* Σ *is a directed edge labeled graph* $G^P_\Sigma = (aff(\Sigma), E)$, *where* $((R, i), (S, j)) \in E$ *if there exists a dependency* $\xi \in \Sigma$ *of the form* $\forall \bar{x}, \bar{y}\ \alpha(\bar{x}, \bar{y}) \rightarrow \exists \bar{z}\ \beta(\bar{x}, \bar{z})$, *and there exists a variable* $x \in \bar{x}$ *that occurs in the body of* ξ *in position* (R, i), *such that* x *occurs only in affected positions in body of* ξ *and either*

- x *occurs in position* (S, j) *in* β, *or*

- *there exists a variable $z \in \bar{z}$ in position (S,j) in β.*

In the first case, the edge is labeled as universal *and, in the second case, the edge is labeled as* existential*.*

Considering again Example 13, as shown, the only affected position is $(R,2)$. On the other hand, only the universally quantified variable y occurs in this affect position in the body of the dependency, variable that occurs in a non-affected position $(R,1)$ in the head of the dependency.

Example 14 [61] *Consider Σ consisting of the following two TGD's:*

$$\begin{aligned}
\xi_1 : \ S(x), R(x,y) &\rightarrow R(y,x) \\
\xi_2 : \ S(x), R(x,y) &\rightarrow \exists z\, R(y,z), R(z,x)
\end{aligned}$$

asserting that all nodes in S have cycles of length 2 (ξ_1) and 3 (ξ_2). The set of affected positions in this case is $aff(\Sigma) = \{(R,1),(R,2)\}$. Figure 4.5 shows the propagation graph corresponding to Σ (the dotted lines represent the existential edges).

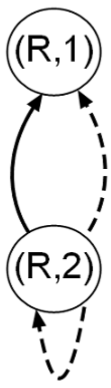

Figure 4.5: Propagation graph

Definition 10 [61] *A set Σ of TGD's is called* safe *if its propagation graph G^P_Σ does not have a cycle going trough an existential edge. By* SD *to be the class of all safe dependency sets.*

Using this definition, it follows that the set of dependencies from Example 13 is safe as the corresponding propagation graph contains a single vertex and no edges. On the other hand, the dependencies from Example 14 are not safe. Note the cycle through an existentially labeled edge in Figure 4.5. It may be also noted that the set of dependencies from Example 14 guarantees the standard-chase termination for all instances.

Theorem 16 [61] $\mathsf{SD} \subset \mathsf{CT}^{\mathbf{std}}_{\forall\forall}$ *and for any* $\Sigma \in \mathsf{SD}$ *and any instance I there is a polynomial, in the size of I, that bounds the execution time of the oblivious-chase algorithm with I and Σ.*

For the strict inclusion part of the previous theorem, consider that the set of dependencies from Example 14 is in $\mathsf{CT}^{\mathbf{std}}_{\forall\forall}$ but it is not safe. Meier et al. [61] also proved that $\mathsf{WA} \subset \mathsf{SD}$. However, this does not hold with the propagation graph defined in their paper. For the strict inclusion between the classes WA and SD consider the set of dependencies from Example 13 which is safe but not weakly acyclic. The following proposition extends the previous result by showing that the safe dependencies also ensure the semi-oblivious-chase termination for all instances but do not guarantee the oblivious-chase termination on all instances.

Proposition 13 $\mathsf{SD} \subset \mathsf{CT}^{\mathbf{sobl}}_{\forall\forall}$ *and* $\mathsf{SD} \nparallel \mathsf{CT}^{\mathbf{obl}}_{\forall\forall}$.

Proof: First it can be observed that if $\Sigma \in \mathsf{SD}$, then its semi-enrichment $\widehat{\Sigma} \in \mathsf{SD}$, as the corresponding propagation graphs restricted to the positions over Σ are the same, and the new relations introduced in $\widehat{\Sigma}$ do not induce any cycle in the corresponding propagation graph. From this and Theorems 5 and 16 it directly follows that $\mathsf{SD} \subseteq \mathsf{CT}^{\mathbf{sobl}}_{\forall\forall}$. For the strict inclusion part, consider $\Sigma_1 = \{R(x,x) \to \exists y\ R(x,y)\}$. It is obvious that $\Sigma_1 \notin \mathsf{SD}$, but the semi-oblivious-chase algorithm terminates for any instance I and Σ_1. For the second part, consider $\Sigma_2 = \{R(x,y) \to \exists z\ R(x,z)\}$. Clearly $\Sigma_2 \in \mathsf{SD}$ and from Example 6 we have that $\Sigma_2 \notin \mathsf{CT}^{\mathbf{obl}}_{\forall\forall}$. Let us not forget that the previous set $\Sigma_1 \in \mathsf{CT}^{\mathbf{obl}}_{\forall\forall}$ and, as shown, $\Sigma_1 \notin \mathsf{SD}$. The consequence is that $\mathsf{SD} \nparallel \mathsf{CT}^{\mathbf{obl}}_{\forall\forall}$ ∎

4.6 Super Weak Acyclicity

The following class of dependencies properly extends the class of safe dependencies and consequently the class of weakly acyclic, richly acyclic dependencies and the class of dependencies with stratified witness. This class of dependencies, introduced by Marnette [59], beside omitting the nulls that can't generate infinite chase sequences, as in the case of safe dependencies, also takes into account the repeating variables. For a uniform presentation of the dependency classes, we will slightly change the notation used in [59]

We assume that the set of dependencies Σ has distinct variable names in each dependency. We also assume that there exists a total order between the atoms in each dependency. With this, we can now define the notion of *atom position* to be a triple (ξ, R, i), where $\xi \in \Sigma$, R is a relation name that occurs in ξ, $i \in [n]$, where n is the maximum number of occurrences of R in ξ, given by the total order between the atoms in the dependency. Clearly each atom position uniquely identifies an atom in Σ. Similarly to the notion of position, a *place* can be defined as a pair $((\xi, R, i), k)$, where (ξ, R, i) is an atom position and $k \in [arity(R)]$. Intuitively, the place identifies the variable that appears in the k-th position in the atom given by the atom position (ξ, R, i).

Let $Var(\xi)$ define the set of all variables that occur in dependency $\xi \in \Sigma$. As mentioned, for any two distinct dependencies ξ_1 and ξ_2 in Σ, we have $Var(\xi_1) \cap Var(\xi_2) = \varnothing$. The mapping Var is extended to a set of dependencies Σ as $Var(\Sigma) = \cup_{\xi \in \Sigma} Var(\xi)$. Similarly, we define mappings $Var^{\exists}$ and $Var^{\forall}$ that map each dependency ξ to the set of existentially marked variables in ξ and to the universally quantified variables in ξ respectively. Clearly, for each dependency ξ, $Var^{\exists}(\xi)$ and $Var^{\forall}(\xi)$ represent a partition of $Var(\xi)$.

Given a TGD ξ and $y \in Var^{\exists}(\xi)$, $\mathsf{Out}(\xi, y)$ is defined to be the set of places in the head of ξ where y occurs. Given a set a TGD ξ and $x \in Var^{\forall}(\xi)$, $\mathsf{In}(\xi, x)$ is defined to be the set of places in the body of ξ where x occurs.

A substitution for Σ is a function θ with $Dom(\theta) = \Delta_{\mathsf{V}}$ and $Im(\theta) = \Delta_{\mathsf{C}}$ such that for any $x \in Var^{\exists}(\Sigma)$ there is no variable $y \in Var(\Sigma)$ with $x \neq y$ such that $\theta(x) = \theta(y)$. Given a substitution θ for Σ and an atom (ξ, R, i) from Σ, the new atom resulting by replacing each variable x in (ξ, R, i) with $\theta(x)$ is denoted by $\theta(\xi, R, i)$. Two atoms, (ξ_1, R, i_1) and (ξ_2, R, i_2), are said to be *unifiable* if there exist two substitutions θ_1 and θ_2 for Σ such that $\theta_1(\xi_1, R, i_1) = \theta_2(\xi_2, R, i_2)$. Two places $p_1 = ((\xi_1, R, i_1), k_1)$ and $p_2 = ((\xi_2, R, i_2), k_2)$ are said to be *unifiable* if $k_1 = k_2$ and (ξ_1, R, i_1) is unifiable with (ξ_2, R, i_2). By $p_1 \sim p_2$ we denote that p_1 and p_2 are unifiable. Let us define Γ_{Σ} to be a function that maps each variable x to the set of places where x occurs in Σ. The function Γ_{Σ}^{H} maps each variable x to the set of places from the head of some dependency where x occurs. Similarly, the function Γ_{Σ}^{B} maps each variable x to the set of places from the body of some dependency where x occurs.

For a better understanding of the previous notions, consider the example:

Example 15 [70] *Let Σ contain two dependencies:*

$$\begin{aligned} \xi_1 &: R(x) \to \exists y, z\, S(x, y, z) \\ \xi_2 &: S(v, w, w) \to R(w) \end{aligned}$$

The following are the atom positions for Σ:

- $(\xi_1, R, 1)$, corresponding to atom $R(x)$;
- $(\xi_1, S, 1)$, corresponding to atom $S(x, y, z)$;
- $(\xi_2, S, 1)$, corresponding to atom $S(v, w, w)$;
- $(\xi_2, R, 1)$, corresponding to atom $R(w)$.

For this setting we have the set $Var^{\forall}(\Sigma) = \{x, v, w\}$, $Var^{\exists}(\Sigma) = \{y, z\}$ and the set $Var^{\forall}(\Sigma) = Var^{\forall}(\Sigma) \cup Var^{\exists}(\Sigma)$. Clearly $(\xi_1, R, 1)$ is unifiable with $(\xi_2, R, 1)$, consider unifier $\theta_1 = \{x/a\}$ and $\theta_2 = \{w/a\}$. On the other hand, atom positions $(\xi_1, S, 1)$ and $(\xi_2, S, 1)$ are not unifiable as there do not exist two unifiers θ_1 and θ_2 for the atoms $S(x, y, z)$ and $S(v, w, w)$ such that the existential variables y, z each are mapped to a new constant. From this we have that $((\xi_1, R, 1), 1) \sim ((\xi_2, R, 1), 1)$. For variable x, we have the sets $\Gamma_{\Sigma}(x) = \{((\xi_1, R, 1), 1), ((\xi_1, S, 1), 1)\}$, $\Gamma^{B}_{\Sigma}(x) = \{((\xi_1, R, 1), 1)\}$ and $\Gamma^{H}_{\Sigma}(x) = \{((\xi_1, S, 1), 1)\}$.

Given two sets of places P and Q, we denote $P \sqsubseteq Q$ if for all $p \in P$ if $p \sim q$, then $q \in Q$. The mapping $\mathsf{Move}(\Sigma, Q)$ is defined to return the smallest set of places P, with $Q \subseteq P$ and for all variables x that occur in a body of some dependency $\xi \in \Sigma$ if $\Gamma^{B}_{\Sigma}(x) \sqsubseteq P$ then $\Gamma^{H}_{\Sigma}(x) \subseteq P$. Intuitively, the $\mathsf{Move}(\Sigma, Q)$ returns the smallest set of places such that the values generated in those places by chasing some tuples that contain the places in Q are dependent on the values in tuples in places Q. Returning to the dependencies from the previous example and considering the place $p_1 = (\xi_2, R, 1), 1)$, with this place we have only place $p_2 = (\xi_1, R, 1), 1)$ such that $p_1 \sim p_2$. Also we have $\Gamma^{B}_{\Sigma}(x) = \{p_2\}$ and $\Gamma^{H}_{\Sigma}(x) = \{p_3\}$, where $p_3 = ((\xi_1, S, 1), 1)$. From this it follows that $\mathsf{Move}(\Sigma, \{p_1\}) = \{p_1, p_2, p_3\}$.

Definition 11 [59] *Given Σ a set of TGD's and $\xi_1, \xi_2 \in \Sigma$, we say ξ_1 triggers ξ_2 in Σ, and it is denoted with $\xi_1 \rightsquigarrow_{\Sigma} \xi_2$, iff there exists a variable $y \in Var^{\exists}(\xi_1)$ and a variable $x \in Var^{\forall}(\xi_2)$ occurring in both the body and the head of ξ_2 such that:*

$$\mathsf{In}(\xi_2, x) \sqsubseteq \mathsf{Move}(\Sigma, \mathsf{Out}(\xi_1, y))$$

Definition 12 [59] *A set of TGD's* Σ *is said to be* super-weakly acyclic *iff the trigger relation* $\rightsquigarrow_\Sigma$ *is acyclic. By* SwA *we denote the set of all super-weakly acyclic dependencies.*

Example 16 *Consider* $\Sigma = \{\xi_1, \xi_2\}$ *as in Example 15. The place* $((\xi_1, S, 1), 1)$ *is not unifiable with* $((\xi_2, S, 1), 1)$ *and* $\mathsf{In}(\xi_2, \mathsf{w}) \not\sqsubseteq \mathsf{Move}(\Sigma, \mathsf{Out}(\xi_1, y))$*, thus* $\xi_1 \not\rightsquigarrow_\Sigma \xi_2$*. Similarly,* ξ_2 *does not contain any existential variables and so it follows that* $\xi_2 \not\rightsquigarrow_\Sigma \xi_1$*. As both dependencies do not share common relation names in the head and the body, it follows that* $\xi_1 \not\rightsquigarrow_\Sigma \xi_1$ *and* $\xi_2 \not\rightsquigarrow_\Sigma \xi_2$*. That is the relation* $\rightsquigarrow_\Sigma$ *does not induce any cycle, following that* Σ *is super-weakly acyclic. Moreover, it can be seen that* $\Sigma \notin \mathsf{SD}$*, as between the affected positions* $(R, 1)$ *and* $(S, 2)$ *there exists a cycle through an existential edge in the corresponding propagation graph.*

Marnette [59] showed that the membership problem *Is* $\Sigma \in \mathsf{SwA}$*?* has a polynomial complexity in the size of Σ. The following theorem proves that super-weakly acyclicity is a sufficient condition for the semi-oblivious-chase termination on all instances.

Theorem 17 $\mathsf{SwA} \subset \mathsf{CT}^{\mathsf{std}}_{\forall\forall}$ *and for any* $\Sigma \in \mathsf{SwA}$ *and any instance* I *there exists a polynomial, in size of* I*, that bounds the execution time of the semi-oblivious-chase algorithm with* I *and* Σ*.*

Proof: It follows directly from Theorem 2 and Theorem 5 in [59].

Proposition 14 [59] $\mathsf{WA} \subset \mathsf{SwA}$*.*

Later, Spezzano and Greco [70] also showed that $\mathsf{SD} \subset \mathsf{SwA}$, that is the super-weak acyclic dependencies properly contain all safe dependencies. A nice property of the class SwA is that it is closed in adding atoms in the body of dependencies, thus given a set of TGD's $\Sigma \in \mathsf{SwA}$, then any set of dependencies Σ', obtained from Σ by adding new atoms in the body of any dependencies, remains super-weakly acyclic.

The previous theorem together with Proposition 7 give us the following corollary:

Corollary 5 *Let $\Sigma \in \mathsf{SwA}$ and let I be an instance. Then there exists a polynomial, in the size of I, that bounds the length of every standard-chase sequence of I and Σ.*

The following proposition shows that super-weak acyclicity class is incomparable with the class of dependencies that ensures the oblivious-chase termination on all instances.

Proposition 15 $\mathsf{SwA} \nparallel \mathsf{CT}^{\mathsf{obl}}_{\forall\forall}$.

Proof: Let us consider $\Sigma_1 = \{R(x,y) \to \exists z\ R(x,z)\}$. It is clear that $\Sigma_1 \in \mathsf{SwA}$ but the oblivious-chase algorithm does not terminate with input $I = \{R(a,b)\}$ and Σ_1. Let us now consider $\Sigma_2 = \{S(x), R(x,y) \to \exists z\ R(y,z)\}$ and denote the dependency in Σ_2 by ξ. With this we have $\mathsf{In}(\xi,x) = \{((\xi,R,1),2)\}$, $\mathsf{Out}(\xi,z) = \{((\xi,R,2),2)\}$ and $\mathsf{Move}(\Sigma,\mathsf{Out}(\xi,z)) = \{((\xi,R,2),2),((\xi,R,1),2)\}$, that is $\mathsf{In}(\xi,x) \sqsubseteq \mathsf{Move}(\Sigma,\mathsf{Out}(\xi,z))$. Thus, based on Definition 11, $\xi \rightsquigarrow_{\Sigma_2} \xi$ following that $\Sigma_2 \notin \mathsf{SwA}$. On the other hand it can be observed that the oblivious-chase algorithm terminates for any instance I. This is because the dependency may be obliviously fired only if the first position in R does not contain a null value ∎

This concludes that the super-weakly acyclic class of dependencies ensures that the standard, semi-oblivious, restricted and core-chase algorithms terminate for all input instances.

4.7 Stratification

The stratified model of dependencies was introduced by Deutsch et al. in [23]. This class relaxes the condition imposed by weak acyclicity by stratifying the dependencies and checking for weak acyclicity on each of these strata.

Before presenting the notion of stratification let us introduce a new notation. Given ξ a tgd of the form:

$$\alpha(\bar{x}, \bar{y}) \to \exists \bar{z}\ \beta(\bar{x}, \bar{z})$$

and a vector $\bar{a} = (a_1, \dots, a_n, a_{n+1}, \dots, a_{n+m})$ of constants where $n = |\bar{x}|$ and $m = |\bar{y}|$, by $\xi(\bar{a})$ we denote the following formula without any universally quantified variables:

$$\alpha((a_1, \dots, a_n), (a_{n+1}, \dots, a_{n+m})) \to \exists \bar{z}\ \beta((a_1, \dots, a_n), \bar{z})$$

Definition 13 [23] *Let ξ_1 and ξ_2 be two TGD's , we write $\xi_1 < \xi_2$, if there exist instances I, J and vector $\bar{a} \subseteq dom(J)$, such that*

1. *$I \models \xi_2(\bar{a})$, and*
2. *there exists a standard trigger (ξ_1, h), such that $I \xrightarrow{(\xi_1,h)} J$, and*
3. *$J \not\models \xi_2(\bar{a})$.*

Example 17 *Consider the following two TGD's:*

$$\begin{aligned} \xi_1 &: \quad \forall x, y\ R(x, y) \to S(x) \\ \xi_2 &: \quad \forall x, y\ S(x) \to R(x, x) \end{aligned}$$

For the instance $I = \{R(a, b)\}$ and $\bar{a} = (a)$ we have that $I \models \xi_2(a)$; and for homomorphism $h = \{X/a, Y/b\}$ we have $I \xrightarrow{(\xi_1,h)} J$, where $J = \{R(a, b), S(a)\}$. Because $J \not\models \xi_2(a)$, it follows that $\xi_1 < \xi_2$. On the other hand, $\xi_2 \not< \xi_1$ because for any vector of *constants $\bar{b} = (a, b)$ and instance I such that $I \models \xi_1(\bar{b})$ and for any trigger (ξ, h) such that $I \xrightarrow{(\xi_2,h)} J$, it follows that $J \models \xi_1(\bar{b})$.*

Given two TGD's ξ_1 and ξ_2, the decision problem *Is* $\xi_1 \prec \xi_2$*?* is known to be in NP [23]. To the best of our knowledge there yet is no result proving the lower bound of this problem.

Given a set of TGD's Σ, the *chase graph* associated with Σ is a directed graph $G^C_\Sigma = (V, E)$, where $V = \Sigma$ and $(\xi_1, \xi_2) \in E$ iff $\xi_1 \prec \xi_2$.

Definition 14 [23] *A set of TGD's Σ is said to be* stratified *if the set of dependencies in every cycle in the corresponding chase graph is weakly acyclic. The class of all stratified sets of dependencies is denoted with* Str*.*

In [61] it is shown that stratification does not guarantee that the standard-chase algorithm terminates on all execution branches for all instances (see example bellow).

Example 18 [61] *Consider the following set of TGD's:*

$$\begin{aligned}
\xi_1 &: R(x) \to S(x, x)\\
\xi_2 &: S(x, y) \to \exists z\, T(y, z)\\
\xi_3 &: S(x, y) \to T(x, y), T(y, z)\\
\xi_4 &: T(x, y), T(x, z), T(z, x) \to R(y)
\end{aligned}$$

The set $\Sigma = \{\xi_1, \xi_2, \xi_3, \xi_4\}$ is stratified since $\xi_1 \prec \xi_2$, $\xi_1 \prec \xi_3 \prec \xi_4 \prec \xi_1$; and the set $\{\xi_1, \xi_3, \xi_4\}$ is weakly acyclic. Let us consider instance $I = \{R(a)\}$. The standard-chase algorithm that triggers repeatedly dependencies ξ_1, ξ_2, ξ_3 and ξ_4 never terminates. On the other hand, the chase sequence that never triggers ξ_2 terminates.

Since the stratification does not ensure termination for the standard chase for all instances, the following question arises: *does the stratification sometimes guarantee* *termination of the standard-chase algorithm on some branches for all instances?* The answer to this question is given by the following theorem:

Theorem 18 $\mathsf{Str} \subset \mathsf{CT}^{\mathsf{std}}_{\forall\exists}$ *and* $\mathsf{Str} \nparallel \mathsf{CT}^{\mathsf{std}}_{\forall\forall}$

Proof: The first part of the theorem is proved in [61]. For the strict inclusion part, consider $\Sigma_1 = \{\xi_1 : \ S(x), R(x,y) \to \exists z \ R(y,z)\}$. Of course, there is no doubt that $\Sigma_1 \in \mathsf{CT}^{\mathsf{std}}_{\forall\forall} \subset \mathsf{CT}^{\mathsf{std}}_{\forall\exists}$. On the other hand, $\xi_1 \prec \xi_1$; as for instance $I = \{R(b,a), S(a), S(b)\}$ we have $I \models \xi(a,b)$, $I \xrightarrow{(\xi,h)} J$, where $h = \{X/a, Y/b\}$, $J = I \cup \{R(a,Z),\}$ and $J \nvDash \xi(a,Z)$, thus Σ is not stratified. Example 18 shows a set of TGD's Σ such that $\Sigma \in \mathsf{Str}$ but $\Sigma \notin \mathsf{CT}^{\mathsf{std}}_{\forall\forall}$ ■

From the previous theorem and Theorem 10 it follows that the core-chase algorithm terminates for all instances over a set of stratified TGD's.

Meier et al. [61] improved the stratification notion in order to ensure also that the standard-chase algorithm terminates on all execution branches for all instances. For this they changed the $\prec$ relation as follows:

Definition 15 [61] *Let* ξ_1 *and* ξ_2 *be two TGD's. We write* $\xi_1 \prec_c \xi_2$*, if there exist instances* I*,* J *and vector* $\bar{a} \subseteq dom(I)$*, such that*

1. $I \models \xi_2(\bar{a})$*, and*
2. *there exists an oblivious trigger* (ξ_1, h)*, such that* $I \xrightarrow{*(\xi_1,h)} J$*, and*
3. $J \nvDash \xi_2(\bar{a})$*.*

Given a set of TGD's Σ, the *c-chase graph* associated with Σ is a directed graph $G^{CC}_{\mathrm{C}} = (V, E)$. With $V = \Sigma$ and $(\xi_1, \xi_2) \in E$ iff $\xi_1 \prec_{\mathrm{c}} \xi_2$. A set of TGD's Σ is said to be *c-stratified* if the set of dependencies in every cycle in the c-chase graph is weakly acyclic. The set of all c-stratified dependencies is denoted by CStr.

Theorem 19 [61] $\mathsf{CStr} \subset \mathsf{CT}^{\mathbf{std}}_{\forall\forall}$ *and for any* $\Sigma \in \mathsf{CStr}$ *and any instance* I *there exists a polynomial, in the size of* I*, that bounds the execution time of the standard-chase algorithm with* I *and* Σ*.*

The previous theorem showed that CStr ensures termination for the standard chase. We can now extend this result by showing that CStr ensures the termination of the semi-oblivious-chase algorithm.

Theorem 20 $\mathsf{CStr} \subset \mathsf{CT}^{\mathbf{sobl}}_{\forall\forall}$ *and* $\mathsf{CStr} \nparallel \mathsf{CT}^{\mathbf{obl}}_{\forall\forall}$*.*

Proof: Let us first prove that $\mathsf{CStr} \subseteq \mathsf{CT}^{\mathbf{sobl}}_{\forall\forall}$. For this, consider a set of TGD's $\Sigma \in \mathsf{CStr}$. Let us now consider the semi-enrichment $\tilde{\Sigma}$. We will show that if $\xi_1 \prec_{\mathrm{c}} \xi_2$ for $\xi_1, \xi_2 \in \Sigma$, then $\tilde{\xi}_1 \prec_{\mathrm{c}} \tilde{\xi}_2$ for $\tilde{\xi}_1, \tilde{\xi}_2 \in \tilde{\Sigma}$. Let I, J be two instances and $\bar{a}$ and $\bar{b}$ be the vectors such that $I \models \xi_2(\bar{a})$, $I \xrightarrow{*(\xi_1,h)} J$ for some homomorphism h and $J \not\models \xi_2(\bar{b})$. Consider instance $I' = I \cup \{H_2(\bar{a})\}$, where H_2 is the relation symbol introduced by the semi-enrichment for ξ_2. By abusing the notation by $H_2(\bar{a})$ we denote the tuple over H_2 that contains the values from $\bar{a}$ that occurs in both the body and the head of ξ_2. Let us also consider instance $J' = J \cup \{H_1(h(\bar{x}))\}$, where H_1 is the relational symbol introduced by the semi-enrichment for ξ_1 and $\bar{x}$ is the vector of universally quantified variables that occurs both in the body and the head of ξ_1. It is easy to observe that $I' \models \tilde{\xi}_2(\bar{a})$, $I' \xrightarrow{*(\tilde{\xi}_1,h)} J'$ for homomorphism h, and $J' \not\models \tilde{\xi}_2(\bar{b})$, thus $\tilde{\xi}_1 \prec_{\mathrm{c}} \tilde{\xi}_2$. From this it follows that $\tilde{\Sigma}$ has an isomorphic c-chase graph with Σ (note that the same result holds for the enrichment $\widehat{\Sigma}$). Also, we know that the dependency graphs corresponding to $\tilde{\Sigma}$ and Σ are isomorphic as well (this property does not hold for the enrichment $\widehat{\Sigma}$). Thus $\Sigma \in \mathsf{CStr}$ implies that $\tilde{\Sigma} \in \mathsf{CStr}$. Because we know from Theorem 19 that $\mathsf{CStr} \in \mathsf{CT}^{\mathbf{std}}_{\forall\forall}$, it follows that $\tilde{\Sigma} \in \mathsf{CT}^{\mathbf{std}}_{\forall\forall}$. On the other hand, Theorem 5 shows that if $\tilde{\Sigma} \in \mathsf{CT}^{\mathbf{std}}_{\forall\forall}$ then $\Sigma \in \mathsf{CT}^{\mathbf{sobl}}_{\forall\forall}$. Thus it follows that $\mathsf{CStr} \subseteq \mathsf{CT}^{\mathbf{sobl}}_{\forall\forall}$. For the strict part of the inclusion, consider $\Sigma_1 = \{\xi_1 : \ S(x), R(x,y) \rightarrow \exists z\ R(y,z)\}$. Similarly to the proof from 18, it can be shown that $\Sigma_1 \notin \mathsf{CStr}$, but $\Sigma_1 \in \mathsf{CT}^{\mathbf{obl}}_{\forall\forall} \subset \mathsf{CT}^{\mathbf{sobl}}_{\forall\forall}$. It remained to show only that $\mathsf{CStr} \nsubseteq \mathsf{CT}^{\mathbf{obl}}_{\forall\forall}$. For this consider $\Sigma_2 = \{\xi : R(x,y) \to \exists z\ R(y,z)\}$. For this set of dependencies we know from the previous proofs that $\Sigma_2 \in \mathsf{WA} \subset \mathsf{CStr}$ and $\Sigma_2 \notin \mathsf{CT}^{\mathbf{obl}}_{\forall\forall}$ ■

The problem of checking if a set of Σ is in Str (or CStr) is known to be in coNP [23] ([61]). To the best of our knowledge the lower bound is still an open problem.

The following Theorem gives the relationship between CStr and the classes WA, SD and SwA.

Theorem 21 $\mathsf{WA} \subset \mathsf{CStr}$, $\mathsf{SD} \nparallel \mathsf{CStr}$ [61] *and* $\mathsf{SwA} \nparallel \mathsf{CStr}$ [70]

4.8 Inductively Restricted Dependencies

Another class of dependencies that guarantees the standard-chase termination is the inductively restricted set of dependencies. Note that the stratification method lifts the weakly acyclic class of dependencies to the class of c-stratified dependencies. The inductively restricted class generalizes the stratification method while still keeping the termination property for the standard-chase algorithm. This generalization is done using the so-called *restriction systems* [61]. With the help of the restriction systems, Meier et al. [61] define the new sufficient condition called *inductive restriction* which guarantees the standard-chase algorithm termination for all instances on all branches. From this condition a new hierarchy of classes of dependencies is revealed with the same termination property, called the *T-hierarchy.* Note that the inductive restriction condition presented here is given from the erratum [62] and not from [61], where the presented condition, as mentioned in the erratum, does not guarantee the standard-chase termination on all execution branches for all instances.

Let Σ be a set of TGD's, I an instance and A a set of null, $A \subseteq \Delta_{\mathsf{N}}$. The set of all positions (R,i) such that there exists a tuple in I which contains a null from A in position (R,i) and which is denoted by *null-pos*(A,I) .

Similarly with "$<$" relation for the stratified dependencies, the binary relation "$<_P$" is defined for the inductive restriction condition for P a set of positions.

Definition 16 [61] *Let Σ be a set of TGD's and P a set of positions from Σ. Let ξ_1, ξ_2 be two dependencies in Σ. It is said that $\xi_1 <_P \xi_2$, if there exist instances I, J and vector $\bar{a}$ such that:*

1. *$I \models \xi_2(\bar{a})$, and*
2. *there exists an oblivious trigger (ξ_1, h), such that $I \xrightarrow{*(\xi_1,h)} J$, for a homomorphism h, and*
3. *$J \not\models \xi_2(\bar{a})$, and*
4. *there exists $X \in \bar{a} \cap \Delta_{\mathsf{N}}$ in head$(\xi_2(\bar{a}))$ such that null-pos$(\{X\},I) \subseteq P$.*

Example 19 *Consider Σ containing a single TGD $\xi: R(x,y) \to \exists z\, R(y,z)$. Example 3 showed that there exists an instance I such that standard-chase algorithm does not terminate for I and Σ. Consider $I = \{R(a,b)\}$ and vector $\bar{a} = (b, X_1)$. It is easy to see that condition 1 from the previous definition is satisfied, thus $I \models \xi(\bar{a})$ as there is no homomorphism from body(ξ) to I. On the other hand, for homomorphism $h = \{X/b, Y/X_1\}$, we have $I \xrightarrow{*(\xi_1,h)} J$, where $J = \{R(a,b), R(b,X)\}$, thus the second condition from the previous definition is fulfilled as well. The third condition is also met as $J \not\models \xi(\bar{a})$. The 4th condition is also satisfied. For this consider the null $X_1 \in \bar{a}$, then we have $\xi(\bar{a})$ representing formula $R(a,X_1) \to \exists z R(X_1,z)$. Thus, X_1 occurs in head$(\xi(\bar{a}))$. On the other hand, null-pos$(\{X_1\},I) = \varnothing$, because instance I does not contain any nulls, that is for any set P, null-pos$(\{X\},I) \subseteq P$. Thus $\xi <_P \xi$, for any set P of positions.*

Definition 17 [61] *Let P be a set of positions and ξ a TGD. aff-cl(ξ, P) is the set of positions (R, i) from the head of ξ such that:*

1. *for all $x \in Var^{\forall}(\xi)$, with x occurs in (R, i), x occurs in the body of ξ only in positions from P, or*

2. *position (R, i) contains a variable $x \in Var^{\exists}(\xi)$.*

Considering the dependency from Example 19, we have *aff-cl*$(\xi, P) = \{(R,1),(R,2)\}$, where $P = \{(R,2)\}$. Given a set of dependencies Σ, by *pos*(Σ) is denoted the set of all positions in Σ.

Definition 18 [61] *A* 2-restriction system *is a pair $(G(\Sigma), P)$, where $G(\Sigma)$ is a directed graph (Σ, E) and $P \subseteq pos(\Sigma)$ such that:*

1. *for all $(\xi_1, \xi_2) \in E$, aff-cl$(\xi_1, P) \cap pos(\Sigma) \subseteq P$ and aff-cl$(\xi_2, P) \cap pos(\Sigma) \subseteq P$,*

2. *for all $\xi_1 \prec_P \xi_2$, $(\xi_1, \xi_2) \in P$.*

A 2-restriction system is *minimal* if it is obtained from $((\Sigma, \varnothing), \varnothing)$ by a repeated application of constraints 1 and 2, from the previous definition, such that P is extended only by those positions that are required to satisfy condition 1. Let us denote *part*$(\Sigma, 2)$ to contain the sets of all strongly connected components in minimal 2-restriction system.

Example 20 *Considering Σ from Example 19, the minimal 2-restriction system is computed as follows. Consider pair $((\{\xi\}, \varnothing), \varnothing)$. We previously showed that $\xi \prec_P \xi$, for any set of positions P. And by particularization we have $\xi \prec_{\varnothing} \xi$. Thus, we add edge (ξ, ξ) to E. Let us remember that the condition 1 from definition 18 stipulates that aff-cl$(\xi, \varnothing) = \{(R,2)\}$. Therefore, we add position $(R,2)$ to P. By repeating this process with $P = \{(R,2)\}$, we also add to P the position $(R,1)$. Thus, the minimal 2-restriction system is $((\Sigma, \{(\xi,\xi)\}), \{(R,1),(R,2)\})$. The only connected component in this restriction system is $\{\xi\}$.*

Meier et al. provide in [61] a simple algorithm to compute the set $part(\Sigma, 2)$.

Definition 19 [61] *A set Σ of TGD's is called* inductively restricted *if every stratum $\Sigma' \in part(\Sigma, 2)$ is safe dependency. The set of all inductively restricted dependencies is denoted with* IR.

Note that the notion of inductively restricted set of dependency is defined very similarly to the classes Str and CStr, in the sense that both classes consider stratification of their set of dependencies and then test each stratum separately. In the case of Str and CStr, each stratum is checked to satisfy the weak acyclicity condition. For the class IR, each stratum is verified to satisfy the safe dependency condition. Similarly to the stratification case, the membership problem (*is* $\Sigma \in$ IR*?*) is known to be in coNP [61]. To the best of our knowledge no lower bound was proved. Next, let us review the position of the IR class compared to the presented classes.

Theorem 22 [61] SD⊂IR *and* Str⊂IR.

From the second part of the theorem and definition of CStr class it follows that we also have CStr⊂IR.

Theorem 23 [70] SwA $\nparallel$ IR.

The following example presents a set of dependencies in IR but not stratified.

Example 21 [61] *Consider the following set of TGD's Σ:*

$$\begin{aligned}
\xi_1 &: \quad S(x), E(x,y) \rightarrow E(y,x) \\
\xi_2 &: \quad S(x), E(x,y) \rightarrow \exists z\, E(y,z), E(z,x)
\end{aligned}$$

It can be easily observed that Σ is neither stratified nor safe, but inductively restricted.

Theorem 24 [61] *$\mathsf{IR} \subset \mathsf{CT}^{\mathsf{std}}_{\forall\forall}$ and for any $\Sigma \in \mathsf{IR}$ and any instance I there exists a polynomial, in size of I, that bounds the execution time of the standard-chase algorithm with I and Σ.*

Meier, Schmidt and Lausen [61] observed that the inductive restriction criterion can be extended to form a hierarchy of classes that ensures the standard-chase termination on all branches for all instances. Intuitively, the lowest level of this hierarchy, noted $T[2]$, is the class of inductively restricted dependencies, thus $T[2] = \mathsf{IR}$. Level $T[k]$, $k > 2$, is obtained by extending the binary relation $\prec_{\mathrm{P}}$ to a k-ary relation $\prec_{k,\mathrm{P}}$. Based on this new relation the set $part(\Sigma, k)$ is computed similarly to $part(\Sigma, 2)$. In [61] is given the algorithm that computes $part(\Sigma, k)$. For all $k \leq 2$, it is shown that $T[k] \subset T[k+1]$ and also that the complexity of the membership problem remains in coNP for a fixed k.

Similarly to the reduction used in the proof of Theorem 20, it can be shown that for any $k \geq 2$ and for any $\Sigma \in T[k]$ the semi-enrichment $\tilde{\Sigma} \in T[k]$. On the other hand, this may not hold for the enrichment $\widehat{\Sigma}$. From this we have the following result:

Theorem 25 *For any $k \geq 2$ we have $T[k] \subset \mathsf{CT}^{\mathsf{sobl}}_{\forall\forall}$ and $T[k] \nparallel \mathsf{CT}^{\mathsf{obl}}_{\forall\forall}$.*

More recently, the $T[k]$ hierarchy of classes was extended by Meier et al. [63] to the $\forall\exists - T[k]$ hierarchy of classes that ensures the standard-chase termination on at least one execution branch. Without giving a formal description of this new class hierarchy, let us just present an example of dependency set from [63] that is in $\forall\exists - T[2]$ but does not belong to any of the classes from the $T[k]$ hierarchy.

Example 22 [63] *Let Σ be the following set of TGD's:*

$$\begin{aligned}
\xi_1 :&\quad R(x) \to S(x,x)\\
\xi_2 :&\quad S(x,y) \to \exists z\, T(y,z)\\
\xi_3 :&\quad S(x,y) \to T(x,y), T(y,x)\\
\xi_4 :&\quad T(x,y), T(x,z), T(z,x) \to R(y)\\
\xi_5 :&\quad T(x,y) \to E(x,y)\\
\xi_6 :&\quad U(x), E(x,y) \to E(y,x)\\
\xi_7 :&\quad U(x), E(x,y) \to \exists z\, E(y,z), E(z,x)
\end{aligned}$$

It is obvious that $\Sigma \in \mathsf{CT}^{\mathbf{std}}_{\forall\exists}$*, as one may apply a terminating chase sequence on* $\Sigma' = \{\xi_1, \xi_2, \xi_3, \xi_4\}$ *and then apply the rest of the rules from* Σ*. Note that, by applying the rest of the rules, it will not affect the satisfiability of* Σ'*. Also note that both chase sequences applied are terminating for all instances. Even more, in [63] it is shown that* Σ *does not belong to any class from the* $T[k]$ *hierarchy and that* $\Sigma \in \forall\exists - T[2]$.

4.9 Comparison between Classes

Before concluding this chapter dedicated to present different classes of dependencies which ensure the termination of one or more chase algorithms, let us review how these classes compare to each other and for which chase variation algorithm they ensure termination.

From the practical point of view one of most important questions is regarding the complexity of testing if a set of dependencies belongs to a given class. Figure 4.6 illustrates the complexity for the following problem: given C a class of dependency sets and Σ a set of dependencies *Is* $\Sigma \in \mathsf{S}$*?*, note that for the classes with membership problem in NP there was not yet proved a lower bound.

Next let us review for which algorithm these classes ensure termination given the

	SW	RA	WA	SD	SwA	Str	CStr	IR/$T[k]$
Complexity	P	P	P	P	P	in NP	in NP	in NP

Figure 4.6: Set order between the presented classes

classes presented in this chapter. In Figure 4.7 we present each class of dependency sets described in this section together with the termination criteria introduced in Chapter 3. The symbol "+" mentiones that the given class is sufficient for the chase algorithm termination with the given criterion. Note also that there are not mentioned all the termination criteria because of the following equalities: $\mathsf{CT}^{\mathbf{obl}}_{\forall\forall} = \mathsf{CT}^{\mathbf{obl}}_{\forall\exists}$, $\mathsf{CT}^{\mathbf{sobl}}_{\forall\forall} = \mathsf{CT}^{\mathbf{sobl}}_{\forall\exists}$, $\mathsf{CT}^{\mathbf{res}}_{\forall\forall} = \mathsf{CT}^{\mathbf{res}}_{\forall\exists}$ and $\mathsf{CT}^{\mathbf{core}}_{\forall\forall} = \mathsf{CT}^{\mathbf{core}}_{\forall\exists}$.

	$\mathsf{CT}^{\mathbf{obl}}_{\forall\forall}$	$\mathsf{CT}^{\mathbf{sobl}}_{\forall\forall}$	$\mathsf{CT}^{\mathbf{std}}_{\forall\forall}$	$\mathsf{CT}^{\mathbf{res}}_{\forall\forall}$	$\mathsf{CT}^{\mathbf{std}}_{\forall\exists}$	$\mathsf{CT}^{\mathbf{core}}_{\forall\forall}$
SW	+	+	+	+	+	+
RA	+	+	+	+	+	+
WA		+	+	+	+	+
SD		+	+	+	+	+
SwA		+	+	+	+	+
Str					+	+
CStr		+	+	+	+	+
IR/$T[k]$		+	+	+	+	+

Figure 4.7: Classes ensuring termination for different chase algorithms

For an easier view we represent the set based comparison between different classes of dependency as a Hasse diagram in Figure 4.8. Before concluding this section we need to mention that more recently Greco et al. [41] extended the classes of dependencies that ensure the standard-chase termination to new larger classes based on the stratification method.

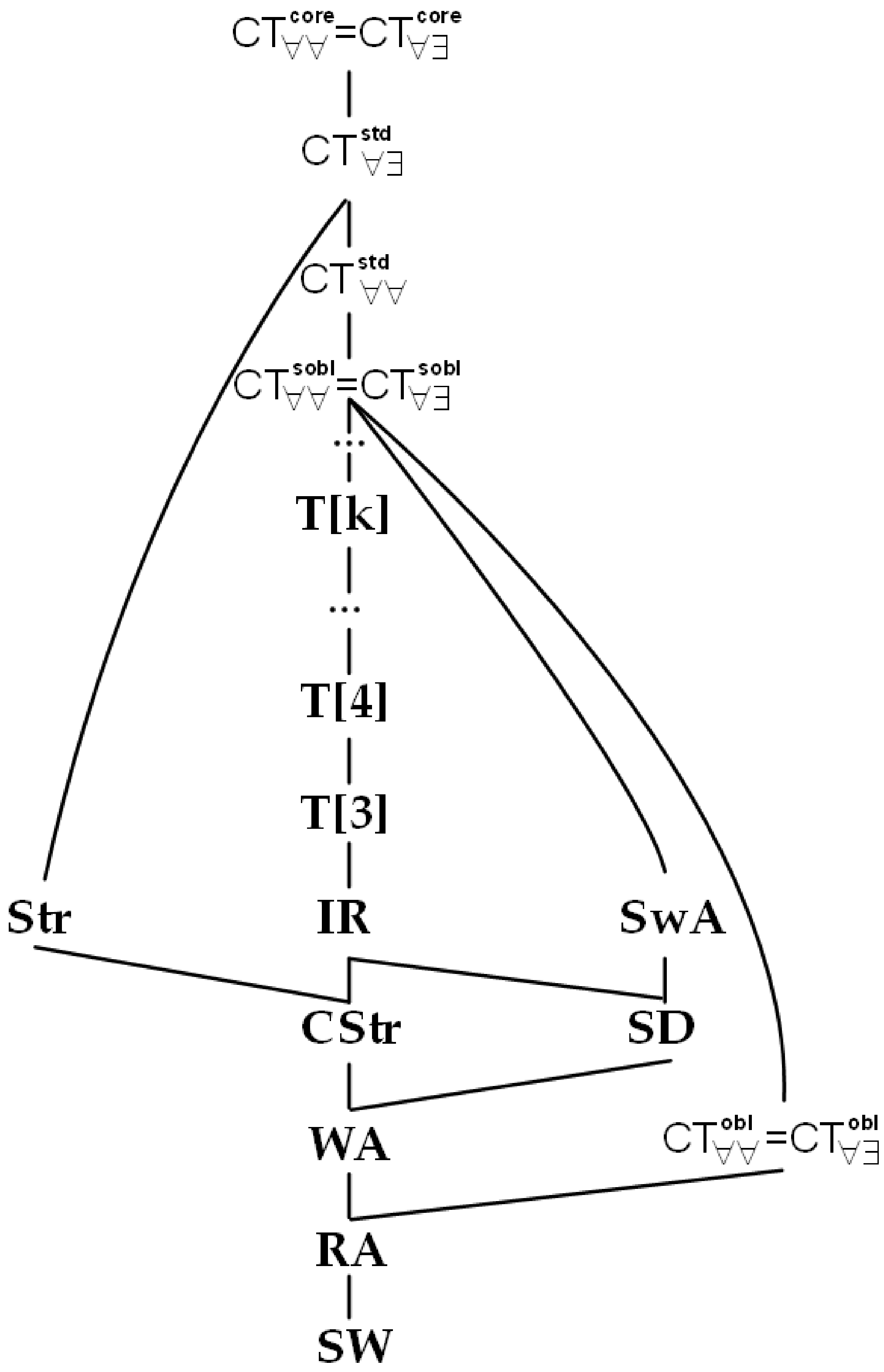
$CT^{core}_{\forall\forall}=CT^{core}_{\forall\exists}$
$CT^{std}_{\forall\exists}$
$CT^{std}_{\forall\forall}$
$CT^{sobl}_{\forall\forall}=CT^{sobl}_{\forall\exists}$
...
T[k]
...
T[4]
T[3]
Str
IR
SwA
CStr
SD
WA
$CT^{obl}_{\forall\forall}=CT^{obl}_{\forall\exists}$
RA
SW

Figure 4.8: Hasse diagram corresponding to classes of dependency sets

4.10 The Rewriting Approach

Spezzano and Greco [70] noticed that all the previous classes may be properly extended using a rewriting technique similarly to the Magic Sets rewriting method [12]. Intuitively, if **T** is one of the classes WA, SD, SwA, Str and CStr, then instead of directly checking if a set of dependencies $\Sigma \in \mathbf{T}$, we check if $\text{Adn}(\Sigma) \in \mathbf{T}$, where $\text{Adn}(\Sigma)$ is an adornment based rewriting of Σ such that, for any instance I, the universal solutions under Σ and I are the "same" (with some schema transformations) with the set of universal solutions under $\text{Adn}(\Sigma)$ and I. Where the adornment of a predicate p of arity m is a string of length m over the alphabet $\{b, f\}$. An adorned atom is of the form $p^{\alpha_1,\alpha_2,\dots,\alpha_m}(x_1, x_2, \dots, x_m)$; if $\alpha_i = b$, then the variable x_i is considered bounded, otherwise the variable is considered free.

We will present this method by using an example from [70]. Consider the following set of dependencies $\Sigma = \{\xi_1, \xi_2\}$:

$$\begin{aligned}
\xi_1 &: \quad N(x) \to \exists y\, E(x, y) \\
\xi_2 &: \quad S(x), E(x, y) \to N(y)
\end{aligned}$$

The affected positions in Σ are $(E,1)$,$(E,2)$ and $(N,1)$. Because the propagation graph corresponding to Σ contains a cycle through an existential edge involving positions $(N,1)$ and $(E,2)$, it follows that $\Sigma \notin$SD.

Let us now construct the set of dependencies $\text{Adn}(\Sigma)$ as follows:

1. For all predicate symbols p of arity m in Σ add the TGD:

$$\forall x_1, x_2, \dots, x_m\, p(x_1, x_2, \dots, x_m) \to p^{\alpha_1,\alpha_2,\dots,\alpha_m}(x_1, x_2, \dots, x_m)$$

   where, for all positive $i \in [m]$, $\alpha_i = b$.

   In our example Σ contains the following relational symbols $\{E, S, N\}$, that is we add to Adn(Σ) the following set of TGD's:

$$\begin{aligned} \xi_1' &: E(x,y) \rightarrow E^{b\,b}(x,y) \\ \xi_2' &: N(x) \rightarrow N^{b}(x) \\ \xi_3' &: S(x) \rightarrow S^{b}(x) \end{aligned}$$

2. Repeat to create new adornment predicate symbols based on the existing dependencies, until none can be added. That is a variable in the head is marked as bounded (free) and if it occurs only bounded (free) places in the body. All existential variables in the head are marked as free.

   Returning to the example, by using ξ_1 from Σ and the new adornment N^{b}, we add the following dependency to Adn(Σ)

$$\xi_4' : N^{b}(x) \rightarrow \exists y\, E^{b\,f}(x,y)$$

   Similarly based on TGD ξ_2 from Σ and new adornments S^{b} and $E^{b\,b}$, we add the following dependency to Adn(Σ)

$$\xi_5' : S^{b}(x), E^{b\,b}(x,y) \rightarrow N^{b}(y)$$

   Repeating this process, we add the following TGD's to Adn(Σ):

$$\begin{aligned} \xi_6' &: S^{b}(x), E^{b\,f}(x,y) \rightarrow N^{f}(y) \\ \xi_7' &: N^{f}(x) \rightarrow \exists y\, E^{f\,f}(x,y) \end{aligned}$$

   After this point no other adornments can be created.

3. Finally, for each of the adornment predicate p^{α} in Adn(Σ) add a new dependency in Adn(Σ) that "copies" p^{α} to a new $\hat{p}$ predicate symbol. For the previous

example, the following new dependencies are added:

$$\begin{aligned}
\xi'_8 &: N^{b}(x) \rightarrow \hat{N}(x)\\
\xi'_9 &: N^{f}(x) \rightarrow \hat{N}(x)\\
\xi'_{10} &: S^{b}(x) \rightarrow \hat{S}(x)\\
\xi'_{11} &: E^{b\,b}(x,y) \rightarrow \hat{E}(x,y)\\
\xi'_{12} &: E^{b\,f}(x,y) \rightarrow \hat{E}(x,y)\\
\xi'_{13} &: E^{f\,f}(x,y) \rightarrow \hat{E}(x,y)
\end{aligned}$$

Greco and Spezzano [70] prove that for all instances I, if J is a finite instance obtained by the standard-chase algorithm on some branch with I and Σ, then $\hat{J}$ is a finite instance obtained by the standard-chase algorithm on some branch with I and Adn(Σ), where $\hat{J}$ is the restriction of the chase result to the hatted predicates from Σ.

Returning to the example, it can be noted that the set Adn(Σ) is safe. Thus, even if the set Σ was not safe, the standard chase will terminate on all branches on Σ with any instances.

Theorem 26 [70] *Let* $\mathbf{T}$ *be one of the classes* WA, SD, SwA, Str *and* CStr, *let* Σ *be a set of TGD's, and let* I *be an instance. Then, if Adn*(Σ) $\in \mathbf{T}$, *then* $\Sigma \in \mathsf{CT}^{\mathbf{std}}_{\forall\forall}$ *and there exists a polynomial, in the size of* I, *that bounds the length of every standard-chase sequence of* I *and* Σ.

Even more, Spezzano and Greco [61] proved that these rewritings strictly extend the previous classes of dependencies.

Theorem 27 [61] *Let* $\mathbf{T}$ *be one of the classes* WA, SD, SwA, Str, CStr *and let* Adn$\mathbf{T}$ *be the set of all* Σ *such that Adn*(Σ) $\in \mathbf{T}$. *Then,* $\mathbf{T} \subset$ Adn$\mathbf{T}$.

This rewriting method is further improved by Greco et al. [41] by indexing the adornment used to specify the free positions. This will ensure that we may equate only variables that have the adornment with the same index.

Chapter 5

Data Exchange, Repair and Correspondence

In this chapter we will review three of the problems that make use of the chase procedure in the process of finding “solutions“. First we will review the Data Exchange problem and see what type of solutions are computed by the standard-chase algorithm, next we will show how this result is improved by using the extended-core-chase algorithm. The third section will review the Data Repair problems and present how the standard-chase procedure helps in solving some of these problems in polynomial time for special classes of dependencies. Finally we formally introduce the Data Correspondence setting and check two of the main problems associated with this setting, that is *solution-existence* and *solution-check* problems; and we will show how the standard-chase algorithm helps to get tractability results for some cases. Part of the results of this section was first published in [37].

5.1 Data Exchange

For a complete and coherent introduction to the application of the chase procedure in data exchange, we need first to present the notion of universal models and their relation with the chase algorithm. This will be followed by a short review of the data exchange problem and the link between universal models and query answering in data exchange. In the final part of this section we will review the query answering problem in case there are no universal models.

5.1.1 Universal Models

Universal models play an important role in many database problems, starting with the testing for conjunctive query containment under functional and inclusion dependencies [50], to the most recent problems of data integration [53], data exchange [27] and query answering over ontologies [18]. Where the notion of model is defined as follows:

Definition 20 [23] *Given an instance I and Σ a set of dependencies, an instance J (finite or infinite) is said to be a* model *of I and set Σ if $J \models \Sigma$, and there exists a homomorphism from I to J, that is $I \to J$.*

Example 23 *Consider instance $I = \{R(a,b), R(b,c)\}$ and the set of dependencies Σ containing one single tuple generating dependency: $R(x,y), R(y,z) \to R(x,z)$. Under these settings the following instance $J = \{R(a,b), R(b,c), R(a,c)\}$ is a model of I and Σ. So the instance $J_1 = J \cup \{R(a,X)\}$, with X a null value from Δ_{N}. On the other hand, instance $J_2 = \{R(a,b), R(a,c)\}$ is not a model of I and Σ. This happens even if $J_2 \models \Sigma$, because there is no homomorphism from I to J_2. Considering instance $J_3 = \{R(a,b), R(b,c), R(b,a)\}$, there is a homomorphism from I into J_3, but $J_3 \not\models \Sigma$, thus J_3 is not a model of I and Σ.*

As it can be seen from in previous example, in the general case there may be an infinite number of models of I and Σ. Still, some of these models are more general than the others in the sense that they have a homomorphism into all the other models. Such models are called universal. This brings us to the following definition:

Definition 21 [23] *An instance*[1] *U is said to be a* weak universal model *of I and Σ if U is a model of I and Σ, and for any finite model J of I and Σ, we have $U \to J$. If we also have that $U \to J$ for all infinite models J of I and Σ, then U is said to be a* strong universal model *or simply a* universal model .

Example 24 *Considering instance I and dependency Σ from example 23, it is easy to observe that both instances J and J_1 are strong universal models. On the other hand, the model $J_3 = \{R(a,b), R(b,c), R(a,c), R(a,a)\}$ is neither a strong or weak universal model because there does not exist a homomorphism from J_3 to model J.*

The following theorem links the instances returned by the standard-chase procedure to universal models.

Theorem 28 [23; 27] *Let I be an instance and Σ a set of TGD's and EGD's. Then any finite instance returned by the standard-chase algorithm is a universal model of I and Σ.*

Intuitively, the theorem says that whenever the standard chase terminates and it does not fail, it gives a universal model of I and Σ. Thus, if $chase^{\mathbf{std}}_{\Sigma}(I) \neq \perp$, then $chase^{\mathbf{std}}_{\Sigma}(I)$ is a universal model of I and Σ.

The following proposition ensures that the previous property holds for other chase algorithms as well.

[1]Note that in this dissertation by *instance* we mean finite instance if not specified otherwise

Proposition 16 *If $chase^{*}_{\Sigma}(I) \neq \bot$, then $chase^{*}_{\Sigma}(I)$ is a universal model of I and Σ, where $*$ is one of the following chase variations: oblivious, semi-oblivious, restricted and core.*

Proof: In Chapter 3 we showed that for oblivious, semi-oblivious and restricted-chase variations represented here by $*$, if $chase^{*}_{\Sigma}(I) \neq \bot$, then $chase^{\mathbf{std}}_{\Sigma}(I) \neq \bot$. Also, in the same chapter it was shown that if $chase^{*}_{\Sigma}(I) \neq \bot$, then it is that $chase^{*}_{\Sigma}(I) \vDash \Sigma$ and $chase^{*}_{\Sigma}(I) \leftrightarrow chase^{\mathbf{std}}_{\Sigma}(I)$. This property also holds for the core chase as shown in Theorem 29. The direct consequence is that $chase^{*}_{\Sigma}(I)$ is a universal model of I and Σ■

The previous result ensures that the chase procedure computes universal models of its input. Naturally the following question raises: *are these algorithms also complete in finding universal models?* The following example shows that we have a negative answer to this question in case of the standard chase.

Example 25 *Let us consider the same instance $I = \{R(a,b)\}$ and $\Sigma = \{\xi_1, \xi_2\}$ a set of TGD's as in Example 8:*

$$\begin{aligned} \xi_1 &: \forall x, y\, R(x,y) \rightarrow \exists z\, R(y,z) \\ \xi_2 &: \forall x, y\, R(x,y), R(y,z) \rightarrow R(y,y) \end{aligned}$$

As shown in Example 8, there is no terminating branch for the standard-chase algorithm, but there exists universal model $J = \{R(a,b), R(b,b)\}$ of I and Σ. Thus the standard chase is not complete in finding universal models.

The result below shows that there exists one chase variation, namely the core chase, that is complete in finding universal models.

Theorem 29 [23] *Let I be an instance and Σ a set of TGD's and EGD's. Then there exists a universal model of I and Σ iff the core-chase algorithm terminates on input I and Σ. Even more, $chase_{\Sigma}^{\mathbf{core}}(I)$ is a universal model of I and Σ.*

From the definition of the universal models it follows that all universal models are also weak universal models. The following example shows that the converse does not always hold.

Example 26 [23] *Let us consider instance $I = \{T(a)\}$ and set $\Sigma = \{\xi_1, \xi_2, \xi_3\}$ of TGD's:*

$$\begin{aligned}
\xi_1 &: T(x) \to \exists y, z\, E(y,z) \\
\xi_2 &: E(x,y) \to \exists z\, E(y,z) \\
\xi_3 &: E(x,y), E(y,z) \to E(x,z)
\end{aligned}$$

Considering that relation E contains edges of a graph, clearly all models have an infinite walk in the graph. This means that for every finite model it has a cycle in the corresponding graph. From this and ξ_3, it follows that any finite model has a self loop. Then the instance $J = \{T(a), E(X,X)\}$ is a model of I and Σ containing a self loop. Consequently, J is a weak universal model of I and Σ. On the other hand, the transitive closure of an infinite path also satisfies Σ, however no finite instance with cycle has a homomorphism into it. This means that J is not a strong universal model of I and Σ.

We cannot conclude this section on universal models without emphasizing the result of Deutsch et al. [23] showing that it is undecidable to say if an instance U is a strong (weak) universal model of instance I and Σ a set of tgds. Even more, they demonstrated that there is no complete chase based procedure for finding weak universal models.

5.1.2 Solutions in Data Exchange

Data exchange is an old database problem that earned more formal treatment only recently. More precisely, data exchange is the problem of transforming data structured under a source schema to data structured under a different target schema. The concept of mappings between schemata was introduced by Bernstein et al. in [14] and by Miller et al. [65] for relational schemata. Let $\mathbf{S}$ and $\mathbf{T}$ be two disjoint schemata, which we call the *source schema* and the *target schema*, respectively. A *data exchange mapping* (or simply mapping) M from source $\mathbf{S}$ to target $\mathbf{T}$ is a set of pairs (I, J), where I is an instance over schema $\mathbf{S}$ and I is an instance over schema $\mathbf{T}$. Thus

$$M = \{(I, J) \mid I \in Inst(\mathbf{S}) \text{ and } J \in Inst(\mathbf{T})\}. \tag{5.1}$$

In most of the practical applications the data exchange mapping M is specified as a tuple $(\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$, where $\mathbf{S}$ represents the source schema, $\mathbf{T}$ represents the target schema, Σ_{st} is a set of constraints representing the relationship between the source and target schema, and Σ_t represents a set of constraints over the target schema. Thus, if $I \in Inst(\mathbf{S})$ and $J \in Inst(\mathbf{T})$, then $(I, J) \in M$ iff $I \cup J \models \Sigma_{st} \cup \Sigma_t$. In this dissertation if not mentioned otherwise, we will consider a mapping specified as previously.

Definition 22 *Given a data exchange mapping $M = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ and instance I over the source schema $\mathbf{S}$, the* data exchange problem *is to find instances J, over the target schema $\mathbf{T}$, such that $I \cup J$ is a model for I and $\Sigma_{st} \cup \Sigma_t$. An instance J with the previous properties is called a* solution *to the data exchange problem, or simply a solution.*

Note that the solution to the data exchange problem is not necessarily a ground instance. By $Sol_M(I)$ is denoted the set of all ground data exchange problem solution for a mapping M and source instance I.

The data exchange problem was first formalized by Fagin et al. in their seminal paper [27]. Most of the data exchange problems considered have Σ_{st} specified by a set of TGD's and Σ_t specified by a set of TGD's and EGD's. The mapping over this classes of dependencies covers most of the practical mappings. From now on, if not mentioned otherwise, we assume that the data exchange dependencies are of this format.

As there may be an infinite number of solutions to a given data exchange problem, a natural question raises: *which solution, or finite set of solutions, should be materialized on the target?* Unfortunately, there is no simple answer to this question, because it also depends on the use of the solutions. In this subsection, we consider one of the common cases when solutions are used to get certain answers for a set of union of conjunctive queries (UCQ) over the target instance. This can be formalized by the following definition:

Definition 23 *Let $M = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$, also let I be a source instance and q a query in* UCQ *over schema $\mathbf{T}$. The certain answer of q for I under M is defined as*

$$certain_M(q, I) =^{def} \bigcap_{J \in Sol_M(I)} Q(J)$$

Fagin et al. [27] showed that the *universal solution* is a good candidate for materialization in data exchange problem under the certain UCQ answer semantics. For more clarity, an instance J is a universal solution for source instance I and data exchange setting $M = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ if J is a solution for I and M and for all solutions K for I and M we have $J \to K$. It can be noted that a strong correlation exists between the notion of data exchange solution and that of model, and correspondingly between the notion of universal solution and universal model. Thus, any solution for a data exchange mapping $M = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ and instance I is the restriction over the target schema $\mathbf{T}$ for a model of I and the dependencies $\Sigma_{st} \cup \Sigma_t$. Similarly, any universal solution for M and I is also a universal model of I and the dependencies of $\Sigma_{st} \cup \Sigma_t$

restricted to the target schema. This means that all results specified in subsection 5.1.1 also hold for universal solutions. In particular, it means that the universal solution can be computed by the standard-chase algorithm and it is undecidable if the universal solution exists for a given mapping M and a given instance I. The following proposition extends this result by showing that it is undecidable for a given mapping if the standard chase will terminate for all source instances.

Proposition 17 *Given a mapping M, it is undecidable if the standard (oblivious/semi-oblivious/restricted/core) chase terminates for all source instances.*

Proof: We will reduce the problem of deciding if the standard (oblivious/semi-oblivious /restricted/core) chase terminates for an instance I and set of dependencies Σ to the stated problem. For this, let us consider an instance I and set of dependencies Σ both over schema $\mathbf{R}$. Construct TGD ξ from I as follows:

$$\xi : \quad \rightarrow R_1(\bar{a}_1), R_2(\bar{a}_2), \ldots, R_n(\bar{a}_n)$$

such that $head(\xi) = I$. Consider $\Sigma_t = \Sigma \cup \{\xi\}$, $\Sigma_{st} = \varnothing$ and mapping $M = (\varnothing, \mathbf{R}, \Sigma_{st}, \Sigma_t)$. We can observe that the standard chase for mapping M and any instance I' terminates if and only if the standard chase terminates on input I and Σ. From this reduction and Theorem 2 (and correspondingly Theorems 3, 7, 8, 11) it follows that it is undecidable if the standard (oblivious/semi-oblivious/restricted/core) chase terminates for all source instances for a given mapping M∎

In [27], Fagin et al. present a sufficient condition for the universal solution to not exist, as shown in the following theorem:

Theorem 30 [27] *Let $M = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ be a mapping and I be a source instance such that there is a failing branch for the standard chase with input I and $\Sigma_{st} \cup \Sigma_t$. Then there is no universal solution for I and M.*

In data exchange we may also have settings for which there exist solutions, but there is no universal solution.

Example 27 *Consider the mapping specified as follows:*

$$M = (\{S\}, \{R\}, \{S(x,y) \to R(x,y)\}, \{R(x,y) \to \exists z\, R(y,z)\})$$

and the source instance $I = \{S(a,b)\}$. Clearly, because of the second dependency, there is no universal solution for this setting, but there exists a solution $J = \{R(a,b), R(b,b)\}$.

Kolaitis et al. in [52] proved that it is undecidable to check for a given instance I and a mapping M if there exists a solution for I and M.

Before presenting how to compute the certain answers on target schema for UCQ queries in a data exchange mapping using a universal solution, let us introduce the notion of *naïve evaluation* as presented in [55]. Let I be an instance, possible with null values, and q be a UCQ query. The $q_{\text{naïve}}(I)$ is defined by evaluating q on I by treating each null as new special constant, and then by eliminating from the result all the tuples that contain such special constants.

Theorem 31 [27] *Let $M = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ be a mapping and I an instance over the source schema that does not contain nulls. Suppose also that there exists a universal solution J for I and M. Then, $certain_M(q, I) = q_{naïve}(J)$ for any $q \in$UCQ.*

Libkin [55] showed that UCQ is the largest class of queries with the property that the certain answers may be computed using the naïve evaluation. We may note here the result from [27] which states that for a data exchange mapping M specified by a weakly acyclic set of TGD's one may get the certain answers to a union of conjunctive query with at most one unequality per disjunct over an instance I and M from the universal solution in polynomial time in the size of the data. This does not contradict with Libkin's result as the certain answer to the query is not computed using the naïve evaluation. We conclude this subsection by reiterating the result which specifies that between the infinite set of universal solution there exists a universal solution which is minimal in size. Such universal solution is called *the core solution* and, as noted in [28], it is unique up to variable renaming. Even more, in [23] it was proved that if there exists a universal solution then its core can be computed using the core-chase algorithm.

5.1.3 Data Exchange beyond Universal Solutions

It is known that there is no universal solution for the mapping and the source instance from Example 27. On the other hand, when considering the query $q(x) \leftarrow \exists y\ R(x,y)$, the certain answer contains the tuples $\{(a),(b)\}$. In their seminal paper [16], Cali, Gottlob and Kifer investigate the problem of conjunctive query answering when the universal solution is not guaranteed to exist. For this, the authors unravel two classes of TGD's , namely *guarded tuple generating dependencies* (GTGD) and *weakly guarded tuple generating dependencies* (WGTGD), for which the problem of conjunctive query evaluation is decidable. Intuitively, a TGD is guarded if its body contains an atom called *guard* that covers all the variables in the body. Clearly LAV dependencies are GTGD's. A set of TGD's is said to be weakly guarded if for each TGD its body contains one atom that covers all the variables that appear in the affected position. That is predicate positions that may contain new labeled nulls generated during the chase process.

Example 28 *Let us consider the following dependencies:*

$$\begin{aligned}
\xi_1 &: S(x), R(x,y) \rightarrow \exists z\ R(y,z)\\
\xi_2 &: R(x,z), R(z,y) \rightarrow R(y,x)
\end{aligned}$$

In ξ_1, the atom $R(x, y)$ covers all the variables in the body, meaning that it is a GTGD. Clearly, ξ_2 is not GTGD as there is no atom to cover all variables from the body. The affected position for the set $\{\xi_1, \xi_2\}$ is $(R, 2)$, meaning that we may introduce new labeled nulls during the chase process only in the second position of R. Because in ξ_2 the atom $R(z, y)$ covers both variables that appear in the affected position in ξ_2, it follows that ξ_2 is a WGTGD.

Cali et al. [16] give data complexity bounds for the conjunctive query answering problem. This problem is expressed as follows: *Does a tuple t belong to the certain answer for a mapping M and a source instance I?* In their paper, the complexity bounds discovered are: (1) for a fixed set GTGD's , the conjunctive query answering problem is NP-complete; (2) for atomic queries, the problem becomes polynomial; (3) in case the fixed dependencies are WGTGD's , the conjunctive query answering problem becomes EXPTIME-complete. More recently other classes of dependencies for which one may get the certain answers to conjunctive queries are revealed by Cali et al. in [19].

A universal solution is enough to compute certain answer for any UCQ query under the certain answer semantics for UCQ queries over the target schema. Therefore, the following question rises: *Is this semantics also a good model for general queries?* As shown in [7] and [54], this semantics is not suitable for general queries, as it may give unintuitive answers even for simple copying data exchange settings.

Example 29 *Let us consider a data exchange setting $M = (\{R\}, \{R'\}, \Sigma_{st}, \varnothing)$, where Σ_{st} simply copies the source into target: $R(x, y) \to R'(x, y)$. Consider the source instance $I = \{R(a, b)\}$ and query over the target schema $q(x, y) \leftarrow R'(x, y) \wedge \neg R'(x, x)$. Since one of the solution is instance $J = \{R'(a, b), R'(a, a)\}$, it follows that the certain answer $certain_M(q, I) = \varnothing$. Now, when applying the same query on the source instance (by replacing relation name R' with R), it returns the set of tuples $\{(a, b)\}$. Clearly* *this is not the expected behavior as the target instance is supposed to be a copy of the source instance.*

In order to solve the problems related to general queries, new closed world semantics were proposed for the data exchange problem [38; 44; 45; 47; 54]. Libkin [54] replaced universal solutions with a set of CWA-solutions (that are in fact universal solutions), used afterwards to compute certain answers for FO queries. Hernich and Schweikardt [47] introduced a new chase based algorithm, called the α-chase, to compute CWA-presolutions, these are instances useful to compute the set of CWA-solutions. In Chapter 6 we introduce a new closed world semantics and we argue that it is suitable for the data exchange problem. The representation of the solution set in this case will be done using conditional tables. A similar chase process for conditional tables that consider only source-to-target dependencies was recently introduced in [10].

We close this sub-chapter by also mentioning the work of Afrati, Li and Pavlaki [5] that extended the data exchange problem by considering dependencies with arithmetic comparisons.

5.2 Data Repair

Data repair (or database repair) is one of the main problems associated with inconsistent databases. A *data repair* solution for an inconsistent database instance is a consistent database instance over the same schema that differs *minimally* from the initial instance. Based on the minimality criterion, different approaches are used: *symmetric-difference repair* presented by Arenas et al. in [8]; *subset-repairs* as developed by Chomicki and Marcinkowski in [22]; *cardinality repair* proposed by Lopatenko and Bertosi in [57]; *attribute based repairs* proposed by Wijsen [75] and the newly introduced *component-cardinality repair* by Afrati and Kolaitis in [4]. The most studied problems related to *database repair* are: the *repair* solution-existence checking and the decision problem if a given instance is a *repair* solution. In their work, Afrati and Kolaitis [4] prove the intractability for the repair check problem in general case and also prove that the subset-repair and symmetric-difference repair are tractable for weakly acyclic LAV dependencies. On the other hand, Staworko and Chomicki [71] demonstrate the tractability for the same problems in case of full dependencies. In this section we will provide an extended class of dependencies, called *semi-LAV* that properly includes both weakly acyclic LAV and full dependencies and still keeps the polynomial complexity for the subset-repair and symmetric-difference repair problems.

5.2.1 Data Repair Solutions

A tuple (I, Σ) is called *data repair setting* (or simply repair setting), if I is a ground instance over schema $\mathbf{R}$ and Σ is a set of dependencies over the same schema $\mathbf{R}$. In the remaining part of this chapter, if not specified otherwise, the set of dependencies are identified by TGD's. An instance J is said to be a *repair solution* (or simply solution) for repair setting (I, Σ) if $J \models \Sigma$ and J differs in a "minimal" way from instance I. The minimality condition gives a set of solutions types for the database repair problem.

Definition 24 *Let (I, Σ) be a repair setting over schema $\mathbf{R}$. A ground instance J over $\mathbf{R}$ such that $J \models \Sigma$ is said to be:*

- subset-solution *if $J \subseteq I$ and it is maximal among subsets of I satisfying Σ.*
- superset-solution *if $J \supseteq I$ and it is minimal among supersets of I satisfying Σ.*
- $\oplus$-solution *if J is $\leq_I$-minimal among instances satisfying Σ.*

Example 30 *Consider the set of dependencies $\Sigma = \{\xi : R(x,y) \to \exists z\, R(x,z)\}$ and the instance $I = \{R(a,b), R(a,c), R(a,d), R(d,a)\}$. Also consider the following instances:*

J_1	J_2	J_3	J_4	J_5
$R(a,d)$	$R(a,b)$	$R(a,b)$	$R(a,b)$	$R(a,d)$
$R(d,a)$	$R(a,c)$	$R(a,c)$	$R(a,d)$	$R(d,a)$
	$R(a,d)$	$R(a,d)$	$R(d,a)$	$R(b,b)$
	$R(d,a)$	$R(d,a)$	$R(b,a)$	
	$R(b,a)$	$R(b,e)$		
	$R(c,a)$	$R(c,e)$		
		$R(e,e)$		

For the data repair setting (I,Σ), instance J_1 is the only subset-solution, J_2 and J_3 are two superset-solutions, J_4 is a $\oplus$-solution and instance J_5 is neither type of solution even if $J_5 \vDash \Sigma$ because $J_5 \smallsetminus \{R(b,b)\}$ is a subset of I that satisfies the dependency.

Note that in the previous example both instances J_2 and J_3 are superset-repairs even if $|J_2| < |J_3|$. There are also other types of repairs that consider cardinality as a measure for minimality. In [57] Lopatenko and Bertossi introduce the *cardinality* repairs and later Afrati and Kolaitis [4] define the *component-cardinality* repairs. In this dissertation we will not consider these types of repairs as these problems become intractable (coNP-complete) even for the simple class of full TGD's. In general, for each type of repairs there may be several solutions or it may also be that there does not exist a solution at all.

Definition 25 *Decision problems considered:*

- Existence-of-Repair(Σ, *subset/superset/$\oplus$-repair*)

  The input is an instance I and the question is whether I has a subset/superset/$\oplus$-repair for (I,Σ)

- Repair-Checking(Σ, *subset/superset/$\oplus$-repair*)

  The input is instances I and K and the question is whether the instance K is a subset/superset/$\oplus$ repair for (I,Σ)

5.2.2 Basic Characterizations

Our first contribution is a general characterization of repairs, applicable to any set of TGD's. For this we need the following lemma and definition.

Lemma 2 [30] *Let I and J be instances and Σ a set of TGD's over a schema* $\mathbf{R}$. *If $I \subseteq J$, then there exists a homomorphism h such that $h(chase_{\Sigma}^{\mathbf{std}}(I)) \subseteq chase_{\Sigma}^{\mathbf{std}}(J)$.*

Before we introduce the next definition let us consider the following example.

Example 31 *Let Σ contain the set of TGD's:*

$$\begin{aligned}
\xi_1 : &\quad Emp(e) \rightarrow \exists m\, EmpMgr(e,m)\\
\xi_2 : &\quad EmpMgr(e,m) \rightarrow Manager(m)\\
\xi_3 : &\quad EmpMgr(e,e) \rightarrow SelfMgr(e)
\end{aligned}$$

Also consider instance $I = \{Emp(john), Emp(ray)\}$. In this case the following instance represents $chase_{\Sigma}^{\mathbf{std}}(I)$:

$chase_{\Sigma}^{\mathbf{std}}(I)$
$Emp(john)$
$Emp(ray)$
$Manager(X)$
$Manager(Y)$
$EmpMgr(john,Y)$
$EmpMgr(ray,Y)$

Let us now consider ground instance J specified as follows: $Emp^J = \{(john),\ (ray)\}$; $MgrEmp^J = \{(john.john),\ (ray,john)\}$; $Manager^J = \{(john)\}$ and $SelfMgr^{\Sigma} = \emptyset$. It is easy to observe that $h = \{X/john, Y/john\}$ is a homomorphism from $chase_{\Sigma}^{\mathbf{std}}(I)$ into J actually $h(chase_{\Sigma}^{\mathbf{std}}(I)) = J$, but on the other hand neither J or $h(chase_{\Sigma}^{\mathbf{std}}(I))$ satisfy Σ.

The previous example leads to the introduction of a new type of homomorphism called Σ-satisfying homomorphism:

Definition 26 *Given an instance K and a set Σ of TGD's over a schema $\mathbf{R}$, as well as a ground instance I over $\mathbf{R}$, a* Σ-satisfying *homomorphism from K into I is a homomorphism h from K into I, such that $h(K) \subseteq J \subseteq I$, and $J \models \Sigma$, for some $J \subseteq I$.*

We can now state some necessary and sufficient conditions for which an instance K, such that $K \models \Sigma$, is a $\oplus$-repair for a given repair setting (I, Σ). The first condition of this theorem assures that $K \smallsetminus I$ does not contain superfluous tuples. The second condition guarantees that no more tuples from I could be added to K and still satisfy the dependencies.

Theorem 32 *Let Σ be a set of TGD's, I and K two ground instances such that $K \models \Sigma$. Then K is a $\oplus$-repair for (I, Σ) if and only if the following conditions are satisfied:*

1. *For all Σ-satisfying homomorphisms h such that $h(chase_{\Sigma}^{\mathbf{std}}(I \cap K)) \subseteq K$, and all instances $J \subseteq K$, such that $J \models \Sigma$ and $h(chase_{\Sigma}^{\mathbf{std}}(I \cap K)) \subseteq J$, it holds that $J = K$.*

2. *There do not exist a tuple $t \in I \smallsetminus K$ and Σ-satisfying homomorphism h with an instance $J \subseteq K \cup I$, $J \models \Sigma$, such that $h(chase_{\Sigma}^{\mathbf{std}}((I \cap K) \cup \{t\})) \subseteq J \subseteq K \cup I$.*

Proof: Suppose K is a $\oplus$-repair for (I, Σ). As $K \models \Sigma$, we have $chase_{\Sigma}^{\mathbf{std}}(K) = K$. From Lemma 2 it is that there exists h with $h(chase_{\Sigma}^{\mathbf{std}}(I \cap K)) \subseteq K$. Toward a contradiction, suppose that there exists a ground instance J, such that $h(chase_{\Sigma}^{\mathbf{std}}(I \cap K)) \subseteq J \subset K$ and $J \models \Sigma$. Then $J \smallsetminus I \subset K \smallsetminus I$ and, since $I \cap K \subseteq chase_{\Sigma}^{\mathbf{std}}(I \cap K)$ and $I \cap K$ is ground, we have that $I \cap K \subseteq J$, meaning that $I \smallsetminus J \subseteq I \smallsetminus K$, and consequently that $J \leq_I K$. But this contradicts the assumption that K is a $\oplus$-repair. Therefore it must be that $J = K$, meaning that K satisfies condition 1.

Suppose then that condition 2 is violated, and tuple t, Σ-satisfying homomorphism h and instance J exist. Since, for such a J, we would have $I \setminus J \subset I \setminus K$, it would follow that $J \leq_I K$, contradicting the assumption that K is a $\oplus$-repair.

For the only if direction, let K be an instance such that $K \vDash \Sigma$ and K satisfies conditions 1 and 2. We need to show that K is $\leq_I$-minimal. If this is not the case, there must be an instance J, such that $J \vDash \Sigma$ and $J <_I K$. Thus $J \setminus I \subseteq K \setminus I$, and $I \setminus J \subseteq I \setminus K$, and at least one of the inclusions is proper. Suppose first that it were the case that $I \setminus J = I \setminus K$. Then we would have $J \subseteq K$. Thus, if $J \setminus I$ were to be a proper subset of $K \setminus I$, it would necessarily be that $J \subset K$. Since then $I \cap K = I \cap J$, we have $chase^{\mathbf{std}}_{\Sigma}(I \cap K) = chase^{\mathbf{std}}_{\Sigma}(I \cap J)$. By Lemma 2 we have a homomorphism h, such that $h(chase^{\mathbf{std}}_{\Sigma}(I \cap J)) \subseteq chase^{\mathbf{std}}_{\Sigma}(J) = J$. Then it would hold that $h(chase^{\mathbf{std}}_{\Sigma}(I \cap K)) \subseteq J \subset K$, contradicting condition 1.

If it were the case that $I \setminus J \subset I \setminus K$, we would find a tuple $t \in (J \cap I) \setminus (K \cap I)$, that is in $I \setminus K$, such that by Lemma 2 we would have a homomorphism h, such that $h(chase^{\mathbf{std}}_{\Sigma}((I \cap K) \cup \{t\})) \subseteq J \subseteq K \cup I$, a contradiction to the assumption that K satisfies condition 2∎

5.2.3 Complexity Results

In order to introduce the data complexity results for the problems formalized in Definition 25, we need first to define some preliminary notions.

Definition 27 *The* Gaifman graph G^I *for an instance* I *is an undirected graph with vertex set* $dom(I)$ *and an edge between two vertices* x *and* y *if* x *and* y *appear together in a tuple of* I.

Definition 28 *The* Gaifman graph of nulls $G^I_{\Delta_\mathsf{N}}$ *is the Gaifman graph* G^I *restricted to the nulls occurring in* I.

Definition 29 *A* block *is a connected set of nulls in $G^I_{\Delta_\mathsf{N}}$. Let $N \subseteq \Delta_\mathsf{N}$. We denote by $blocks(N, I)$ the set of all blocks from the Gaifman graph of nulls $G^I_{\Delta_\mathsf{N}}$ restricted to the nulls in N. If b is a block we denote by $blocksize(b)$ the cardinality of b.*

The blocksize definition is extended to an instance I. Thus $blocksize(I)$ is the size of the largest block of nulls in I. For an instance I, we said that it had a *bounded block size* if there existed a constant c such that $blocksize(I) \leq c$. In this case we say that the blocksize is bounded by c. Clearly any finite instance has a bounded blocksize.

The next theorem is due to Fagin, Kolaitis and Popa [28] and was further highlighted by Gottlob and Nash [34]. Note that any set of source-to-target tgds is non-recursive, hence weakly acyclic.

Theorem 33 [28]

1. *Let I be a ground instance, and Σ a weakly acyclic set of TGD's. Then the size of $chase^{\mathbf{std}}_{\Sigma}(I)$ is polynomial in I.*
2. *Let I_1 be a ground instance over $\mathbf{R}_1$ and I_2 be a ground instance over $\mathbf{R}_2$. Let Σ be a set of source-to-target TGD's. Then $chase^{\mathbf{std}}_{\Sigma_{12}}(I_1 \cup I_2)$ has bounded blocksize.*
3. *Let I and K be instances such that $blocksize(K) \leq c$, for some constant c. Then it can be tested whether there exists a homomorphism from K to I in time $O(|K|^c)$.*

We will also use the following result obtained by Afrati and Kolaitis.

Theorem 34 [4] *Let Σ be a set of weakly acyclic TGD's and I an instance, such that $I \models \Sigma$. Then there exists a constant c, depending only on Σ, such that for each tuple $t \in I$ there is an instance $K_t \subseteq I$, such that $t \in K_t, K_t \models \Sigma$, and $|K_t| \leq c$.*

Armed with these results we will derive a polynomial time algorithm that determines the existence of database repair solution for weakly acyclic set of TGD's.

Theorem 35 *Let Σ be a weakly acyclic set of TGD's. Then the problem*

$$\textsc{Existence-of-Repair}(\Sigma, \mathit{subset})$$

can be solved in polynomial time.

Proof: Let (I, Σ) be a repair setting and c the constant mentioned in Theorem 34 Consider the following algorithm:

```
REPAIR-SEARCH($I, \Sigma, c$)
1  for $i \leftarrow 1$ to $c$
2        do for $K \subseteq I, |K| = i$
3                do if $K \models \Sigma$ return TRUE
4  return FALSE
```

The algorithm is obviously sound. The completeness follows directly from Theorem 34. The inner loop on line 2 is executed at most $\binom{|I|}{i}$ times and the outer loop on line 1 at most c times. Thus, line 3 is executed at most $|I|^c$ times. Each execution of line 3 is done in space logarithmic in at most c. Consequently, the algorithm runs in time polynomial in the size of I ∎

We note that Existence-of-Repair(Σ, superset) is a non-problem, since superset-solutions always exist (recall that no egd's are present). We now turn our attention to the solution checking problem.

Afrati and Kolaitis derived the following complexity theoretic characterization for repair checking with weakly acyclic TGD's.

Theorem 36 [4] *There is a weakly acyclic set Σ of TGD's such that the following problems:* Repair-Checking(Σ, subset), Repair-Checking($\Sigma, \oplus$) *are both coNP-complete.*

To this we can now add the missing piece.

Theorem 37 *There is a weakly acyclic set Σ of TGD's such that the problem of* Repair-Checking$(\Sigma, \text{superset})$ *is coNP-complete.*

Proof: We will show that the complementary problem of deciding whether an instance K is not a superset-repair of I w.r.t. Σ is NP-complete.

For the upper bound, we check in logarithmic space if $K \vDash \Sigma$, if not, K is not a superset-repair. Otherwise non-deterministically choose an instance J such that $K \supset J \supseteq I$. It follows that K is not a superset-repair if and only if $J \vDash \Sigma$. Again the satisfaction can be tested in logarithmic space.

For the lower bound we will again reduce the Positive 1-In-3-SAT problem to the superset-repair checking problem.

Recall the Positive 1-In-3-SAT is the following decision problem: given a Boolean formula φ in a conjunctive normal form with each conjunct is a disjunction of exactly 3 literals, is there a truth assignment for φ that makes true exactly one variable in each conjunct? This problem was showed by Schaefer [69] to be NP-complete.

Returning to our reduction, consider the schema having five relations P, T, S, D, E of arities 3,2,3,2 and 1 respectively. The dependencies Σ will be :

$$\begin{array}{rcl} P(x,y,z) & \rightarrow & \exists u,v,w\, T(x,u), T(y,v), T(z,w), S(u,v,w) \\ T(x,u), T(x,u'), D(u,u') & \rightarrow & \exists y\, E(y) \\ T(x,u), E(y) & \rightarrow & \exists u'\, T(x,u'), D(u,u') \end{array}$$

Given an instance $\mathcal{P}$ of the Positive 1-In-3-SAT problem we construct I as

$$\begin{array}{rcl} P^I & = & \{(x,y,z) \,:\, x \vee y \vee z \text{ is a clause in } \mathcal{P}\} \\ T^I & = & \varnothing \\ S^I & = & \{(0,0,1),\ (0,1,0),\ (1,0,0)\} \\ E^I & = & \varnothing \\ D^I & = & \{(0,1),\ (1,0)\} \end{array}$$

and instance K as containing I plus the following tuples:

$$\begin{aligned} T^K &= \{(x,1),\ (x,0)\ :\ x \text{ is a variable in } \mathcal{P}\} \\ E^K &= \{(0)\} \end{aligned}$$

This reduction clearly is being polynomial and, since obviously $K \vDash \Sigma$, it remains only to prove that there exists a truth assignment for $\mathcal{P}$ making exactly one variable in each clause true if and only if K is not a superset-repair. For the *if* part suppose that there exists such a truth assignment. Let J be an instance that contains I plus:

$$T^J = \{(x,u)\ :\ u \text{ is the assignment for variable x in the solution of } \mathcal{P}\}$$

It is easy to see that $J \vDash \Sigma$ and that $K \supset J \supseteq I$. In other words, K is not a superset-repair.

For the *only if* part, suppose that K is not a superset-repair. This means that there exists an instance J such that $K \supset J \supseteq I$, and $J \vDash \Sigma$. As the inclusion between J and K is proper, it means that there exists at least one tuple in $K \smallsetminus J$. From the construction of K it follows that the tuple is either in T^K or in E^K. If the tuple is in T^K, since $J \vDash \Sigma$, the last dependency tells us that the tuple (0) cannot be in E^J. Then the second dependency implies that T^J contains a proper truth assignment, and the first dependency implies that this truth assignment actually is Positive 1-In-3 and satisfies $\mathcal{P}$■

In order to overcome the intractability barriers for the previous problems, it is clear that the set Σ of dependencies has to be restricted. We next introduce a large class of TGD's for which all of the considered problems can be solved in polynomial time. The proofs rely on recently developed homomorphism techniques (i.e. [4; 27; 33]), and on the Σ-satisfying homomorphisms introduced in Definition 26.

First, we introduce the new class of TGD's. For this, we need the concept of the rank of a node in the dependency graph (see Definition 6) of a set Σ of TGD's.

Definition 30 ([27]) *Let (R, i) be a vertex in the dependency graph of a set Σ of TGD's. Then rank(R, i) is the maximum number of existential edges along any path in the graph ending in (R, i).*

Lemma 3 ([27]) *If Σ is a weakly acyclic set of TGD's, then for each vertex (R, i) of the dependency graph of Σ, rank(R, i) is finite and is bounded by some constant c depending only on Σ.*

Definition 31 *Let Σ be a set of TGD's over a schema $\mathbf{R}$. For all relational symbols $R \in \mathbf{R}$ that occur in Σ, we say that a position (R, i) is* unsafe *if rank$(R, i) > 0$. Any relational symbol R that contains an unsafe position is said to be unsafe. A set Σ of weakly acyclic dependencies is said to be* semi-LAV *if all unsafe relational symbols occur in the body of LAV-dependencies only.*

Example 32 *Consider the following simple set of dependencies that is neither full or LAV:*

$$\begin{array}{ll} \xi_1 : & R(x,y) \rightarrow \exists z\, R(x,z), S(x) \\ \xi_2 : & S(x), S(y) \rightarrow \exists z\, R(x,z), R(z,y) \end{array}$$

In this set we have rank$(R, 1) = 1$, rank$(R, 2) = 2$ and rank$(S, 1) = 0$. Thus, relation R is unsafe. On the other hand, relation R is safe so it may appear in the body of the non-LAV dependency ξ_2.

The class of semi-LAV TGD's possesses several useful properties, as it will be shown next.

Lemma 4 *Let Σ be a set of semi-LAV TGD's, let I be a ground instance and K be the result of an arbitrary standard-chase sequence using Σ starting from I. Then for any null X that occurs in K, the number of tuples in K containing X is bounded by a constant that depends only on Σ.*

Proof: Let J be the first instance in the chase sequence from I to K where the null X appears, and denote with J_X the subset of tuples of J where X appears. If r is the maximum number of atoms in the consequent of any dependency in Σ, it is clear that $|J_X| \leq r$. Furthermore, as null value X can appear only in tuples of unsafe relations (called *unsafe tuples*), it means that the tuples in J_X can only be matched with the body of some LAV-dependencies, each such body consisting of a single atom. Consequently, the set of tuples K_X in K that contain X can be generated by chasing J_X using only the LAV-dependencies in Σ. As Σ is weakly acyclic, it follows from Theorem 33 that there is a polynomial p such that $|K_X \leq p(|J_X|)$. Thus $|K_X| \leq p(r)$, which is a constant depending only on Σ■

Lemma 5 *Let Σ be a set of weakly acyclic TGD's, I a ground instance, and K the result of some arbitrary standard -chase sequence using Σ starting from I. Then there exists a partitioning $\{V_0, V_1, \dots, V_m\}$ of the nulls in K, such that null $X \in V_i$ iff X was generated during the chase process using only nulls from $\{V_0, \dots, V_{i-1}\}$.*

Proof: Let $N_0, N_1, \dots, N_m$ be a partition of the nodes in the dependency graph of Σ based on their rank. Since Σ is weakly acyclic, such a partitioning exists and, for any dependency in Σ, the rank of the positions in the consequent is larger than or equal to the rank of any position in the body of the dependency [27]. We show the existence of the null partitioning constructively based on the chase steps. Initially let V_i be the empty set for all $i \in \{0, 1, \dots, m\}$. During the chase process, when a new null X is created in a position (R, i), we add this null to set V_1, if there were no nulls in the tuples that triggered the dependency. We add the null X to the set V_i if the set of nulls belonging to the tuples that triggered the dependency that generated X is a subset of $\{V_0 \cup V_1 \cup \dots \cup V_{i-1}\}$ and if there exists a null Y in that set such that $Y \in V_{i-1}$. Because the rank of each position is at most m, it means that null X cannot be generated by a null belonging to V_m or higher. This proves the existence of partitioning $\{V_0, V_1, \dots, V_m\}$ of the nulls. Note that $V_0 = \varnothing$■

Theorem 38 *Let Σ be a set of semi-LAV TGD's, let I be a ground instance, and K be the result of some arbitrary standard-chase sequence using Σ starting from I. Then there is a constant c, depending only on Σ, such that blocksize$(G^K) \leq c$.*

Proof: Recall that we denote by G^K the Gaifman graph of nulls for instance K (see Definition 28). Let $\{V_0, V_1, \dots, V_m\}$ be the partitioning of the variables in K, as described in Lemma 5. We prove by an induction on i that there are constants c_i, such that, for any block $b \in blocks(V_0 \cup \cdots \cup V_i, G^K)$, we have $blocksize(b) \leq c_i$, meaning that $blocksize(G^K) = max\{blocksize(b) : b \in blocks(V_0 \cup \cdots \cup V_m, G^K)\} \leq c_m$.

Basis: Clearly $V_0 = \varnothing$, and $blocks(V_0, G^K) = \varnothing$. We can therefore set $c_0 = 0$.

Inductive step: Let b be a block in $blocks(V_0 \cup \cdots \cup V_i, G^K)$. If all the nulls in b are in $V_0 \cup \cdots \cup V_{i-1}$, the block b is also in $blocks(V_0 \cup \cdots \cup V_{i-1}, G^K)$ and, by the inductive hypothesis, we have $blocksize(b) \leq c_{i-1}$. Suppose then that X is a null in b from V_i. This means that X was generated from variables in $V_0 \cup \cdots \cup V_{i-1}$ by an existential dependency ξ. There are three cases to consider:

Case 1: ξ is a dependency with possibly several atoms in the body. Then all these atoms are over safe relations, thus not mapped to nulls during the chase process. Thus X, along with the other existential nulls in the consequent of ξ will form their own block in $blocks(V_0 \cup \cdots \cup V_i, G^K)$, with blocksize at most s, where s is the maximum number of special edges incident from any node in the dependency graph of Σ.

Case 2: ξ has a single unsafe atom in the body, but none of the (universally quantified) variables in the body occurs in the head. Then ξ does not propagate any nulls from the body to the head and, as in case 1, the null X will belong to a block in $blocks(V_0 \cup \cdots \cup V_i, G^K)$, with blocksize at most s.

Case 3: ξ has a single unsafe atom in the antecedent. Then ξ was triggered by a single tuple containing only variables in $V_0 \cup \dots \cup V_{i-1}$. Thus each of these variables belong to a block $b' \in blocks(V_0 \cup \dots \cup V_{i-1}, G^K)$ and, by the inductive hypothesis, $blocksize(b') \leq c_{i-1}$. Therefore b' consists of at most c_{i-1} nulls, say $N = \{Y_1, \dots, Y_{c_{i-1}}\}$. Let K_N symbolize the subset of tuples of K that contains nulls in N and let K_{N_ℓ} denote the tuples that contain null y_ℓ. By Lemma 4, there are constants $d_1, \dots, d_{c_{i-1}}$ such that $|K_{y_\ell}| \leq d_\ell$, for $\ell \in \{1, \dots, c_{i-1}\}$. Consequently $|K_N| \leq \sum_{\ell=1}^{c_{i-1}} d_\ell$. Each tuple in K_N can generate at most $D \cdot s$ new null X in V_i, where D is the number of dependencies in Σ. Each such generated null X increases the blocksize of b' by at most one. Consequently $blocksize(b) \leq blocksize(b') + |K_N| \cdot D \cdot s \leq c_{i-1} + \sum_{\ell=1}^{c_{i-1}} d_\ell \cdot D \cdot s$ ■

The following theorem shows that for the semi-LAV class of dependency sets the existence of the Σ-satisfying homomorphism can be done in polynomial time. This is a crucial result proving the tractability results for the repair problems.

Theorem 39 *Let Σ be a set of semi-LAV TGD's, and I and K be two ground instances such that $K \subseteq I$. Then the problem of deciding if there exists a Σ-satisfying homomorphism from $chase_\Sigma^{\mathbf{std}}(K)$ into I is polynomial.*

Proof: We know that the result of chasing a ground instance with a set of semi-LAV TGD's has the size of each block bounded by a constant that depends only on Σ. Also, we know from the definition of semi-LAV TGD's that unsafe relational symbols (that is relational symbols that may contain nulls generated during the chase process) in Σ appear only in the body of LAV TGD's.

The following algorithm takes as input a set Σ of semi-LAV TGD's, a ground instance I and the constant c from Theorem 34. The algorithm returns **true** if there is a Σ-satisfying homomorphism from $chase_\Sigma^{\mathbf{std}}(K)$ into I and it returns **false** otherwise. For a block b of nulls of J, we denote by J_b the set of all tuples where nulls of b occur. Also, for an instance I, we denote by I_u the subset of unsafe tuples in I.

```
HOMOM-SEARCH($K, I, \Sigma, c$)
 1  $J \leftarrow chase_{\Sigma}^{\mathbf{std}}(K)$, $result \leftarrow$ TRUE
 2  for each block $b \in blocks(G^J)$ do
 3      $result_b \leftarrow$ false
 4      for each homom. $h_b$ from $b$ to $I$ do
 5          for each tuple $t \in h_b(J_b)$ do
 6              for $i \leftarrow 1$ to $c$ do
 7                  for $A_{b,t} : \{t\} \subseteq A_{b,t} \subseteq I_u$, $|A_{b,t}| = i$ do
 8                      if $(h_b(b) \cup A_{b,t}) \vDash \Sigma$
 9                          then $result_b \leftarrow$ true
10                      $result \leftarrow result$ & $result_b$
11  return $result$
```

Suppose that the algorithm returns **true**. Let b be a block in G^J. Based on an enumeration of $dom(K \cup I)$, let h_b be the first homomorphism the algorithm discovers and, for each $t \in J_b$, let $A_{b,t}$, the smallest corresponding set, such that $(h_b(J_b) \cup A_{b,t}) \vDash \Sigma$. Let

$$J^{\Sigma} = \bigcup_{b \in blocks(G^J)} \left(h_b(J_b) \cup \{A_{b,t} : t \in h_b(J_b)\}\right).$$

We claim that $h = \bigcup\{h_b : b \in blocks(G^J)\}$ is a Σ-satisfying homomorphism form J to I. Clearly $h(J) \subseteq J^{\Sigma} \subseteq I$. We thus have to show that $J^{\Sigma} \vDash \Sigma$, that is that $J^{\Sigma} \vDash \xi$ for each $\xi \in \Sigma$. There are two cases to consider:

Case 1: ξ is a LAV-tgd whose body consists of a single atom over an unsafe predicate. Let $t \in J^{\Sigma}$ be an unsafe tuple triggering ξ. Then there is a block b in G^J, such that $t \in h_b(J_b)$, or $t \in A_{b,t}$. Then $(h_b(b) \cup A_{b,t}) \vDash \xi$. Since LAV TGD's are closed under union [73] and all sets $A_{b,t}$ contain only unsafe tuples, it follows that $J^{\Sigma} \vDash \xi$.

Case 2: The body of ξ consists of possibly several atoms and all these atoms are over safe predicates. Since any possible set of triggering tuples is already present in J and J was obtained by chasing K with Σ, it follows that $J^{\Sigma} \vDash \xi$.

To show that the algorithm is complete, let $J = chase^{\mathbf{std}}_{\Sigma}(K)$ and suppose that there exists a homomorphism h and an instance $L \models \Sigma$, such that $h(J) \subseteq L \subseteq I$. Let h be the smallest such homomorphism, based on the enumeration of $dom(K \cup I)$. We need to show that the algorithm returns **true**. The homomorphism h can obviously be expressed as a union of homomorphisms h_b, each from a block of $b \in blocks(G^J)$. Thus, the algorithm will certainly discover h. Since $h(J) \subseteq L$, and $L \models \Sigma$, Theorem 34 guarantees that, for each b and tuple $t \in I_u$, the corresponding set $A_{b,t} \subseteq L$ will be discovered by the algorithm. Consequently, the algorithm will return **true**.

It remains to analyze the time complexity of the algorithm. We know from Theorem 33 that the size of J is polynomial in the size of K and that J, therefore, has a polynomial number of blocks. The same theorem also tells us that, for each block of $b \in blocks(G^J)$, there are at most $|I|^c$ homomorphisms from b into I. Similarly to the algorithm in Theorem 35, line 8 is repeated at most $O(|I|^c)$ times. Finally, the test if $(h_b(b) \cup A_{b,t}) \models \Sigma$ can be done in space logarithmic in $|I|$. In conclusion, the algorithm runs in time polynomial in $|K| + |I|$∎

The preceding theorems allow us to derive polynomial time algorithms for the repair problems, for the case when Σ is a set of semi-LAV TGD's.

Theorem 40 *Let Σ be a set of semi-LAV dependencies. Then the following problems* REPAIR-CHECKING(Σ, *superset/subset/*$\oplus$) *are polynomial.*

Proof: Let us first consider the $\oplus$-case. We want to check if an instance K is a $\oplus$-repair of I w.r.t. Σ. First test if $K \models \Sigma$. If so, K is indeed a $\oplus$-repair if and only if both conditions in Theorem 32 are fulfilled. Since Σ is semi-LAV, the required homomorphism tests can be done in polynomial time, as per Theorem 39. For the subset and superset cases, it suffices to test condition 2 and condition 1 respectively, of Theorem 32∎

We conclude this complexity result section with a review of the complexity results known for data repair problems. For the repair existence problem, we showed that for a weakly acyclic set of TGD's deciding of the existence of the subset-repair can be done in polynomial time (Theorem 35). Clearly, the existence of the superset and $\oplus$ repairs are non-problems for weakly acyclic set of TGD's. The table bellow shows the complexity results for the repair checking problem based on the weakly acyclic and semi-LAV classes of TGD's.

Repair Check	Σ weakly acyc.	Σ semi-LAV
subset	coNPC [4]	P *Thrm. 40*
superset	coNPC *Thrm. 37*	P *Thrm. 40*
$\oplus$	coNPC [4]	P *Thrm. 40*

Figure 5.1: The complexity of data repair

5.3 Data Correspondence

Data correspondence is an old practical problem that has not yet been formalized. For a better intuition behind the correspondence problem, consider a company that keeps periodical backups of its database. Through schema evolution the database structure may change in time. Now, one may want to verify the consistency of the current database instance with a backup made when the database had a different schema. Figure 5.2 presents such a scenario when the current database instance at time t_n becomes inconsistent and the "repair" of this instance is done in correspondence with the backup instance I_{bak} from time t_0 when the schema was not normalized. Note that the arrows are in both directions. The direction from instance I_{bak} to the current instance I_n ensures the completeness criterion. The direction from I_n to I_{bak} represents the soundness criterion.

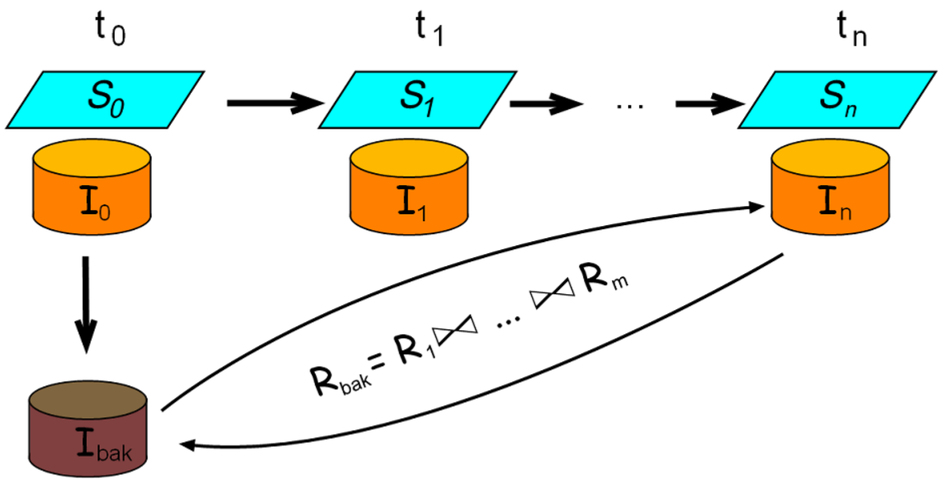

Figure 5.2: Data Correspondence for schema evolution

5.3.1 The Data Correspondence Problem

In this subsection we will formally introduce the data correspondence setting, the data corresponding problems and we will also introduce some results needed in order to provide complexity bounds for these problems.

Let $\mathbf{R}_1$ and $\mathbf{R}_2$ be two schemata with no relation symbols in common. A (*data*) *correspondence* TGD for $(\mathbf{R}_1, \mathbf{R}_2)$ is either a TGD of the form

$$\phi_1(\bar{x}) \to \exists \bar{y}\ \phi_2(\bar{x}, \bar{y}),$$

or a TGD of the form

$$\phi_2(\bar{x}) \to \exists \bar{y}\ \phi_1(\bar{x}, \bar{y}),$$

where all the atoms of ϕ_1 are over $\mathbf{R}_1$ and the atoms of ϕ_2 are over $\mathbf{R}_2$. We consider finite sets $\Sigma = \Sigma_{12} \cup \Sigma_{21}$ of such correspondence TGD's, where Σ_{12} contains all the

TGD's of the first form, and Σ_{21} the TGD's of the second form. Note that Σ_{12} are used to specify a data exchange mapping [26], where the source schema is $\mathbf{R}_1$ and the target schema is $\mathbf{R}_2$.

A *(data) correspondence mapping* is as a triple of the form $(\mathbf{R}_1, \mathbf{R}_2, \Sigma)$, where $\Sigma = \Sigma_{12} \cup \Sigma_{21}$ is a set of correspondence mappings for $(\mathbf{R}_1, \mathbf{R}_2)$ as above. We say that a ground instance (I_1, I_2) over $(\mathbf{R}_1, \mathbf{R}_2)$ *satisfies* the setting if $(I_1, I_2) \models \Sigma$.

Let $(\mathbf{R}_1, \mathbf{R}_2, \Sigma)$ be a data correspondence setting, and (I_1, I_2) a ground instance over $(\mathbf{R}_1, \mathbf{R}_2)$. Then $\langle (I_1, I_2), (\mathbf{R}_1, \mathbf{R}_2, \Sigma) \rangle$ denotes an *instance* of the *uniform-data correspondence problem*. For simplicity, we call it uniform correspondence problem. For such an instance, usually simply denoted (I_1, I_2, Σ), there are three types of solutions, namely: subset-solutions, superset-solutions, and $\oplus$-solutions.

Definition 32 *Let (I_1, I_2, Σ) be an instance of the uniform data correspondence problem and (K_1, K_2) a ground instance over $(\mathbf{R}_1, \mathbf{R}_2)$. If (K_1, K_2) satisfies Σ and $K_1 \cup K_2 \neq \emptyset$, we say that (K_1, K_2) is a*

- *subset-solution if $(K_1, K_2) \subseteq (I_1, I_2)$ and (K_1, K_2) is maximal among the subsets of (I_1, I_2) satisfying Σ;*
- *superset-solution if $(K_1, K_2) \supseteq (I_1, I_2)$ and (K_1, K_2) is minimal among the supersets of (I_1, I_2) satisfying Σ;*
- *$\oplus$-solution if (K_1, K_2) is $\leq_{(I_1, I_2)}$-minimal among the instances satisfying Σ,*

where the notion of subset is extended to pair of sets as follows: $(A, B) \subseteq (C, D)$ iff $A \subseteq C$ and $B \subseteq D$. Note, in the previous definition, that in all cases an instance can have one or several solutions. Subset and $\oplus$-solutions might not exist, whereas superset-solutions always do.

The *non-uniform-data correspondence problem* (shortly called the non-unform correspondence problem) is similar to the uniform one, except that the instance I_1 is kept fixed when looking for solutions. This has, among other things, the consequence that not even superset-solutions are guaranteed to exist.

Definition 33 *Let $(I_1, I_2, \Sigma_{12} \cup \Sigma_{21})$ be an instance of the non-uniform-data correspondence problem and K_2 a ground instance over $\mathbf{R}_2$. If (I_1, K_2) satisfies Σ, we say that K_2 is a*

- *subset-solution if $K_2 \subseteq I_2$ and K_2 is maximal among the subsets of I_2, such that (I_1, K_2) satisfies Σ;*
- *superset-solution if $K_2 \supseteq I_2$ and K_2 is minimal among the supersets of I_2, such that (I_1, K_2) satisfies Σ;*
- *$\oplus$-solution if K_2 is $\leq_{I_2}$-minimal among the instances of $\mathbf{R}_2$, such that (I_1, K_2) satisfies Σ.*

Example 33 *Consider schemata $\mathbf{R}_1 = \{R\}$ and $\mathbf{R}_1 = \{S\}$. Consider also instance (I_1, I_2) over schema $(\mathbf{R}_1, \mathbf{R}_2)$, where $I_1 = \{R(a,b), R(c,d)\}$ and $I_2 = \{S(c,a)\}$. Finally, let $\Sigma_{12} = \{R(x,y) \to \exists z\ S(x,z)\}$ and $\Sigma_{21} = \{S(x,y), S(y,z) \to R(x,z)\}$. Now consider the following pairs of instances (we used square boxes as pair delimiter):*

K_1^1	K_2^1
$R(c,d)$	$S(c,a)$

K_1^2	K_2^2
$R(a,b)$	$S(c,a)$
$R(c,d)$	$R(a,b)$
$R(c,b)$	

K_1^3	K_2^3
$R(a,b)$	$S(a,d)$

For the uniform correspondence problem $(I_1, I_2, \Sigma_{12} \cup \Sigma_{21})$, instance (K_1^1, K_2^1) is a subset-solution; similarly, instance (K_1^2, K_2^2) is a superset-solution; and (K_1^2, K_2^2) is a $\oplus$-solution. In the case of uniform correspondence problem $(I_1, I_2, \Sigma_{12} \cup \Sigma_{21})$, instance $J_2 = \{S(c,a), S(c,a)\}$ is a superset-solution because $(I_1, J_2) \vDash \Sigma_{12} \cup \Sigma_{21}$ and because $J_2 = I_2 \cup \{S(a,d)\}$. We may also note that there does not exist a subset-solution for the *non-uniform correspondence problem, following that all $\oplus$-solutions are the superset-solutions.*

Similarly to the data repair case, the above definitions give rise to classes of decision problems that we will study in the next subsection. The first class is the existence of solutions to the correspondence problem. While the second class is to check whether a given instance is a solution. We note that Fuxman, Kolaitis, Miller and Tan [11] have investigated, under the name of *Peer Data Exchange*, the existence of superset-solutions to the non-uniform correspondence problem. However, they do not require a solution to be minimal, which in fact leaves out an additional computational hurdle similar to computing the core of a solution. In addition, it turns out that the uniform correspondence problem can be seen as a special case of the problem of the *repair checking* problem.

Lemma 6 *Let $(I_1, I_2, \Sigma_1, \Sigma_2)$ be an instance of the uniform-data correspondence problem and (K_1, K_2) a ground instance. Then (K_1, K_2) is a subset(superset, $\oplus$)-solution to the uniform correspondence problem if and only if $K_1 \cup K_2$ is a subset (superset, $\oplus$, respectively) repair of $I_1 \cup I_2$ w.r.t. $\Sigma_1 \cup \Sigma_2$.*

From this lemma it follows that the characterization used for the $\oplus$-solution in Theorem 32 works for $\oplus$-uniform-data correspondence solutions as well. In case of solutions to the non-uniform correspondence problem, we have a similar characterization.

Theorem 41 *Let $(I_1, I_2, \Sigma_{12}, \Sigma_{21})$, with $\Sigma = \Sigma_{12} \cup \Sigma_{21}$, be an instance of the non-uniform correspondence problem, and K_2 an instance such that $(I_1, K_2) \vDash \Sigma$. Then K_2 is a $\oplus$-solution for $(I_1, I_2, \Sigma_{12}, \Sigma_{21})$ if and only if the following conditions are satisfied:*

1. *For all Σ-satisfying homomorphism h with $h(chase_\Sigma^{\mathbf{std}}(I_1, I_2 \cap K_2)) \subseteq (I_1, K_2)$ and all $J_2 \subseteq K_2$ with $(I_1, J_2) \vDash \Sigma$ and $h(chase_\Sigma^{\mathbf{std}}(I_1, I_2 \cap K_2)) \subseteq (I_1, J_2)$, it holds that $J_2 = K_2$.*

2. *There does not exist a tuple $t \in I_2 \smallsetminus K_2$ and a Σ-satisfying homomorphism h with $J_2 \subseteq (I_2 \cup K_2)$, s.t. $(I_1, J_2) \vDash \Sigma$ and $h(chase_\Sigma^{\mathbf{std}}(I_1, I_2 \cap K_2 \cup \{t\})) \subseteq (I_1, J_2)$.*

Proof: Suppose instance K_2 is a $\oplus$-solution. As $(I_1, K_2) \vDash \Sigma$, we have the result of the chase, $chase_\Sigma^{\mathbf{std}}(I_1, K_2) = (I_1, K_2)$. By Lemma 2 there then exists a homomorphism h such that $h(chase_\Sigma^{\mathbf{std}}(I_1, I_2 \cap K_2)) \subseteq (I_1, K_2)$. Toward a contradiction, suppose that condition 1 is violated. This means that there exists an instance J_2 such that $h(chase_\Sigma^{\mathbf{std}}(I_1, I_2 \cap K_2)) \subseteq (I_1, J_2) \subset (I_1, K_2)$ and $(I_1, J_2) \vDash \Sigma$. Then we have $(J_2 \smallsetminus I_2) \subset (K_2 \smallsetminus I_2)$ and, since $(I_1, I_2 \cap K_2) \subseteq chase_\Sigma^{\mathbf{std}}(I_1, I_2 \cap K_2)$ and $(I_1, I_2 \cap K_2)$ is a ground instance, we have that $(I_1, I_2 \cap K_2) \subseteq (I_1, J_2)$. As the two schemata are distinct, it follows that $(I_2 \cap K_2) \subseteq J_2$. Thus $(I_2 \smallsetminus J_2) \subseteq (I_2 \smallsetminus K_2)$ and $J_2 <_{I_2} K_2$. But this contradicts the assumption that K_2 is a $\oplus$-solution. Therefore it must be that $J_2 = K_2$, meaning that K_2 satisfies condition 1.

Suppose then that condition 2 is violated and tuple t, Σ-satisfying homomorphism h and instance J_2 exist. Since for such a J_2 we would have $(I_2 \smallsetminus J_2) \subset (I_2 \smallsetminus K_2)$, it would follow that $J_2 <_{I_2} K_2$, contradicting the assumption that K_2 is a $\oplus$-solution.

For the only if direction let K_2 be an instance such that $(I_1, K_2) \vDash \Sigma$ and K_2 satisfies conditions 1 and 2. We need to show that K_2 is $\oplus$-minimal. If this is not the case there must be an instance J_2, such that $J_2 <_{I_2} K_2$ and $(I_1, J_2) \vDash \Sigma$. Thus $(J_2 \smallsetminus I_2) \subseteq (K_2 \smallsetminus I_2)$ and $(I_2 \smallsetminus J_2) \subseteq (I_2 \smallsetminus K_2)$, and at least one of the inclusions is proper. Suppose first that it were the case that $(I_2 \smallsetminus J_2) = (I_2 \smallsetminus K_2)$. Then we would have $J_2 \subseteq K_2$. So if $J_2 \smallsetminus I_2$ were to be a proper subset of $K \smallsetminus I_2$, it would necessarily be that $J_2 \subset K_2$. Since $(I_2 \cap J_2) = (I_2 \cap K_2)$, $chase_\Sigma^{\mathbf{std}}(I_1, I_2 \cap J_2) = chase_\Sigma^{\mathbf{std}}(I_1, I_2 \cap K_2)$, and by Lemma 2 there is a homomorphism h such that $h(chase_\Sigma^{\mathbf{std}}(I_1, I_2 \cap J)) \subseteq chase_\Sigma^{\mathbf{std}}(I_1, J_2) = (I_1, J_2)$.

But then it would hold that $h(chase_{\Sigma}^{\mathbf{std}}(I_1, I_2 \cap J)) \subseteq (I_1, J_2) \subset (I_1, K_2)$, contradicting condition 1.

On the other hand, if it were the case that $(I_2 \setminus J_2) \subset (I_2 \setminus K_2)$, we would find a tuple $t \in (I_2 \cap J_2) \setminus (I_2 \cap K_2)$, and Lemma 2 would give us a homomorphism h, such that $h(chase_{\Sigma}^{\mathbf{std}}(I_1, I_2 \cap K \cup \{t\})) \subseteq (I_1, J_2) \subset (I_1, I_2 \cup K_2)$, a contradiction to the assumption that K_2 satisfies condition 2$\blacksquare$

In the following subsection we will consider the following decisions problems for the uniform and non-uniform correspondence cases. Note that, even if by Lemma 6 the uniform correspondence problem can be reduced to a corresponding data repair problem, the reverse does not always hold; as in data correspondence problem we consider only correspondence TGD's, which is not the case in data repair.

Definition 34 *Decision problems considered:*

- Existence-of-Solution($\Sigma_{12} \cup \Sigma_{21}$, *subset/superset/*$\oplus$, *uniform/non-uniform*)

  The input is (I_1, I_2) *and the question is whether the uniform/non-uniform correspondence problem* $(I_1, I_2, \Sigma_{12} \cup \Sigma_{21})$ *has a subset/superset/*$\oplus$*-solution;*

- Solution-Checking($\Sigma_{12} \cup \Sigma_{21}$, *subset/superset/*$\oplus$, *uniform/non-uniform*)

  The input is pairs of instances (I_1, I_2) *and* (K_1, K_2) *and the question is whether the* (K_1, K_2) *is a subset/superset/*$\oplus$*-solution for the uniform/non-uniform correspondence problem* $(I_1, I_2, \Sigma_{12} \cup \Sigma_{21})$.

5.3.2 Complexity Results

In this section we are going over all previously defined decision problems using different classes of dependencies and investigate the corresponding data complexity results. The Existence-of-Solution(Σ, superset/$\oplus$, uniform) decision problem is a non-problem for the class of weakly acyclic set of TGD's, since superset-solutions always exist (recall that no denial constraints are present). In this subsection we shall see that the uniform version of the correspondence problem is coNP-complete for weakly acyclic TGD's and for all types of solutions. However, using homomorphism techniques, we are able to show that the non-uniform version of the problem is polynomial for subset and superset-solutions. Somewhat surprisingly then it turns out that the non-uniform version is coNP-complete for $\oplus$-solutions.

Let us first start considering the problems when Σ is specified by a weakly acyclic set of TGD's.

Corollary 6 *Let $\Sigma = \Sigma_{12} \cup \Sigma_{21}$ be a set of weakly acyclic TGD's. Then the problem* EXISTENCE-OF-SOLUTION(Σ, *subset, uniform*) *can be solved in polynomial time.*

Proof: Follows directly from Lemma 6∎

Theorem 42 *Let $\Sigma = \Sigma_{12} \cup \Sigma_{21}$ be a set of weakly acyclic TGD's. Then the problem* SOLUTION-CHECKING(Σ, subset, non-uniform) *can be solved in polynomial time.*

Proof: Let (I_1, I_2) be an instance, Then instance $K_2 \subseteq I_2$ is a subset-solution for $(I_1, I_2, \Sigma_{12}, \Sigma_{21})$, if $(I_1, K_2) \vDash \Sigma$ and there does not exist an instance J_2 such that $K_2 \subset J_2 \subseteq I_2$ and $(I_1, J_2) \vDash \Sigma$. The test $(I_1, K_2) \vDash \Sigma$ can be done in space logarithmic in the size of (I_1, K_2). It is obvious that J_2, as mentioned above, does exist iff there is a tuple $t \in I_2 \smallsetminus K_2$ and homomorphism h such that $h(chase^{\mathbf{std}}_{\Sigma_{21}}(K_2 \cup \{t\})) \subseteq I_1$. It follows from Theorem 33 that the existence of such a homomorphism h can be determined in polynomial time in the size of I_1. The chase with Σ_{21} and the homomorphism test is repeated for each tuple in $I_2 \smallsetminus K_2$. The whole process thus runs in polynomial time in the size of $|I_1| + |I_2|$∎

Theorem 43 *Let $\Sigma = \Sigma_{12} \cup \Sigma_{21}$ be a set of weakly acyclic TGD's. Then the problem* SOLUTION-CHECKING(Σ, superset, non-uniform) *can be solved in polynomial time.*

Proof: Consider the non-uniform correspondence problem (I_1, I_2, Σ). And also consider K_2 the candidate-solution. First test if $(I_1, K_2) \vDash \Sigma$. If so, look for a tuple $t \in K_2 \smallsetminus I_2$ and a homomorphism h from $chase^{\mathbf{std}}_{\Sigma_{12}}(I_1)$ to $K_2 \smallsetminus \{t\}$. By Theorem 33 this process can be carried out in polynomial time in the size of (I_1, K_2). Evidently, if such a tuple t and homomorphism h are found, K_2 is not a superset-solution, since then $(I_1, K_2 \smallsetminus \{t\}) \vDash \Sigma$, meaning that K_2 is not minimal. If no tuple t and homomorphism h is found, K_2 is obviously minimal$\blacksquare$

We just saw that for the non-uniform correspondence problem, checking whether an instance is a subset-solution or a superset-solution is polynomial. If we allow a solution to be in part a subset, and in part a superset the problem becomes coNP-complete.

Theorem 44 *Let* $\Sigma = \Sigma_{12} \cup \Sigma_{21}$ *be a set of tuple generating dependencies. Then the problem* SOLUTION-CHECKING$(\Sigma, \oplus, \text{non-uniform})$ *is in* coNP, *and is* coNP*-hard even for weakly acyclic TGD's.*

Proof: Let (I_1, I_2, Σ) be the non-uniform correspondence problem and K_2 the candidate solution. To see that the problem is in coNP, consider the complementary problem: *Does there exist an instance* J_2 *such that* $(I_1, J_2) \vDash \Sigma$, *and* $J_2 <_{I_2} K_2$*?* This problem is clearly in NP, since we can guess J_2 and then check in logarithmic space whether $(I_2, J_2) \vDash \Sigma$.

For the lower bound we will reduce the POSITIVE 1-IN-3-SAT problem to the $\oplus$-repair checking one. The POSITIVE 1-IN-3-SAT problem asks whether a set of disjunctive clauses, each having three positive variables, is satisfiable with a truth assignment that makes exactly one variable in each clause true. This problem is known to be NP-complete.

We consider the schema $(\mathbf{R}_1, \mathbf{R}_2)$, where $\mathbf{R}_1 = \{P, E, V, F\}$ and $\mathbf{R}_2 = \{T, S, D\}$. Let Σ_{12} consist of the following dependencies:

$$
\begin{aligned}
P(x,y,z) &\rightarrow \exists u,v,w : T(x,u),T(y,v),T(z,w),S(u,v,w) && (5.2)\\
F(u,v,w,u',v',w') &\rightarrow D(u,v,w,u',v',w') && (5.3)
\end{aligned}
$$

and Σ_{21} consist of

$$
\begin{aligned}
T(x,u),T(x,u') &\rightarrow E(u,u') && (5.4)\\
S(u,v,w),S(u',v',w'),D(u,v,w,u',v',w') &\rightarrow V(u,v,w). && (5.5)
\end{aligned}
$$

Note that $\Sigma_{12} \cup \Sigma_{21}$ is a weakly acyclic set.

Given an instance $\mathcal{P}$ of the Positive 1-In-3-SAT problem we construct I_1 as

$$
\begin{aligned}
P^{I_1} &= \{(x,y,z) : x \vee y \vee z \text{ is a clause in } \mathcal{P}\}\\
E^{I_1} &= \{(0,1),\ (1,0)\}\\
V^{I_1} &= \{(0,0,1),\ (0,1,0)\ (1,0,0)\}\\
F^{I_1} &= \{(u,v,w)\ (u',v',w') : u,v,w,u',v',w' \in \{1,0\},\ (u,v,w) \neq (u',v'.w')\}
\end{aligned}
$$

and I_2 as

$$
\begin{aligned}
T^{I_2} &= \{(x,0) : x \text{ variable in } \mathcal{P}\}\\
S^{I_2} &= \{(0,0,1),\ (0,1,0),\ (1,0,0)\}\\
D^{I_2} &= F^{I_1}.
\end{aligned}
$$

Finally, let K_2 be

$$
\begin{aligned}
T^{K_2} &= \{(x,1) : x \text{ is a variable in } \mathcal{P}\}\\
S^{K_2} &= \{(1,1,1)\}\\
D^{K_2} &= D^{I_2}.
\end{aligned}
$$

It is straightforward to verify that $(I_1, K_2) \models \Sigma$. The reduction clearly being polynomial, it remains only to prove that there exists a truth assignment for $\mathcal{P}$ making exactly one variable in each clause true, if and only if K_2 is not a $\oplus$-repair.

Suppose that there exists such a Positive 1-In-3 truth assignment making $\mathcal{P}$ true. Let this truth assignment be represented by a mapping ν from the variables of $\mathcal{P}$ to $\{0,1\}$. Consider then an instance J_2 with

$$\begin{array}{rcl} T^{J_2} & = & \{(x,\nu(x)) : x \text{ is a variable in } \mathcal{P}\} \\ S^{J_2} & = & \{(0,0,1),\ (0,1,0),\ (1,0,0)\} \\ D^{J_2} & = & D^{K_2}. \end{array}$$

It is easy to verify that $(I_1, J_2) \models \Sigma$ and that $J_2 <_{I_2} K_2$. Thus K_2 is not a $\oplus$-repair.

For the other direction, suppose that there does not exist a Positive 1-In-3 truth assignment making $\mathcal{P}$ true. Let us try to construct an instance J_2 such that $(I_1, J_2) \models \Sigma$ and $J_2 <_{I_2} K_2$, by manipulating K_2 to become $\oplus$-closer to I_2.

First we note that dependencies 5.2 and 5.4 force the interpretation of T to contain exactly a truth assignment for each variable and that dependency 5.3 blocks us from deleting tuples from D^{K_2}.

We could make K_2 closer to I_2 by replacing some tuples $(x,1)$ in T^{K_2} with $(x,0)$. But then, in order to satisfy dependency 5.2, we would have to replace the tuple $(1,1,1) \in S^{K_2}$ with at least one tuple containing at least one '0'. Suppose first that there were only one new tuple and that this tuple would contain only one '0'. This would make the resulting instance $\leq_{I_2}$-incomparable with K_2. To avoid this, we could leave tuple $(1,1,1)$ in the interpretation of S. But then the interpretation of S would contain at least two tuples. So, in any case, dependency 5.5 will be triggered, forcing the interpretation of S to contain only tuples from V^{I_1}. We would then have constructed a Positive 1-In-3 truth assignment for $\mathcal{P}$ in the interpretation of T. We have now exhausted all possibilities to improve the solution K_2 and can therefore conclude that K_2 indeed is a $\oplus$-solution$\blacksquare$

The next theorem shows that the result of Afrati and Kolaitis [4] can actually be sharpened to also hold for subset-solution checking for the correspondence problem.

Theorem 45 *There are weakly acyclic sets* $\Sigma = \Sigma_{12} \cup \Sigma_{21}$ *of TGD's such that the problem* SOLUTION-CHECKING$(\Sigma, \text{subset}, \text{uniform})$ *is* coNP*-complete.*

Proof: Let (I_1, I_2, Σ) be a uniform correspondence problem and (K_1, K_2) a candidate solution. As before, in order to see that the problem is in coNP, we will consider complementary problem of deciding if there exists an instance (J_1, J_2) such that $(K_1, K_2) \subset (J_1, J_2) \subseteq (I_1, I_2)$ and $(J_1, J_2) \vDash \Sigma$. This problem is clearly in sf NP as some may guess such an instance (J_1, J_2) and using a polynomial verifier check if $(J_1, J_2) \vDash \Sigma$.

For the lower bound we will reduce the POSITIVE 1-IN-3-SAT to the subset-solution checking problem.

We consider the schemata $\mathbf{R}_1 = \{A, P, E\}$ and $\mathbf{R}_2 = \{T, S, D\}$. The dependencies considered are $\Sigma = \Sigma_{12} \cup \Sigma_{21}$, where Σ_{12} consists of the following:

$$A(n), P(x, y, z) \;\; \to \;\; \exists u, v, w \, : \, T(x, u), T(y, v), T(z, w), S(u, v, w)$$

and Σ_{21} consists of:

$$\begin{array}{rcl} T(x, u), T(x, u'), D(u, u') & \to & E(u) \\ T(x, u) & \to & A(x) \end{array}$$

Now we construct I_1 from the instance $\mathcal{P}$ of the POSITIVE 1-IN-3-SAT to be as:

$$\begin{array}{rcl} A^{I_1} & = & \{(x) \, : \, x \text{ is a variable in } \mathcal{P}\} \\ P^{I_1} & = & \{(x, y, z) \, : \, x \vee y \vee z \text{ is a clause in } \mathcal{P}\} \\ E^{I_1} & = & \varnothing \end{array}$$

and instance I_2 as

$$\begin{array}{rcl} T^{I_2} & = & \{(x,0),(x,1) \,:\, x \text{ is a variable in } \mathcal{P}\} \\ D^{I_2} & = & \{(1,0),(0,1)\} \\ S^{I_2} & = & \{(1,0,0),(0,1,0),(0,0,1)\}. \end{array}$$

Instance K_1 contains all the tuples from I_1 with the exception of tuples from A^{I_1}, that is $A^{K_1} = \varnothing$. Instance K_2 contains the tuples from I_2 with the exception of tuples from T^{I_2}, that is $T^{K_2} = \varnothing$. We have that $(K_1, K_2) \subseteq (I_1, I_2)$ and $(K_1, K_2) \vDash \Sigma$.
This reduction clearly being polynomial, it remains only to prove that there exists a truth assignment for $\mathcal{P}$ making exactly one variable in each clause true if and only if (K_1, K_2) is not a subset-solution to the uniform correspondence problem.

Suppose that there exists such a truth assignment for the variables in $\mathcal{P}$ and let v representing the mapping from variables in $\mathcal{P}$ to $\{0,1\}$ representing this truth assignment. We construct the instance (J_1, J_2) to contain all the tuples from (K_1, K_2) to which we add the following:

$$\begin{array}{rcl} A^{J_1} & = & \{(x) \,:\, x \text{ is a variable in } \mathcal{P}\} \\ T^{J_2} & = & \{(x, v(x)) \,:\, x \text{ variable in } \mathcal{P}\} \end{array}$$

It is easy to verify that (J_1, J_2) satisfies Σ and that $(K_1, K_2) \subset (J_1, J_2) \subseteq (I_1, I_2)$. In other words, (K_1, K_2) is not a solution.

For the other direction, suppose that (K_1, K_2) is not a subset-solution to the uniform correspondence problem. This means that there exists an instance (J_1, J_2) such that $(K_1, K_2) \subset (J_1, J_2) \subseteq (I_1, I_2)$ and $(J_1, J_2) \vDash \Sigma$. As the inclusion between (K_1, K_2) and (J_1, J_2) is proper, it means that there is at least one tuple in $(J_1, J_2) \smallsetminus (K_1, K_2)$. From the construction of (I_1, I_2) it is clear that this tuple is either in instance A^{I_1} or in instance T^{I_2}. Since (J_1, J_2) satisfies Σ_{21}, the second dependency tells us that A^{J_1} is non-empty, implying that the first dependency in Σ_{12} is triggered for each tuple

(x,y,z) in P^{J_1}, thus forcing each variable to have valuation (0 or 1) in T^{J_2}, and by the tuples in S^{J_2}, each "clause" must have exactly one variable valuated to true. The first dependency in Σ_{21} together with the tuples in D^{J_2} ensure that each variable gets only one value in T^{J_2}. Clearly now the values recorded in T^{J_2} constitute a desired truth assignment of $\mathcal{P}$■

From Lemma 6 and Theorem 45 we get

Corollary 7 *There is a weakly acyclic set set $\Sigma = \Sigma_{12} \cup \Sigma_{21}$ of TGD's such that the* SOLUTION-CHECKING$(\Sigma, \oplus, \textit{uniform})$ *problem is* coNP*-complete.*

Let us now turn our attention to the superset problem for the uniform correspondence.

Theorem 46 *There is a weakly acyclic set $\Sigma = \Sigma_{12} \cup \Sigma_{21}$, of TGD's, such that the problem* SOLUTION-CHECKING$(\Sigma, \textit{superset, uniform})$ *is* coNP*-complete.*

Proof: The upper bound follows directly from Lemma 6 and Theorem 37.

For the lower bound we will use a reduction from the same POSITIVE 1-IN-3-SAT problem with a set of dependencies slightly modified from the proof of Theorem 37. Consider the schemata $\mathbf{R}_1 = \{P, R, E\}$ and $\mathbf{R}_2 = \{T, S, D\}$ together with the dependencies $\Sigma = \Sigma_{12} \cup \Sigma_{21}$, with Σ_{12} containing the following set of dependencies:

$$P(x,y,z) \rightarrow \exists u,v,w\, T(x,u), T(y,v), T(z,w), S(u,v,w) \quad (5.6)$$
$$R(x,u), E(y) \rightarrow \exists u'\, T(x,u'), D(u,u'). \quad (5.7)$$

and Σ_{21} containing:

$$T(x,u), T(x,u'), D(u,u') \rightarrow \exists y\, E(y) \quad (5.8)$$
$$T(x,u) \rightarrow R(x,u). \quad (5.9)$$

Let $\mathcal{P}$ be an instance of the Positive 1-In-3-SAT problem. Also consider (I_1, I_2) as an instance and (K_1, K_2) a candidate superset-solution for the corresponding uniform correspondence problem. Similarly to the previous proofs, we will consider the complementary problem of deciding if there exists an instance (J_1, J_2) such that the following holds $(I_1, I_2) \subseteq (J_1, J_2) \subset (K_1, K_2)$ and $(J_1, J_2) \models \Sigma$.

Instance I_1 is constructed from instance $\mathcal{P}$ in the following way:

$$\begin{aligned} P^{I_1} &= \{(x, y, z) : x \vee y \vee z \text{ is a clause in } \mathcal{P}\} \\ R^{I_1} &= \varnothing \\ E^{I_1} &= \varnothing \end{aligned}$$

Instance I_2 is built to contain the following constant tuples:

$$\begin{aligned} T^{I_2} &= \varnothing \\ S^{I_2} &= \{(1,0,0),\ (0,1,0),\ (0,0,1)\} \\ D^{I_2} &= \{(1,0),\ (0,1)\} \end{aligned}$$

The candidate superset-solution (K_1, K_2) is also constructed from instance $\mathcal{P}$, with K_1 containing the following tuples:

$$\begin{aligned} P^{K_1} &= \{(x, y, z) : x \vee y \vee z \text{ is a clause in } \mathcal{P}\} \\ R^{K_1} &= \{(x,0),\ (x,1) : x \text{ variable in } \mathcal{P}\} \\ E^{K_1} &= \{(0)\} \end{aligned}$$

We consider the following tuples for K_2:

$$\begin{aligned} T^{K_2} &= \{(x,0),\ (x,1) : x \text{ variable in } \mathcal{P}\} \\ S^{K_2} &= \{(1,0,0),\ (0,1,0),\ (0,0,1)\} \\ D^{K_2} &= \{(1,0),\ (0,1)\} \end{aligned}$$

This reduction is clearly polynomial and we also have that $(I_1, I_2) \subseteq (K_1, K_2)$ and

$(K_1, K_2) \vDash \Sigma$. It remains to prove that there exists a truth assignment for $\mathcal{P}$ that makes exactly one variable true in each clause, if and only if (K_1, K_2) is not a superset-solution for the given uniform correspondence problem.

First suppose that there exists such a truth assignment for the variables in $\mathcal{P}$ and let v representing the mapping from variables in $\mathcal{P}$ to $\{0,1\}$, representing the truth assignment. Using this mapping we can construct instance (J_1, J_2) as follows:

$$\begin{array}{rcl}
P^{J_1} & = & \{(x,y,z) \,:\, x \vee y \vee z \text{ is a clause in } \mathcal{P}\} \\
R^{J_1} & = & \{(x, v(x)) \,:\, x \text{ variable in } \mathcal{P}\} \\
E^{J_1} & = & \varnothing \\
T^{J_2} & = & \{(x, v(x)) \,:\, x \text{ variable in } \mathcal{P}\} \\
S^{J_2} & = & \{(1,0,0),\ (0,1,0),\ (0,0,1)\} \\
D^{J_2} & = & \{(1,0),\ (0,1)\}
\end{array}$$

Now clearly we have $(I_1, I_2) \subseteq (J_1, J_2) \subset (K_1, K_2)$ and also, because v maps the variables from $\mathcal{P}$ to the Positive 1-In-3-SAT solution assignment, we have that $(J_1, J_2) \vDash \Sigma$, that is (K_1, K_2) is not a superset-solution to the given uniform correspondence settings.

For the other direction suppose that (K_1, K_2) is not a superset-solution for the given uniform correspondence problem. This means that there exists an instance (J_1, J_2) properly included in (K_1, K_2) such that $(J_1, J_2) \vDash \Sigma$. As the inclusion is proper, it implies that the set $(K_1, K_2) \smallsetminus (J_1, J_2)$ is not empty. On the other hand, we know that (J_1, J_2) includes all the tuples from (I_1, I_2). This implies that the set $(K_1, K_2) \smallsetminus (J_1, J_2)$ may contain only tuples corresponding to relational symbols T, R or E. But we have a transitive relation forced by dependencies 5.8 and 5.9, that is if we take a tuple from R^{K_1} then in order to satisfy 5.9, we also have to remove a corresponding tuple from T^{K_2} implying from 5.8 that we need to remove also the tuple E^{K_1} as well. From this it follows that $E^{J_1} = \varnothing$, that is (using dependency 5.8) for each variable in $\mathcal{P}$ we have a single truth assignment given by instance T^{J_2}. But now from dependency 5.6, it follows that for each clause in $\mathcal{P}$ we have only one variable evaluated to true. Consequently the assignment given by the instance T^{J_2} is a solution for Positive 1-In-3-SAT problem over instance $\mathcal{P}$ $\blacksquare$

Let us now consider the existence of the solution problem for the non-uniform correspondence settings. For this, as it will be proved, all the three cases *subset/superset/*$\oplus$ are intractable for general weakly acyclic set of tuple generating dependencies.

Theorem 47 *There is a weakly acyclic set* $\Sigma = \Sigma_{12} \cup \Sigma_{21}$ *such that the problem* Existence-of-Solution(Σ, *subset, non-uniform*) *is* NP*-complete.*

Proof: Let (I_1, I_2, Σ) be a non-uniform correspondence problem. We need to decide if there exists an instance J such that $J \subseteq I_2$ and $(I_1, J) \models \Sigma$. This problem is clearly in NP as some may guess such an instance and the verifier is polynomial.

For the lower bound we will use the same Positive 1-In-3-SAT problem to be reduced to an existence subset-solution problem over a uniform correspondence setting. For this let's consider the schemata $\mathbf{R}_1 = \{P, V, E\}$ and $\mathbf{R}_2 = \{T, S, D\}$. Let us also consider the following set of dependencies for Σ_{12}:

$$P(x,y,z) \rightarrow \exists u,v,w\, T(x,u), T(y,v), T(z,w), S(u,v,w) \tag{5.10}$$

$$V(u,u') \rightarrow U(u,u') \tag{5.11}$$

and for Σ_{21} the following set of dependencies:

$$T(x,u), T(x,u'), D(u,u') \rightarrow E(u) \tag{5.12}$$

Now, given an instance $\mathcal{P}$ of the Positive 1-In-3-SAT problem, we construct instance I_1 as:

$$\begin{array}{rcl} P^{I_1} & = & \{(x,y,z) \,:\, x \vee y \vee z \text{ is a clause in } \mathcal{P}\} \\ V^{I_1} & = & \{(0,1),(1,0)\} \\ E^{I_1} & = & \varnothing \end{array}$$

and instance I_2 as:

$$\begin{array}{rcl} T^{I_2} & = & \{(x,0),(x,1) : \text{x variable in } \mathcal{P}\} \\ D^{I_2} & = & \{(1,0),(0,1)\} \\ S^{I_2} & = & \{(1,0,0),(0,1,0),(0,0,1)\}. \end{array}$$

The reduction is clearly polynomial. It remains only to prove that there exists a truth assignment for $\mathcal{P}$ making exactly one variable in each clause true if and only if there exists a subset-solution for the non-uniform correspondence problem given by $(I_1, I_2, \Sigma_{12}, \Sigma_{21})$.

First let us suppose that there exists a solution for the Positive 1-In-3-SAT problem over instance $\mathcal{P}$. Consider v to be the mapping from variables in $\mathcal{P}$ to $\{0,1\}$ given by the solution. We can now construct the following instance J over schema $\mathbf{R}_2$:

$$\begin{array}{rcl} T^{J} & = & \{(x,v(x)) : \text{x variable in } \mathcal{P}\} \\ D^{J} & = & \{(1,0),(0,1)\} \\ S^{J} & = & \{(1,0,0),(0,1,0),(0,0,1)\}. \end{array}$$

It is clear that $J \subseteq I_2$ and $(I_1, J) \vDash \Sigma$. That is if there exists a Positive 1-In-3-SAT truth assignment for instance $\mathcal{P}$, then there exists a subset-solution for the created non-uniform correspondence problem.

For the other direction, let us suppose that for the given non-uniform correspondence instance (I_1, I_2) there exists an instance J over $\mathbf{R}_2$ such that $J \subseteq I_2$ and $(I_1, J) \vDash \Sigma$. From this last assumption it follows that (by using dependency 5.10) there exist truth and assignment for each variable in $\mathcal{P}$, given by T^J, such that it makes true exactly one variable from each clause. On the other hand, dependency 5.11 ensures that the instance D^J contains the same tuples as V^{I_1}. This ensures, by using dependency 5.12, that there exists a unique assignment for each variable. From these it follows that the assignment given by instance T^J is a POSITIVE 1-IN-3-SAT truth assignment for instance $\mathcal{P}$∎

We get the same result if we consider the problem of the existence for a superset-solution over a non-uniform correspondence setting.

Theorem 48 *There is a set* $\Sigma = \Sigma_{12} \cup \Sigma_{21}$ *of weakly acyclic TGD's such that the problem* EXISTENCE-OF-SOLUTION(Σ, *superset, non-uniform*) *is* NP*-complete.*

Proof: Let (I_1, I_2) be an instance. The problem is to decide if there exists an instance J such that $J \supseteq I_2$ and $(I_1, J) \vDash \Sigma$. This problem is clearly in NP as some may guess such an instance over a finite domain given by the domain of $dom(I_1) \cup dom(I_2)$. Also the verifier is polynomial.

Again, for the lower bound we will use POSITIVE 1-IN-3-SAT problem to be reduced to an existence subset-solution problem over a uniform correspondence setting. For this let's consider the schemata $\mathbf{R}_1 = \{P, V, E\}$ and $\mathbf{R}_2 = \{T, S, D\}$. Let us also consider the following set of dependencies for Σ_{12}:

$$P(x,y,z) \quad \rightarrow \quad \exists u,v,w\, T(x,u), T(y,v), T(z,w), S(u,v,w) \tag{5.13}$$

and for Σ_{21} the following dependencies:

$$\begin{aligned} T(x,u), T(x,u'), D(u,u') &\rightarrow E(u,u,u) && (5.14)\\ D(u,v) &\rightarrow V(u,v) && (5.15)\\ S(u,v,w) &\rightarrow E(u,v,w). && (5.16) \end{aligned}$$

Given an instance $\mathcal{P}$ for the POSITIVE 1-IN-3-SAT problem, we construct instance I_1 as follows:

$$\begin{aligned}
P^{I_1} &= \{(x,y,z) \,:\, x \vee y \vee z \text{ is a clause in } \mathcal{P}\}\\
V^{I_1} &= \{(1,0),(0,1)\}\\
E^{I_1} &= \{(1,0,0),(0,1,0),(0,0,1)\}.
\end{aligned}$$

The instance I_2 is considered to be empty. Using the same method as in Theorem 47, it can be proved that there exists a POSITIVE 1-IN-3-SAT truth assignment for $\mathcal{P}$ if and only if there exists a superset-solution for the given instance of the non-uniform correspondence problem■

The last result for the hard cases is the checking for existence of a $\oplus$-solution over a non-uniform correspondence settings using weakly acyclic tuple generating dependencies.

Theorem 49 *There is a weakly acyclic set $\Sigma = \Sigma_{12} \cup \Sigma_{21}$, of TGD's such that the problem* EXISTENCE-OF-SOLUTION$(\Sigma, \oplus, \textit{non-uniform})$ *is* NP*-complete.*

Proof: Let (I_1, I_2) be an instance. The problem is to decide if there exists an instance J over $\mathbf{R}_2$ such that $(I_1, J) \models \Sigma$. As in the superset case, one may guess such an instance over a finite domain given by $dom(I_1) \cup dom(I_2)$. Also, the verifier is clearly polynomial for a set of weakly acyclic TGD's. Observe that the found instance J is not necessary a $\oplus$-solution but the existence of such an instance guarantees the existence of a $\oplus$-solution. This means that the problem is in NP.

For the lower bound we will use the reduction from POSITIVE 1-IN-3-SAT problem to the $\oplus$-solution over a non-uniform correspondence setting. We construct the instance $(I_1, I_2, \Sigma_{12} \cup \Sigma_{21})$ using schemata $\mathbf{R}_1 = \{P, E, V\}$ and $\mathbf{R}_2 = \{T, S\}$. The dependency considered for Σ_{12} is:

$$P(x, y, z) \rightarrow \exists u, v, w\, T(x, u), T(y, v), T(z, w), S(u, v, w) \tag{5.17}$$

And the dependencies in Σ_{21} will be:

$$T(x, u), T(x, u') \rightarrow E(u, u') \tag{5.18}$$

$$S(u, v, w) \rightarrow V(u, v, w) \tag{5.19}$$

Given an instance $\mathcal{P}$ of the Positive 1-In-3-SAT problem, we construct the instance I_1 as:

$$\begin{aligned} P^{I_1} &= \{(x, y, z) : x \vee y \vee z \text{ is a clause in } \mathcal{P}\} \\ E^{I_1} &= \{(0,0), (1,1)\} \\ V^{I_1} &= \{(0,0,1), (0,1,0), (1,0,0)\} \end{aligned}$$

and instance I_2 is built as follows (it can be noted that, for symmetric difference in a non-uniform environment, the instance I_2 doesn't play a key role, that is any instance may be considered):

$$\begin{aligned} S^{I_2} &= \{(0,0,1), (0,1,0), (1,0,0)\} \\ T^{I_2} &= \varnothing \end{aligned}$$

This reduction is clearly polynomial. Using a similar proof path as in the previous theorem, it can be proved that there exists a Positive 1-In-3-SAT truth assignment for instance $\mathcal{P}$ if and only if there exists a $\oplus$-solution for the non-uniform correspondence problem■

We will turn now our attention to the semi-LAV dependency class that, as we will see, ensures polynomial time for all the previous mentioned problems.

Theorem 50 *Let $\Sigma = \Sigma_{12} \cup \Sigma_{21}$ be a set of semi-LAV TGD's. Then the problem* $\textsc{Existence-of-Solution}(\Sigma, \text{subset}, \text{non-uniform})$ *can be solved in polynomial time.*

Proof: Let $(I_1, I_2, \Sigma_{12} \cup \Sigma_{21})$ be an instance of the non-uniform correspondence. In order to determine if there exists an instance $K_2 \subseteq I_2$, such that $(I_1, K_2) \models \Sigma$, it suffices to determine if there exists a Σ-satisfying homomorphism from $\mathit{chase}_{\Sigma}^{\mathbf{std}}(I_1, \varnothing)$ to (I_1, I_2). Since Σ is a set of semi-LAV TGD's, it follows from Theorems 39 and 33 that the latter condition can be tested in polynomial time∎

Theorem 51 *Let $\Sigma = \Sigma_{12} \cup \Sigma_{21}$ be a set of semi-LAV TGD's. Then the problem* $\textsc{Existence-of-Solution}(\Sigma, \mathit{superset}, \mathit{non\text{-}uniform})$ *is polynomial.*

Proof: Let $(I_1, I_2, \Sigma_{12} \cup \Sigma_{21})$ be an instance of the non-uniform correspondence problem. First, compute $(K_1, K_2) = \mathit{chase}_{\Sigma}^{\mathbf{std}}(I_1, I_2)$. Then, search for a homomorphism h such that $h(K_1) \subseteq I_1$ and $(h(K_1), h(K_2)) \models \Sigma$. The crucial point is that (K_1, K_2) has bounded blocksize, since Σ is semi-LAV∎

Theorem 52 *Let $\Sigma = \Sigma_{12} \cup \Sigma_{21}$ be a set of semi-LAV dependencies. Then the problem* $\textsc{Existence-of-Solution}(\Sigma, \oplus, \mathit{non\text{-}uniform})$ *is polynomial.*

Proof: Search for a homomorphism h, such that $(I_1, h(\mathit{chase}_{\Sigma_{12}}^{\mathbf{std}}(I_1))) \models \Sigma$. From Theorem 39 we know that we can find this in polynomial time∎

From Theorem 40 we conclude the following corollary:

Corollary 8 *Let $\Sigma = \Sigma_{12} \cup \Sigma_{21}$ be a set of semi-LAV dependencies. The problems* $\textsc{Solution-Checking}(\Sigma, \text{subset}/\text{superset}/\oplus, \text{ uniform})$ *can be solved in polynomial time.*

Proof: Follows directly from Lemma 6 and Theorem 40. ∎

Theorem 53 *Let Σ be a set of semi-LAV tuple generating dependencies. Then the problems* SOLUTION-CHECKING$(\Sigma, \text{subset}/\text{superset}/\oplus, \text{non-uniform})$ *are polynomial.*

Proof: The *subset/superset* cases follow directly from Theorems 42 and 43 respectively. In the $\oplus$-case we need to check if an instance K is a $\oplus$-solution for the non-uniform correspondence problem of (I_1, I_2) w.r.t. $\Sigma = \Sigma_{12} \cup \Sigma_{21}$. First, test if $K \vDash \Sigma$. If so, K is indeed a $\oplus$-solution for the non-uniform correspondence problem if and only if both conditions in Theorem 41 are fulfilled. Since Σ is semi-LAV, the required homomorphism tests can be done in polynomial time, as per Theorem 39■

We can now conclude this section with two tables summarizing the data complexity results for the correspondence solution-existence and for the correspondence solution-check problems under different dependency classes.

EXISTENCE Σ	unif. corr. sol. weakly acyc.	non-unif. corr. sol. weakly acyc.	non-unif. corr. sol. semi-LAV
subset	P *Cor. 6*	NPC *Thrm. 47*	P *Thrm. 50*
superset	-	NPC *Thrm. 47*	P *Thrm. 51*
$\oplus$	-	NPC *Thrm. 47*	P *Thrm. 52*

Figure 5.3: Data complexity for solution-existence problem

SOLUTION CHECK Σ	uniform weakly acyc.	uniform semi-LAV	non-uniform weakly acyc.	non-uniform semi-LAV
subset	coNPC *Thrm. 45*	P *Thrm. 53*	P *Thrm. 42*	P *Thrm. 42*
superset	coNPC *Thrm. 45*	P *Thrm. 53*	P *Thrm. 43*	P *Thrm. 43*
$\oplus$	coNPC *Thrm. 45*	P *Thrm. 53*	coNPC *Thrm. 44*	P *Thrm. 53*

Figure 5.4: Data complexity for the solution-check problem

Chapter 6

Closed World Chasing

In Chapter 5 we studied the chase procedure applicability in Data Exchange, Data Repair and Data Correspondence problems. In Section 5.1 dedicated to data exchange, we saw that the chase procedure could be used to materialize a "universal solution" on the target that may be used, after all, to get certain answers to any UCQ query. In order to deal with a larger class of queries, Deutsch Nash and Remmel [23] introduced the extended core chase to materialize on the target a set of universal solutions which could answer a more larger class of queries, namely $\mathsf{UCQ}^{\neg,\neq}$. The semantics for certain query answering in both cases is closed to the left and open to the right, that is the source instance is considered closed, but the target instance open. As pointed out by Libkin in [54], the tacit open world assumption creates anomalies in the (first order) query evaluation even in the simplest data exchange setting where the target is declared to be a copy of the source. To see this anomaly introduced by the open world semantics, consider the following example:

Example 34 *Recall the data exchange mapping M without target dependencies, from Example 29, that simply copies the source instance to the target. Thus, as one may* *expect, the certain answer to queries over the source instance should return the same result when running the similar query on the materialized target instance. Consider mapping* $M = (\{\mathbf{R}\}, \{\mathbf{R'}\}, \Sigma_{st}, \varnothing)$ *with* $\Sigma_{st} = \{R(x,y) \to R'(x,y)\}$. *Using the extended core-chase algorithm presented in Section 3.2 with source instance* $I = \{R(a,b)\}$, *we get* $\mathcal{J} = \{J_1, J_2, J_3, J_4, J_5, J_6, J_7, J_8\}$ *as the universal solution set materialized on the target, where the instances* J_1 *to* J_8 *are tabularly represented as follows:*

J_1
$R'(a,b)$

J_2
$R'(a,b)$
$R'(a,a)$

J_3
$R'(a,b)$
$R'(b,b)$

J_4
$R'(a,b)$
$R'(a,a)$
$R'(b,b)$

J_5
$R'(a,b)$
$R'(b,a)$

J_6
$R'(a,b)$
$R'(a,a)$
$R'(b,a)$

J_7
$R'(a,b)$
$R'(b,b)$
$R'(b,a)$

J_8
$R'(a,b)$
$R'(a,a)$
$R'(b,b)$
$R'(b,a)$

Considering the boolean $\mathsf{UCQ}^{\neg}$ *query over the target schema* $q' \leftarrow R'(x,y) \wedge \neg R'(x,x)$, *the certain answer for* q' *over the materialized target universal set* $\mathcal{J}$ *is* **false**. *On the other hand, running the query* $q \leftarrow R(x,y) \wedge \neg R(x,x)$ *over the source instance we get the certain answer* **true**.

Moreover, even under the open world (OWA) semantics, the universal solution set returned by the extended core chase is not suitable to be used to answer arbitrary FO queries, as it is shown in the following example:

Example 35 *Consider data exchange mapping* $M = (\{R\}, \{S, T\}, \{R(x) \to S(x)\}, \varnothing)$. *Then consider source instance* $I = \{R(a)\}$ *and the boolean query over the target instance* $q \leftarrow T(x) \to S(x)$ *(note that query* q *is not an* $\mathsf{UCQ}^{\neg,\neq}$, *thus not contradicting the results from [23]). Clearly, if one may use an open world assumption on the target, the instance*

$J = \{S(a), T(b)\}$ is a possible instance on the target schema for mapping M and source instance I. It can be expected that the certain answer for query q to be **false**. *By using the extended core-chase algorithm for this setting the following universal solution set $\{\{S(a)\}, \{S(a), T(a)\}\}$ is returned. The certain answer returned by running query q on this universal solution set is, contrary to the expected result,* **true**.

From the previous examples it follows that the OWA semantics does not ensure intuitive answers for most of the non-monotonic queries. Thus, some sort of closed world (CWA) semantics is needed in data exchange. In this chapter we will review a few of the close world semantics considered in deductive databases and see why they are not suitable for Data Exchange. Also, we will investigate some of the recently proposed closed world semantics for Data Exchange and see some of the issues associated with these approaches. We will then introduce a new closed world semantics, the constructible models semantics, and argue that it is a suitable semantics for data exchange. Finally, we will show that under this new constructible models semantics one may materialize a conditional table on the target that may be used to obtain certain and possible answers to any FO query under the constructible models semantics.

6.1 Closed World Semantics for Data Exchange

Let us now take a brief look at the main close world semantics proposed in deductive databases and see why these are not suitable for Data Exchange. We will not give a formal definition of all of these closed world semantics, but rather we will give the intuition behind these semantics and show how they are reflected in Data Exchange. For this we will consider a data exchange mapping $M = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$, with $\Sigma = \Sigma_{st} \cup \Sigma_t$, and a source instance I. In the following, by *data exchange solution space*, we mean the set of possible solutions to a given data exchange problem under a given CWA semantics.

Closed World Assumption (**CWA**). The closed world assumption semantics was first introduced by Reiter [68] in reference to non-monotonic query answering in deductive databases. Reiter considers in his model any fact from the database that cannot be directly inferred to be false. More formally, the data exchange solution space under Reiter's semantics can be specified as:

$$\begin{aligned} Sol_M^{\mathtt{CWA}}(I) = \{J \in Inst(\mathbf{T}) \mid \ & J \cup I \vDash \Sigma \text{ and } J \subseteq K \\ & \text{for all } K \in Inst(\mathbf{T}) \text{ such that } K \cup I \vDash \Sigma\} \end{aligned} \tag{6.1}$$

Unfortunately, even if this semantics works well for deductive database specified by a set of Datalog rules, it does not work as expected when considering a data exchange mapping containing existentially quantified TGD's. Consider for example the source-to-target TGD :

$$Employee(\mathrm{emp}) \rightarrow \exists \mathrm{mgr}\ Manages(\mathrm{emp}, \mathrm{mgr}), Manager(\mathrm{mgr}) \tag{6.2}$$

Intuitively, the dependency mentions that there exists a manager for each employee and that the list of managers is stored under relation $Manager$. Consider now source instance I consisting of a single tuple $Employee(\mathrm{john})$ and the boolean query over the target schema: $q \leftarrow Manages(\mathrm{emp}, \mathrm{mgr})$. Thus the query returns **true** if there exists at least one employee in the $Manages$ relation. Clearly under the specified settings, one may expect this query to return **true**. Even more, as $q \in \mathsf{UCQ}$ it is easy to see that the certain answer under OWA semantics for this query is **true**. On the other hand, for the given data exchange mapping and instance $Sol_M^{\mathtt{CWA}}(I) = \varnothing$ that is the certain answer to q under CWA semantics is **false**. Clearly this is not the expected result as the given dependency specifies that there needs to exists at least one manager for each employee. From this we can conclude that the CWA semantics is not a good approach for data exchange mappings specified by existential TGD's.

Generalized Closed World Assumption (**GCWA**). As the name suggests it, GCWA semantics, introduced by Minker [66], generalizes CWA such that a data exchange solution under this semantics needs not to be a subset of each model but to be a subset of some union of subset minimal models. Before presenting the solution space for data exchange under GCWA semantics, let us introduce the notion of subset minimality w.r.t. a set of dependencies Σ:

$$\begin{aligned}\subseteq^{\text{min}}(M,I) = \{J \in Inst(\mathbf{T}) |\ J \cup I \vDash \Sigma \text{ and } \nexists\, K \text{ such that}\\ K \in Inst(\mathbf{T}),\ K \cup I \vDash \Sigma \text{ and } K \subsetneq J\}\end{aligned} \tag{6.3}$$

Similarly to the CWA case, we are now ready to present the solution space under the GCWA semantics for the data exchange mappings as the following set of ground instances:

$$Sol_M^{\texttt{GCWA}}(I) = \{J \in Inst(\mathbf{T}) |\ J \cup I \vDash \Sigma \text{ and } J \subseteq \bigcup_{i \in [n]} K_i \text{ for some } K_i \in \subseteq^{\text{min}}(M,I)\} \tag{6.4}$$

Consider under this semantics the same data exchange mapping from the CWA semantics example:

$$Employee(\text{emp}) \to \exists \text{mgr}\ Manages(\text{emp}, \text{mgr}), Manager(\text{mgr}) \tag{6.5}$$

Consider also the boolean query q that returns **true** if for each manager there exists at least one employee managed by the manager. Thus the query can be formally stated as the following FO expression:

$$q \leftarrow \forall \text{mgr}\ Manager(\text{mgr}) \to \exists \text{emp}\ Manages(\text{emp}, \text{mgr}) \tag{6.6}$$

Clearly from the definition of the source-to-target dependency, we would expect this query to return **true**for any non-empty source instance. Let I be the instance consisting of a single tuple *Employee*(john). The data exchange solution space for these settings under GCWA semantics is the following infinite set of instances:

J_1	J_2	J_3	
$Manages(\text{john}, \text{john})$	$Manages(\text{john}, \text{ray})$	$Manages(\text{john}, \text{ray})$	$\dots$
$Manager(\text{john})$	$Manager(\text{ray})$	$Manager(\text{john})$	
		$Manager(\text{ray})$	

For example, instance J_3 is part of $Sol_M^{\texttt{GCWA}}(I)$ as $J_3 \cup I \models \Sigma_{st}$ and $J_3 \subset J_1 \cup J_2$. From this it follows that certain answer to q under this semantics is **false**. As mentioned, clearly this is not the expected result.

Possible World Semantics(**PWS**). The possible world semantics was introduced by Chan [20] in the context of disjunctive databases. The solution space under this semantics can be extended for data exchange settings without target dependencies and it is formally defined as:

$$\begin{aligned} Sol_M^{\texttt{PWS}}(I) = \{J \in Inst(\mathbf{T}) \mid \; & J \cup I \models \Sigma_{st} \text{ and } \forall t \in J,\ \exists\big(\alpha(\bar{x}) \rightarrow \exists \bar{y}\, \beta(\bar{x}, \bar{y})\big) \in \Sigma_{st} \\ & \text{and } \exists\ \bar{a}, \bar{b} \text{ such that } I \models \alpha(\bar{a}) \\ & \text{and } J \models \beta(\bar{a}, \bar{b}) \text{ and } t \in head(\alpha(\bar{a}) \rightarrow \beta(\bar{a}, \bar{b}))\} \end{aligned} \tag{6.7}$$

Consider the previously used dependency:

$$Employee(\text{emp}) \rightarrow \exists \text{mgr}\ Manages(\text{emp}, \text{mgr}), Manager(\text{mgr}) \tag{6.8}$$

Also consider instance $I = Employee(\text{john})\}$. For this data exchange setting the following infinite set of instances represents the solution space under PWS semantics:

K_1	K_2	K_3	
$Manages(\text{john}, \text{john})$	$Manages(\text{john}, \text{ray})$	$Manages(\text{john}, \text{john})$	
$Manager(\text{john})$	$Manager(\text{ray})$	$Manages(\text{john}, \text{ray})$	$\dots$
		$Manager(\text{john})$	
		$Manager(\text{ray})$	

Note that, compared to the GCWA semantics, instance J_3 is not present in the PWS solution space. Thus the certain answer to the given query will be **true**, as expected.

Let us now consider another example with the mapping specified by the following source-to-target TGD:

$$Manages(\text{emp}, \text{mgr}) \rightarrow \exists \text{eid}\, ManageIds(\text{eid}, \text{mgr}) \tag{6.9}$$

Intuitively, this TGD replaces the employee names from relation $Manages$ with employee id's in relation $ManageIds$. Consider now the instance:

$$I = \{Manages(\text{john}, \text{ray}), Manages(\text{mike}, \text{ray})\}. \tag{6.10}$$

Based on the source-to-target TGD and the given instance we have the following infinite PWS solution space:

U_1	U_2	U_3	
$ManageIds(101, \text{ray})$	$ManageIds(101, \text{ray})$	$ManageIds(101, \text{ray})$	
	$ManageIds(102, \text{ray})$	$ManageIds(102, \text{ray})$	$\dots$
		$ManageIds(210, \text{ray})$	

Let us now consider the following boolean query: *Does each manager have at most two employees reporting to him?* The expected result, based on the source instance I, is **true**. Using the PWS solution space to get the certain answer to this query, the result is **false**. This is because in PWS solution space the same variable assignment to a TGD may result in multiple tuples.

Before presenting the proposed semantic for data exchange let us review some new semantics specially defined for the data exchange problem.

CWA-solutions. The CWA-solutions approach was first introduced by Libkin [54] for mappings specified by a set of source-to-target TGD's. This approach is based on the intuition that any fact in the target instance should follow logically from the source instance and the set of dependencies, and also that no two nulls in the target should be gratuitously equated. This is formalized using a new notion called *justification*. These justifications conveniently restrict the solutions to the universal ones, called *closed world solutions* in [54], here denoted as $Sol_M^{\texttt{CWA}_{\texttt{SOL}}}(I)$. Hernich and Schweikardt [47] extended Libkin's justification based approach by allowing the mapping to be specified by target TGD's and they generalize the set $Sol_M^{\texttt{CWA}_{\texttt{SOL}}}(I)$. Their solution space has the peculiarity that there are generalized instances $J \in Sol_M^{\texttt{CWA}_{\texttt{SOL}}}(I)$ such that $rep^{CWA}(J) \cup I \nvDash \Sigma_{st} \cup \Sigma_t$. The certain answer to a target query q under the previously presented data exchange settings is defined as:

$$\bigcap_{J \in Sol_M^{\texttt{CWA}_{\texttt{SOL}}}(I)} \quad \bigcap_{K \in rep^{CWA}(J), K \cup I \vDash \Sigma} q(K). \tag{6.11}$$

Without going in too much detail, we need to mention that it is enough to materialize the set of “maximal“ CWA-solutions in order to get certain answers to FO-queries. Consider for example the data exchange mapping M specified by the following set of TGD's:

$$\Sigma_{st}: \quad Scholar(s) \rightarrow Student(s) \tag{6.12}$$

$$Scholar(s) \rightarrow \exists m_1, m_2\ Major(s, m_1, m_2) \tag{6.13}$$

$$\Sigma_t: \quad Major(s, m, m) \rightarrow StdOneMajor(s) \tag{6.14}$$

Consider the source instance $I = \{Scholar(\text{john})\}$. The unique maximal CWA-solution is instance $J = \{Student(\text{john}), Major(\text{john}, M_1, M_2)\}$. Now, for the following query $q(x) \rightarrow Student(x) \wedge \neg StdOneMajor(x)$, one would expect the certain answer to return the empty set, as it may be possible that student “john“ has only one major. On the other hand, running the query against the maximal solution, the certain answer for this query is “john“. Clearly not the expected result.

GCWA*. We conclude this section with the latest closed world semantics, namely GCWA*, introduced by Hernich [45], as an improvement for some of the previously described anomalies. In certain query answering, GCWA* semantics was also proved to be closed under logical equivalence of mappings. The new semantics generalizes the GCWA one by considering a ground instance part of the data exchange solution space if the instance is the union of some minimal solutions. More formally, the solution space under GCWA* can be described as the following set:

$$Sol_M^{\texttt{GCWA}^*}(I) = \{J \in Inst(\mathbf{T})|\ J \cup I \vDash \Sigma,\ J = \bigcup_{i \in [n]} K_i \text{ for some } K_i \in \subseteq^{\min} (M, I)\} \quad (6.15)$$

For this semantics, Hernich proved that when the mappings are specified by a special class of TGD’s, called packed TGD’s, and if the core of the universal solution is given, then one may compute the certain answer in polynomial time. In case the core of the universal solution is not provided and the core-chase algorithm terminates, then one may find the certain answer in NP time with a DP oracle ([28]).

Let us now see an example of certain answer under GCWA* semantics. Consider a mapping M specified by the following two source-to-target TGD’s:

$$Manages(\text{emp}, \text{mgr}) \to \exists \text{eid}\ ManageIds(\text{eid}, \text{mgr}) \quad (6.16)$$

$$SelfManager(\text{mgr}, \text{mgrid}) \to ManageIds(\text{mgrid}, \text{mgr}) \quad (6.17)$$

Where the first dependency has the same meaning as the second example used for PWS semantics, and the second dependency specifies that each manager with the id (“mgrid“) from the relation $SelfManager$ is his own manager under target relation $ManageIds$. Let us now add to this mapping the source instance:

$$I = \{Manages(\text{john}, \text{ray}), Manages(\text{mike}, \text{ray}), SelfManager(\text{ray}, 411)\}. \quad (6.18)$$

Finally, consider the query: *Does each manager manage exactly one employee?* It is easy to see that under natural semantics one may expect the certain answer to this query to be **false** , as it may be that the employee id's, assigned for each employee from the $Manages$ relation, are distinct. On the other hand, the set $\subseteq^{\min}$ $(M, I$ contains only the instance $J = \{ManageIds(\text{ray}, 411)\}$. Thus, under the GCWA* semantics the certain answer to the given query is **true**, which is clearly not the expected result. The same unnatural behavior can be observed even in case the mapping is specified only by the first dependency with the same source instance and the query: *Do all the managers manage at most 2 employees each?* In this case, the certain answer will be **false** under GCWA*, because the solution space is infinite, whereas the expected natural result should be **true**.

6.2 Constructible Models Semantics

In the previous section we saw a few closed world semantics and their unnatural behavior when getting the certain answers for some general queries. In this section (part of these results were first presented in [38]) we propose a new closed world semantics which, we argue, is a good candidate as a closed world semantics for data exchange, even if there are a few drawbacks with it. One such drawback is that the constructible models semantics are not closed under the mapping equivalence, as shown by the following example:

Example 36 *Consider the following two sets of source-to-target mappings:*

$$\begin{array}{rcl} \Sigma_1 & = & \{R(x,y) \to T(x,y)\} \\ \Sigma_2 & = & \{R(x,y) \to T(x,y);\ R(x,y) \to \exists z\, T(x,z)\} \end{array}$$

Clearly Σ_1 is logically equivalent with Σ_2, i.e. $I \models \Sigma_1$ iff $I \models \Sigma_2$ for any instance I. On the other hand considering source instance $I = \{R(a,b)\}$ under constructible models semantics we have $\{\{T(a,b)\}\}$ as the solution space under Σ_1 and the infinite set $\{\{T(a,b)\}, \{T(a,b), T(a,a)\}, \{T(a,b), T(a,c)\}, \ldots\}$ as the solution space under Σ_2.

Here are some strong points attached to this new semantics:

1. *natural certain/possible answers for most FO queries.* At the end of this section we will review all the previous examples and see that the certain answer obtained under this new semantic is the expected one.

2. *there exists a simple chase based procedure able to compute the target instance that may be used to compute certain answers.*

3. *in case the chase procedure terminates, there exists a single instance, usually of small size, that needs to be materialized on the target.*

Let I be a ground instance and Σ a set of TGD's. Consider a standard-chase step on I with TGD $\xi = \alpha(\bar{x}, \bar{y}) \rightarrow \exists \bar{z}\, \beta(\bar{x}, \bar{z})$ and trigger (ξ, h). Instead of adding the tuple $head(\beta(h(\mathbf{x}), \bar{z}))$, where $\bar{Z}$ is a sequence of new fresh variables, we create a branch for each sequence $\bar{c}$ of constants and add the ground tuple $\beta(h(\bar{x}), \bar{c})$ to I. Continuing in this fashion, we will have a *chase tree*, instead of a chase sequence. Notably, all nodes in the tree will be ground instances. The leaves of the chase tree form the set of constructible models of I and Σ, denoted $\Sigma(I)$. If the initial input to the chase is a general instance (that may contain nulls) J, the constructible solutions of J with Σ will be $\Sigma(rep^{CWA}(J)) = \bigcup_{I \in rep^{CWA}(J)} \Sigma(I)$.

More formally, let Σ be a finite set of TGD's. Then the set of variables occurring in Σ is finite. Consequently, there is an enumeration, say $v_0, v_1, \ldots, v_n, \ldots$, of all mappings (i.e. valuations) from this set of variables to Δ_{C}. We define

$$\mathit{ground}\big(\forall \bar{x}, \bar{y}\ \alpha(\bar{x}, \bar{y}) \to \exists \bar{z}\ \beta(\bar{x}, \bar{z})\big) = \bigcup_{i\in\omega}\bigcup_{j\in\omega}\bigcup_{k\in\omega} \big\{\alpha(v_i(\mathbf{x}), v_j(\mathbf{y})) \to \beta(v_i(\bar{x}), v_k(\bar{z}))\big\}.$$

We assume an enumeration $\xi_1, \ldots, \xi_n$ of the dependencies under consideration. If $\xi_p = \alpha(\bar{x}, \bar{y}) \to \exists \bar{z}\ \beta(\bar{x}, \bar{z})$, a dependency $\alpha(v_i(\bar{x}), v_j(\bar{y})) \to \beta(v_i(\bar{x}), v_k(\bar{z}))$ in $\mathit{ground}(\xi_p)$ can uniquely be denoted $\xi_{p,i,j,k}$. For a set $\Sigma = \{\xi_1, \xi_2, \ldots, \xi_n\}$ of tuple generating dependencies we have $\mathit{ground}(\Sigma) = \mathit{ground}(\xi_1) \cup \cdots \cup \mathit{ground}(\xi_n)$. The set $\mathit{ground}(\Sigma)$ thus consists of all ground instances of the dependencies in Σ. The elements of $\mathit{ground}(\Sigma)$ are called *ground dependencies*, or *ground TGD's.* A set G of ground TGD's is said to be *finitely generated*, if $G \subseteq \mathit{ground}(\Sigma)$, for some finite set Σ of TGD's.

Let I be an instance, G a set of finitely generated ground dependencies, and ground dependency $\xi_{p,i,j,k} = \alpha \to \beta$ in G. Then we say $\alpha \to \beta$ *applies to* (I, G) if $\alpha \subseteq I$. Furthermore, we say that $\xi_{p,i,j,k} = \alpha \to \beta$ *derives* (I', G') from (I, G), if

$$(I', G') = \big(I \cup \{\beta\}, G \setminus \{\xi_{p,i,j,\ell} : \ell \in \omega\}\big).$$

This relation is denoted $(I, G) \Rightarrow_{\alpha\to\beta} (I', G')$. The intuition behind deleting the groundings $\xi_{p,i,j,\ell}$ from G is that since dependency ξ_p was fired using valuations v_i and v_j for the body, and with $v_k(\bar{z})$ as a "witness" for the existential variables in the head, we do not want to fire this dependency again, using v_i and v_j.

Consider now a sequence

$$(I_0, G_0), (I_1, G_1), \ldots, (I_n, G_n), \ldots \tag{6.19}$$

where $(I_0, G_0) = (I, G)$ and $(I_i, G_i) \Rightarrow_{\alpha\to\beta} (I_{i+1}, G_{i+1})$, for some ground dependency $\alpha \to \beta \in G_i$ that applies to I_i, or where $(I_i, G_i) = (I_{i+1}, G_{i+1})$, if no $\alpha \to \beta \in G_i$ applies to I_i. We call such a sequence a *chase sequence originating from* (I, G).

A chase sequence originating from (I, G) is said to be *fair* if for all ground dependencies $\xi_{p,i,j,k} \in G$, for which there exists an $n \in \omega$ such that $\xi_{p,i,j,k}$ applies to (I_n, G_n). There also exists $m, \ell \in \omega$, $m \geq n$ such that $(I_m, G_m) \Rightarrow_{\alpha \to \beta} (I_{m+1}, G_{m+1})$, where $\alpha \to \beta = \xi_{p,i,j,\ell}$. Fairness means that all applicable dependencies are eventually fired for some instantiation of the existential variables in the head. From now on, unless stated otherwise, we consider all chase sequences to be fair.

It is obvious that $I_i \subseteq I_{i+1}$ and that $G_i \supseteq G_{i+1}$, for each $i \in \omega$. We can therefore define the *limit of a chase sequence* as

$$(\bigcup_{i \in \omega} I_i, \bigcap_{i \in \omega} G_i). \tag{6.20}$$

The notation $\mathcal{C}(I, G)$ will stand for the set of the limits of *all* chase sequences originating from (I, G).

We are now ready to define $\Sigma(I)$, the set of all *constructible models* of I and Σ, as

$$\Sigma(I) = \{J \; : \; (J, G) \in \mathcal{C}\big(I, ground(\Sigma)\big), G \subseteq ground(\Sigma)\}. \tag{6.21}$$

It is clear that the definition of $\Sigma(I)$ does not depend on the particular enumeration of the valuations of the variables in the dependencies in Σ. Note also that $\Sigma(I)$ may contain finite as well as infinite ground instances.

The following lemma derives directly from the definition of $\Sigma(I)$:

Lemma 7 *$\Sigma(I) \subseteq Sat(\Sigma)$, where $Sat(\Sigma)$ denotes the set of all instances I such that $I \models \Sigma$.*

The definition of function $\Sigma(I)$ is extended to a set of ground instances $\mathcal{I}$ point-wise as follows:

$$\Sigma(\mathcal{I}) = \bigcup_{I \in \mathcal{I}} \Sigma(I). \tag{6.22}$$

The data exchange solution space under constructible models semantics can be formalized as follows:

Definition 35 *Let $M = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ be a data exchange mapping specified by a set of TGD's. Let $I \in Inst(\mathbf{S})$ be a source instance. The solution space for M and I under constructible models semantics is the following set of ground instances, where $\Sigma = \Sigma_{st} \cup \Sigma_t$:*

$$Sol_M^{\mathtt{CM}}(I) = \{J \in Inst(\mathbf{T}) | \; \exists K \in \Sigma(I) \text{ and } J = K|_{\mathbf{T}}\} \tag{6.23}$$

Let us consider the following example that captures all the previous notions.

Example 37 *Let $\Sigma = \{\xi\}$, where ξ is the sample TGD 6.8 used to present the PWS semantic.*

$$Manages(emp, mgr) \rightarrow \exists eid \, ManageIds(eid, mgr) \tag{6.24}$$

The grounding $ground(\xi)$ is the following infinite set of ground dependencies:

$$\begin{aligned}
ground(\xi) = \{Manages(john, ray) &\rightarrow ManageIds(101, ray); \\
Manages(john, ray) &\rightarrow ManageIds(102, ray); \\
Manages(john, ray) &\rightarrow ManageIds(103, ray); \quad \ldots \\
Manages(mike, ray) &\rightarrow ManageIds(101, ray); \\
Manages(mike, ray) &\rightarrow ManageIds(102, ray); \\
Manages(mike, ray) &\rightarrow ManageIds(103, ray); \quad \ldots \\
Manages(ann, ray) &\rightarrow ManageIds(101, ray); \\
Manages(ann, ray) &\rightarrow ManageIds(102, ray); \\
Manages(ann, ray) &\rightarrow ManageIds(103, ray); \quad \ldots \\
&\ldots \\
Manages(ray, john) &\rightarrow ManageIds(101, john); \quad \ldots \\
&\ldots \qquad \}
\end{aligned} \tag{6.25}$$

As $\Sigma = \{\xi\}$*, it follows that* $G = ground(\xi)$*. Figure 6.1 represents the chase tree corresponding to* $ground(\Sigma)$ *and instance:*

$$I = \{Manages(john, ray), Manages(mike, ray)\}. \tag{6.26}$$

In figure 6.1 it can also be observed that, after applying to (I, G) *the ground dependency* $Manages(john, ray) \rightarrow ManageIds(101, ray)$*, we change the existing set of groundings to* $G_1 = G \setminus \{Manages(john, ray) \rightarrow ManageIds(*, *)\}$*. That is* G_1 *will not contain any ground dependency with the body* $Manages(john, ray)$*. Thus the tuple* $Manages(john, ray)$ *from* I *will not contribute to any other tuple in the limit instance. This ensures that each tuple from the limit is "justified" by different tuples in the instance or by a different dependency. We may note that in this example, because the dependency is a source-to-target TGD, the depth of the tree is 2. Thus there are finite limits for each branch of the chase tree.*

The solution space under constructible models semantics for the data exchange settings specified in the previous example is given by the following infinite set of instances:

J_1	J_2	J_3	
$ManageIds(101, \text{ray})$	$ManageIds(101, \text{ray})$	$ManageIds(102, \text{ray})$	$\ldots$
	$ManageIds(102, \text{ray})$		

Let us now reconsider the query: *Do all managers have at most two employees each?* for which the certain answer under PWS semantics was **false**. Under constructible models semantics the certain answer is, the expected one, **true**, as each branch in chase tree from Figure 6.1 has depth 2 with the target instance containing at most two tuples.

In the remaining part of this section we will review each of the examples used for the different closed words semantics presented in Section 6.1. First let us start with the mapping M specified by the TGD 6.5 presented for both CWA and GCWA semantics.

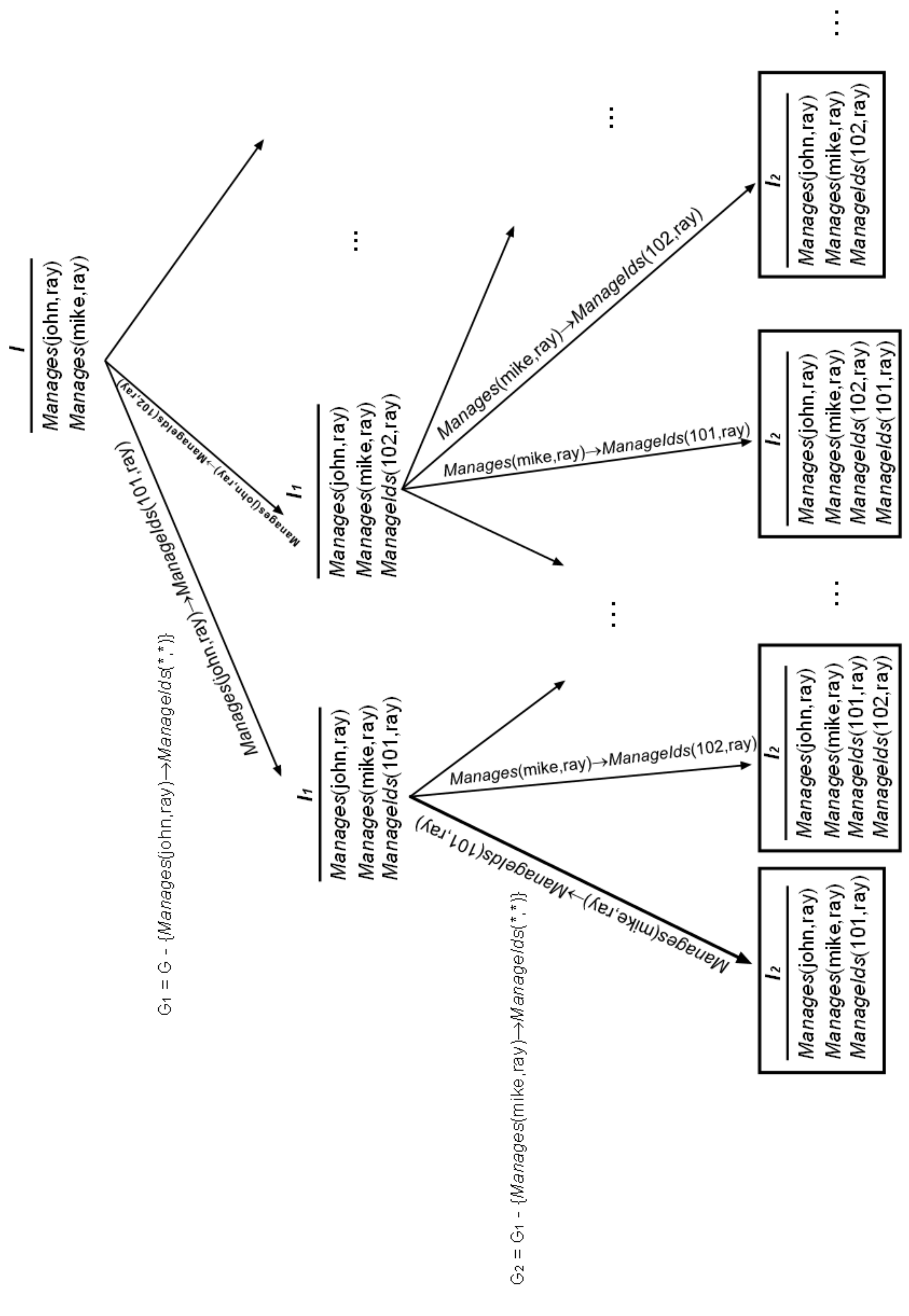

Figure 6.1: Chase tree

$$Employee(\text{emp}) \rightarrow \exists \text{mgr}\ Manages(\text{emp}, \text{mgr}), Manager(\text{mgr}) \tag{6.27}$$

Consider also the same source instance $I = Employee(\text{john})$. In this case the solution space under constructible models semantics is the following infinite set of ground instances:

K_1	K_2	K_3	
$Manages(\text{john}, \text{john})$	$Manages(\text{john}, \text{ray})$	$Manages(\text{john}, \text{mike})$	$\dots$
$Manager(\text{john})$	$Manager(\text{ray})$	$Manager(\text{mike})$	

Consider now the same query (6.6) presented for the CWA and GCWA semantics:

$$q \leftarrow \forall \text{mgr}\ Manager(\text{mgr}) \rightarrow \exists \text{emp}\ Manages(\text{emp}, \text{mgr}) \tag{6.28}$$

Under both semantics CWA and GCWA, the certain answer to this query was **false**. Under our new semantic the certain answer to this query is, again the expected one, **true**.

Under CWA-solution semantics we considered the mapping M specified by the following source-to-target and target TGD's :

$$\begin{aligned} \Sigma_{st}: \quad & Scholar(s) \rightarrow Student(s) & (6.29)\\ & Scholar(s) \rightarrow \exists m_1, m_2\ Major(s, m_1, m_2) & (6.30)\\ \Sigma_t: \quad & Major(s, m, m) \rightarrow StdOneMajor(s) & (6.31) \end{aligned}$$

The solution space under constructible models semantics for M and source instance $I = \{Scholar(\text{john})\}$ is the following infinite set of ground instances:

U_1	U_2	U_3	
$Student(\text{john})$	$Student(\text{john})$	$Student(\text{john})$	
$Major(\text{john}, \text{c1}, \text{c2})$	$Major(\text{john}, \text{c1}, \text{c3})$	$Major(\text{john}, \text{c1}, \text{c1})$	$\dots$
		$StdOneMajor(\text{john})$	

Under this closed world semantics the certain answer to the safe conjunctive query with one negation: $q(x) \rightarrow Student(x) \wedge \neg StdOneMajor(x)$ is, the expected one, “john“.

Finally, let us consider the TGD’s specifying the mapping M presented for the latest GCWA* semantics:

$$Manages(\text{emp}, \text{mgr}) \rightarrow \exists \text{eid}\ ManageIds(\text{eid}, \text{mgr}) \tag{6.32}$$
$$SelfManager(\text{mgr}, \text{mgrid}) \rightarrow ManageIds(\text{mgrid}, \text{mgr}) \tag{6.33}$$

Also consider the same source instance I used in presenting the GCWA* semantics:

$$I = \{Manages(\text{john}, \text{ray}), Manages(\text{mike}, \text{ray}), SelfManager(\text{ray}, 411)\}. \tag{6.34}$$

We saw that under GCWA* semantics the query: *Do all the managers manage exactly one employee each?* for mapping M and source instance I has the certain answer **true**. Under constructible models semantics the solutions space considered is the following infinite set of ground instances:

V_1	V_2	V_3	
$ManageIds(101, \text{ray})$	$ManageIds(101, \text{ray})$	$ManageIds(411, \text{ray})$	...
$ManageIds(102, \text{ray})$	$ManageIds(411, \text{ray})$		
$ManageIds(411, \text{ray})$			

For the given query the certain answer under the constructible models semantics is **false**, as expected, since there are instances part of the solution space for which “ray“ manages more than one employee.

In this section we presented the constructible models semantics which, as we saw, is a good candidate for the closed world semantics in data exchange. After presenting this new semantics, we need to find a tool to materialize, in some form, the solution space associated with it. In the following sections we show that there exists a chase based procedure able to return for a mapping and a source instance a conditional table such that its representation is exactly the solution space under constructible models semantics.

6.3 Conditional Tables and Unification Process

Before presenting the chase procedure able to create the solution space under constructible models semantics, we introduce now some new notions and review some existing ones needed to introduce the chase procedure based on conditional tables. For this, first we present the conditional tables in a way that clearly shows that they are generalizations of general instances. We then introduce the concept of unification of two instances which is needed as a technical tool for generalizing the standard-chase algorithm to our conditional chase, introduced in Section 6.4. For the ease of notation, in this section, we will refer by instance to a ground instance. This, we will refer as *tableau* to the instance that may contain null values.

6.3.1 Conditional Tables

A conditional table (c-table) [49] is a pair (T, φ), where T is a tableau, and φ is a mapping that associates a *local condition* $\varphi(t)$ with each tuple $t \in T$. A (local) condition is a Boolean formula built up from atoms of the form $x = y$, $x \neq y$, $x = a$, $x \neq a$, $a = b$ and $a \neq b$ for $x, y \in \Delta_{\mathsf{N}}$ and $a, b \in \Delta_{\mathsf{C}}$. An atomic equality of the form $a = a$ for $a \in \Delta_{\mathsf{C}}$ represents the logical constant **true** and for two distinct constants a and b the equality $a = b$ represents **false**. We denote the logical implication between conditions φ and ϕ by $\varphi \vDash \phi$ and their logical equivalence by $\varphi \equiv \phi$.

If $\varphi(t) \equiv$ **true**, for all $t \in T$, the conditional table (T, φ) is sometimes denoted simply with T.

A conditional table (T, φ) *represents* a set of possible worlds (ground instances).

Let v be a valuation, then:

$$v(T,\varphi) = \{v(t) \mid t \in T, \text{ and } v\big(\varphi(t)\big) \equiv \mathbf{true}\}. \tag{6.35}$$

The set of possible worlds represented by (T,φ) is:

$$rep_{\mathbf{C}}(T,\varphi) = \{v(T,\varphi) \mid v \text{ is a valuation}\}. \tag{6.36}$$

Note that we are making the *closed world assumption* [68], meaning that no ground tuple can be true in a possible world, unless it is an instantions of a tuple in the c-table. Furthermore, if (T,φ) and (U,ϕ) are c-tables such that $rep_{\mathbf{C}}(T,\varphi) = rep_{\mathbf{C}}(U,\phi)$, we say (T,φ) and (U,ϕ) are *equivalent*. We denote this by $(T,\varphi) \equiv (U,\phi)$.

Example 38 *Consider the conditional table (T,φ) with the tabular representation as shown below:*

t	$\varphi(t)$
$R(a,X)$	**true**
$R(b,c)$	**true**
$R(a,c)$	$X = b$

For the same conditional table we will use the following in-line representation:

$$(T,\varphi) = \{(R(a,X),\mathbf{true}),(R(b,c),\mathbf{true}),(R(a,c),X = b)\}$$

The set of possible worlds is given by the set of valuations for (T,φ). Bellow are presented the tabular representations of some of possible instances for (T,φ), where $I_i = v_i(T,\varphi)$, for $i \in \{1,2,3\}$, $v_1(X) = a$, $v_2(X) = b$, and $v_3(X) = c$.

I_1
$R(a,a)$
$R(b,c)$

I_2
$R(a,b)$
$R(b,c)$
$R(c,d)$

I_3
$R(a,c)$
$R(b,c)$

When it comes to dependencies, a conditional table (T, φ) is said to *satisfy* a set Σ of dependencies, if all possible worlds it represents satisfy Σ, that is, if we have $rep_{\mathbf{C}}(T, \varphi) \subseteq Sat(\Sigma)$.

Example 39 *Consider the conditional table (T, φ) from example 38 and the dependencies $\Sigma_1 = \{R(x,y), R(y,z) \rightarrow R(x,z)\}$ and $\Sigma_2 = \{R(x,y) \rightarrow \exists z\, R(y,z)\}$. Clearly we have $rep_{\mathbf{C}}(T, \varphi) \subseteq Sat(\Sigma_1)$. On the other hand, the same conditional table (T, φ) does not satisfy Σ_2 as, for example, in I_1 we have tuple $R(b,c)$ but there is no tuple in I_1 that has c as value in the first column in relation R, thus $I_1 \nvDash \Sigma_2$.*

The standard chase on instances is essentially a fixed point computation, where the underlying partial order is that of set inclusion. The conditional chase will also be a fixed point computation, so we will need a suitable partial order on conditional tables. Let (T, φ) and (U, ψ) be c-tables, by $(T, \varphi) \sqsubseteq (U, \phi)$ we mean that $T \subseteq U$ and that, for all valuations v, it is that $v(T, \varphi) \subseteq v(U, \phi)$.

If both $(T, \varphi) \sqsubseteq (U, \phi)$ and $(T, \varphi) \sqsubseteq (U, \phi)$, we write $(T, \varphi) \cong (U, \phi)$ and say that the c-tables are *congruent*. Obviously $(T, \varphi) \cong (U, \phi)$ if and only if $T = U$ and for all valuations v, $v(T, \varphi) = v(U, \phi)$. Consider the following binary union operation between conditional tables: $(T, \varphi) \sqcup (U, \phi) =_{\text{def.}} (T \cup U, \varphi{\sqcup}\phi)$, where

$$\varphi{\sqcup}\phi\,(t) = \begin{cases} \varphi(t) \vee \phi(t) & \text{if } t \in T \cap U, \\ \varphi(t) & \text{if } t \in T \setminus U, \\ \phi(t) & \text{if } t \in U \setminus T. \end{cases} \tag{6.37}$$

Thus $\sqcup$ is a partial order on equivalence classes under $\cong$ of c-tables. Recall that two conditional tables are *equivalent* if they represent the same set of possible worlds. The following lemma states that congruence is stronger than equivalence.

Lemma 8 *Let (T, φ) and (U, ϕ) be two conditional tables. If $(T, \varphi) \cong (U, \phi)$, then $(T, \varphi) \equiv (U, \phi)$. There are c-tables (T, φ) and (U, ϕ), such that $(T, \varphi) \equiv (U, \phi)$ and $(T, \varphi) \ncong (U, \phi)$.*

Proof: Let us suppose that $(T,\varphi) \cong (U,\phi)$. From the definition of $\cong$ it follows that $T = U$ and for all valuations v and tuples $t \in T$ if $\phi(t) \equiv \mathbf{true}$, then $v(t) \in v(T,\varphi)$ and inverse. Let v be a valuation, we will show that $v(U,\phi) = v(T,\varphi)$. Let $t \in v(U,\phi)$. It follows that there exists a tuple $t_U \in U$ such that $v(t_U) = t$ and $v(\phi(t_U)) = \mathbf{true}$. From this it follows that $t = v(t_U) \in v(T,\varphi)$, thus $v(U,\phi) \subseteq v(T,\varphi)$. Similarly, it can be proved that $v(T,\varphi) \subseteq v(U,\phi)$. For the second part of this lemma, consider conditional tables $(T,\varphi) = \{(R(a,b),\mathbf{true})\}$ and $(U,\phi) = \{(R(a,X), X = b)\}$. It is easy to see that $(T,\varphi) \equiv (U,\phi)$ but $(T,\varphi) \ncong (U,\phi)$■

6.3.2 Unification of Tableaux

In order to extend the standard-chase procedure to conditional tables, in this section we introduce first the notions of unifiers and most general unifiers for pairs of tableaux. For this, let us first extend the notion of retraction (as defined in Section 2.1) to the notion of D-retraction, for a set of constants D.

Definition 36 *Let T be a tableau and let D be a finite set of constants. A D-*retraction *for T is a mapping $h : dom(T) \to dom(T) \cup D$, identity on $\Delta_{\mathsf{C}} \cup dom(h(T))$.*

Note that if $D = \varnothing$, then a D-retraction for T is a retraction for T. From now on we will refer to a $\varnothing$-retraction simply as retraction.

Example 40 *Let $T = \{R(a,X), R(a,Y), R(a,b)\}$ and $D = \{c,d\}$, then the mappings $h_1 = \{X/c\}$, $h_2 = \{X/a, Y/a\}$ are D-retractions for T. On the other hand, the mapping $h_3 = \{X/Y, Y/a\}$ is not a D-retraction because h_3 maps X to Y and it is not identity on Y.*

We shall also need the concept of an h-core of an instance, for a retraction h.

Definition 37 *Let T be a tableau, D a set of constants, and h a D-retraction for T. Then we say that T is an h-* core*, if there is no $T' \subsetneq T$, such that $h(T') = h(T)$.*

It is easy to see that a tableau T is a core if and only if T is an h-core for all retractions h on T.

Example 41 *Consider $T = \{R(a,X), R(a,b)\}$ and retraction $h = \{X/b\}$. It is easy to see that T is not an h-core because for $T' = \{R(a,b)\}$ we have $T' \subsetneq T$ and $h(T') = h(T)$. On the other hand, retraction $h' = \{X/c\}$ is an h'-core. Clearly tableau T is not a core.*

Recall that for a tableau T by $\Delta_{\mathsf{C}}(dom(T))$ we denote the set of all constants in T.

Definition 38 *Let T and U be two tableaux. A* unifier *for tableaux T and U, if it exists, is a pair (θ_1, θ_2), where θ_1 is a homomorphism from the set $dom(T)$ to the set $dom(U) \cup \Delta_{\mathsf{C}}(dom(T))$ and θ_2 is a $\Delta_{\mathsf{C}}(dom(T))$-retraction for U, such that the following holds $\theta_1(T) = \theta_2(U)$.*

Note the asymmetrical role of T and U: a unifier for (T,U) is not necessarily a unifier for (U,T).

Example 42 *Consider tableaux $T = \{R(a, X_1, X_1, X_2)\}$ and $U = \{R(Y_1, b, Y_2, Y_3)\}$. The pair (θ_1, θ_2), where $\theta_1 = \{X_1/b, X_2/b\}$ and $\theta_2 = \{Y_1/a, Y_2/b, Y_3/b\}$ is a unifier for T and U as $\theta_1(T) = \theta_2(U) = \{R(a,b,b,b)\}$. It can be noticed that (θ_1, θ_2) is not the unique unifier for T and U. For example (θ'_1, θ'_2), where $\theta'_1 = \{X_1/b, X_2/Y_3\}$ and $\theta'_2 = \{Y_1/a, Y_2/b\}$, is also a unifier for T and U.*

Definition 39 *A unifier (θ_1, θ_2) for tableaux T and U is* more general *than a unifier (γ_1, γ_2), if there is a mapping f on $dom(U)$, $f \notin \mathsf{Id}$, identity on constants, such that $\gamma_1 = f \circ \theta_1$ and $\gamma_2 = f \circ \theta_2$.*

Clearly this defines a partial preorder on unifiers and a partial order on equivalence classes of isomorphic unifiers. Two unifiers (θ_1, θ_2) and (γ_1, γ_2) of (T, U) are considered isomorphic if $\theta_1(T)$ is isomorphic with $\gamma_1(T)$.

Definition 40 *A unifier* (θ_1, θ_2) *is a* most general unifier *(mgu) for tableaux T and U, if all unifiers (γ_1, γ_2) of T and U that are more general than (θ_1, θ_2) actually are isomorphic with (θ_1, θ_2). We denote by $mgu(T, U)$ the set of (representatives of the) equivalence classes of all mgu's of T and U.*

Example 43 *In Example 42 unifier (θ'_1, θ'_2) is a more general unifier than (θ_1, θ_2) and it is also an mgu. Let us consider another example over instances $T = \{R(X, Y), R(Y, Z)\}$ and $U = \{R(a, V), R(W, b)\}$. In this case, the only mgu of T and U is given by the equivalence class of unifier $(\{X/a, Y/W, Z/b\}, \{V/W\})$.*

It is well known that, if two atoms are unifiable, then there exists a most general unifier that is unique up to isomorphism. But, when unifying two tableaux, there might be several non-isomorphic mgu's. For example, let us consider the following tableaux $T = \{R(X, Y), R(Y, Z)\}$ and $U = \{R(a, V), R(b, W)\}$. Then (θ_1, θ_2), with $\theta_1 = \{X/a, Y/b, Z/W\}$ and $\theta_2 = \{V/b\}$, is an mgu for T and U. On the other hand, the unifier (γ_1, γ_2), with $\gamma_1 = \{X/b, Y/a, Z/V\}$ and $\gamma_2 = \{W/a\}$, is also an mgu for T and U. It is easy to see that (γ_1, γ_2) and (θ_1, θ_2) are not isomorphic.

The following step is to find an upper bound for the complexity of the membership problem for the set $mgu(T, U)$. The next lemma follows directly from Definition 37.

Lemma 9 *Let T and U be two tableaux and (θ_1, θ_2) a most general unifier for T and U. If U is θ_2-core, then $|U| \leq |T|$.*

Proof: Obviously, if tableau U is a θ_2-core, then $|U| = |\theta_2(U)|$, on the contrary, we would have instance $U' = \theta_2(U)$ such that $U' \subsetneq U$ and $\theta_2(U') = \theta_2(U)$ (note that this does not work if θ_2 would be an endomorphism and not a retraction see Example 40). On the other hand, we have that $\theta_1(T) = \theta_2(U)$, thus $|T| \geq |U|$∎

From this we now have:

Proposition 18 *Let $|U| \leq |T| = c$. Then the set $mgu(T, U)$ can be computed in time $O((2c)^{2c}c^2)$.*

Proof: Giving a pair of mapping (θ_1, θ_2), we can check if (θ_1, θ_2) is a unifier for T and U in $O(c)$ time. Given a unifier (θ_1, θ_2), we can check if (θ_1, θ_2) is an mgu in time $O(c^2)$ using the following algorithm:

```
1  for $X/c \in \theta_2$
2       do
3          $\theta_2' \leftarrow \theta_2 \setminus \{X/c\}$
4          if exists $\theta_1'$ such that $(\theta_1', \theta_2')$ is a unifier for $T$ and $U$
5             then return false
6             else  return true
```

The testing at Step 4 can be done in time $O(c)$, thus the algorithm runs in time $O(c^2)$. The number of mappings from U to $dom(U) \cup const(T)$ is bounded by $(2c)^c$; similarly the number of mappings from T to $dom(U) \cup const(T)$ is bounded by $(2c)^c$. Thus we need to check with the previous algorithm $(2c)^{2c}$ unifiers each such checking running in time $O(c^2)$. That is, the time needed to find the set $mgu(T, U)$ is $O((2c)^{2c}c^2)$∎

6.3.3 The Core of Conditional Tables

Contrary to the standard-chase step, the conditional-chase one is more sophisticated. For this reason, before presenting the conditional-chase algorithm, we need to introduce a new function on conditional tables called the conditional core (or simply core). Given a relation name R, we denote by $R(\bar{X})$ a tuple over R, where $\bar{X} \in (\Delta_{\mathsf{N}} \cup \Delta_{\mathsf{C}})^{arity(R)}$.

Definition 41 *Let $R(\bar{X})$ and $R(\bar{Y})$ be two tuples over R. We say that θ is a* tuple unifier *(or simply unifier) for $R(\bar{X})$ and $R(\bar{Y})$, if $\theta(R(\bar{X})) = \theta(R(\bar{Y}))$ and θ is a mapping from set $dom(\{R(\bar{X})\}) \cup dom(\{R(\bar{Y})\})$, identity on Δ_{C} with values in the set $(\Delta_{\mathsf{N}} \smallsetminus \Delta_{\mathsf{N}}(\{R(\bar{X}), R(\bar{Y})\})) \cup \Delta_{\mathsf{C}}$.*

Note that it may be that there is no unifier between two tuples. Also, it can be observed that the unifier is symmetric. That is, if θ is unifier for $R(\bar{X})$ and $R(\bar{Y})$, then θ is also a unifier for $R(\bar{Y})$ and $R(\bar{X})$. We say that two tuples are *unifiable* if there exists a unifier for those tuples. Also, note that the unifier assigns new null values, thus the unifier does not reuse any nulls from the unified tuples.

Example 44 *Consider tuples $t_1 : R(a, X_1, c)$ and $t_2 : R(X_2, X_3, c)$. The mappings $\theta_1 = \{X_1/d, X_2/a, X_3/d\}$, $\theta_2 = \{X_1/X, X_2/a, X_3/X\}$ are unifiers for t_1 and t_2, where X is a new null value. On the other hand, the mapping $\theta_3 = \{X_1/X_3, X_2/a\}$ is not a unifier even if $\theta_3(t_1) = \theta_2(t_2)$, because it maps null X_1 to existing null X_3.*

Similarly to the unifiers between instances, we have the notion of more general unifier for tuples.

Definition 42 *Let also θ_1, θ_2 be two unifiers for tuples t_1 and t_2. We say that θ_1 is a more general unifier than θ_2 if there exists a non-isomorphic mapping $f \notin \mathsf{Id}$, that is identity on constants, such that $\theta_2 = f \circ \theta_1$.*

Returning to Example 44, θ_2 is a more general unifier than θ_1 for tuples t_1 and t_2.

Definition 43 *A unifier θ is said to be a most general unifier for tuples t_1 and t_2 if there does not exist unifier θ' such that θ' is more general than θ.*

Clearly any two most general unifiers for two tuples t_1 and t_2 are isomorphic. By $mgu(t_1, t_2)$ we denote a representative unifier from the equivalence class of most general unifiers for t_1 and t_2. For a unifier θ we shall use the abbreviation:

$$\text{⩕}(\theta) \;=_{\mathsf{def}}\; \Big(\bigwedge_{X,Y\in\Delta_{\mathsf{N}},\, \theta(X)=\theta(Y)} X = Y\Big) \wedge \Big(\bigwedge_{X\in\Delta_{\mathsf{N}},\, a\in\Delta_{\mathsf{C}},\, \theta(X)=a} X = a\Big) \tag{6.38}$$

Example 45 *Returning to the unifiers mentioned in Example 44, $\text{⩕}(\theta_1)$ represents the formula: $X_1 = d \wedge X_2 = a \wedge X_3 = d$. and $\text{⩕}(\theta_2)$ represents: $X_1 = X_3 \wedge X_2 = a$.*

Definition 44 *A conditional table (T, φ) is said to be a conditional core if for any valuation v and for any two distinct tuples $t_1, t_2 \in T$ it cannot be that $v(t_1) = v(t_2)$ and $v(\varphi(t_1)) \equiv v(\varphi(t_2)) =$ **true**.*

Example 46 *Consider the following conditional table:*

(T, φ)

t	$\varphi(t)$
$R(a, X_1, c)$	$X_5 = a$
$R(a, b, X_2)$	$X_6 = b$
$R(X_3, X_4, c)$	$X_7 = c$
$R(a, b, c)$	$X_8 = d$

*For valuation $v = \{X_1/b, X_2/c, X_3/a, X_4/b, X_5/a, X_6/b, X_7/d, X_8/d\}$ we have for all tuples $t \in T$, $v(t) = R(a, b, c)$ and $v(\varphi(t)) \equiv$ **true**. Thus (T, φ) is not a conditional core.*

Lemma 10 *Given a conditional table (T, φ) such that $\varphi(t) \equiv$ **true**, for all $t \in T$, then testing if (T, φ) is a conditional core can be done in time $O(n^2)$*

Proof: This testing comes up to checking if there are two unifiable tuples in T ■

The previous lemma considered the local conditions tautologies. However, for arbitrary boolean formula over equalities and inequalities, the problem becomes coNP-complete.

Even if the complexity of checking if a conditional table is a conditional core is coNP-complete in general, there exists a polynomial time algorithm that computes a core for a conditional table, assuming that there exists a total order between the tuples in the conditional table. Considering this, we denote the order relation between tuples by the symbol "$\lessdot$". In our conditional-chase algorithm, the order considered will be the one in which the tuples are generated at each step. This is why in the following algorithm we will traverse the tuples in descending order.

```
CORE-COMP(($T, \varphi$)
1   $(U, \phi) \leftarrow (T, \varphi)$, such that $U = \{t_1, \ldots, t_{|T|}\}$ and $t_j \lessdot t_i$, for $1 \leq i < j \leq |T|$.
2   for $i \leftarrow 1$ to $|T| - 1$
3       do
4           for $j \leftarrow i + 1$ to $|T|$
5               do
6                   if $t_i$ is unifiable with $t_j$
7                      then
8                           $\theta \leftarrow mgu(t_i, t_j)$
9                           $\phi(t_i) \leftarrow \phi(t_i) \wedge \neg \bigwedge(\theta)$
10                          $\phi(t_j) \leftarrow \phi(t_j) \vee (\phi(t_i) \wedge \bigwedge(\theta))$
11  return $(U, \phi)$
```

Figure 6.2: The Conditional Table Core Computation

Figure 6.2 presents the *CORE-COMP*(T, φ) algorithm which computes the core of a given conditional table (T, φ) with the tuples ordered by $\lessdot$. It is easy to observe that this algorithm runs in time $O(n^2)$, where $n = |T|$. As the algorithm is deterministic and terminates for all input conditional tables by *CORE-COMP*(T, φ), we will also denote the conditional table returned by the algorithm. The following theorem states the main result of this section:

Theorem 54 *Let (T, φ) be a conditional table with a total order $\lessdot$ on the tuples in T,* *then* $CORE\text{-}COMP(T,\varphi)$ *is a conditional core and* $CORE\text{-}COMP(T,\varphi) \cong (T,\varphi)$.

Proof: First we will prove that $CORE\text{-}COMP(T,\varphi)$ is a conditional core. Let us denote $(U,\phi) = CORE\text{-}COMP(T,\varphi)$. Towards a contradiction let us assume that (U,ϕ) is not a conditional core. Thus, there exist a valuation v and the tuples $t_1, t_2 \in U$, such that $v(\phi(t_1)) \equiv v(\phi(t_2)) \equiv \textbf{true}$ and $v(t_1) = v(t_2)$. Wlog. We may assume that $t_2 \lessdot t_1$, if not we interchange the notation of the two tuples. Because $v(t_1) = v(t_2)$, it follows that v is a unifier for t_1 and t_2, thus there exists $\theta = mgu(t_1,t_2)$. On the other hand, because $t_2 \lessdot t_1$, it follows, from the description of the algorithm, that $\phi(t_1)$ is a formula of the form $\mu \wedge \neg ⩕ (\theta)$. Where μ is an expression, because t_1 is unified with t_2 only after all the unification between t_1 and t (line 6 of the algorithm) took place, for all t such that $t_1 \lessdot t$. On the other hand, because θ is a more general unifier than v, it follows that $⩕(v) \vDash ⩕(\theta)$, $v(\phi(t_1)) = v(\mu \wedge \neg ⩕ (\theta))$, thus $v(\phi(t_1)) = v(\mu) \wedge \neg v(⩕(\theta))$. But we have that $v(⩕(\theta)) \equiv \textbf{true}$ following $v(\phi(t_1)) = \textbf{false}$ contradicting with the initial assumption that $v(\phi(t_1)) \equiv \textbf{true}$.

Before continuing the proof, let us introduce a few notations, where t is a tuple and v a valuation:

- Let $parent(t)$ be the ordered sequence $(t_1,\dots,t_n)$ such that $t_j \lessdot t_i$, for all integers $1 \le i < j \le n$, $t_i \lessdot t$ (note that algorithm uses the descending order) and the tuples t_i and t are unifiable, for all $i \in [n]$. We denote by $parent_v(t)$ the subsequence of sequence $parent(t)$ such that, for any tuple $t' \in parent_v(t)$, we have $v(t') = v(t)$.
- Let $child(t)$ be the ordered sequence $(t_1,\dots,t_n)$ such that $t_j \lessdot t_i$, for $1 \le i < j \le n$, $t \lessdot t_i$ and the tuples t and t_i are unifiable, for $i \in [n]$. We then denote by $child_v(t)$ the subsequence of $child(t)$ such that, for any tuple $t' \in child_v(t)$, we then have $v(t') = v(t)$.
- It is clear that for any $t \in U$, $\phi(t)$ is a formula of the form:
$$\phi(t) = ((\varphi(t) \vee \mu_1 \vee \ldots \vee \mu_n) \wedge \gamma_1 \wedge \ldots \wedge \gamma_m) \tag{6.39}$$

  where $n = |parent(t)|$ and $m = |child(t)|$. Each disjunct μ_i, $i \in [n]$ was obtained by unifying tuples $t_i \in parent(t)$ and t on line 10 of the algorithm. And each conjunct γ_j, $j \in [m]$ was obtained by unifying tuples t and $t_j \in child(t)$ on line 9 of the algorithm.

- For each $t \in U$, we denote by $\phi^*(t)$ the conjunct: $(\varphi(t) \vee \mu_1 \vee \ldots \vee \mu_n)$ from the formula $\phi(t)$ specified in 6.39.

It can be easily observed that $|parent_v(t)| > |parent_v(t')|$, for any $t' \in parent_v(t)$. Also $|child_v(t)| > |child_v(t')|$ for any $t' \in child_v(t)$. Note that $|parent(t)| > |parent(t')|$ for any $t' \in parent(t)$ it does not always hold (the same for *child* sequence).

Returning to our proof, we need to show that $(U, \phi) \cong (T, \varphi)$. Note that the algorithm *CORE-COMP* does not add or eliminate tuples, thus $U = T$. Let us first show that $(U, \phi) \sqsubseteq (T, \varphi)$. For this let v be a valuation and $t \in U$. We need to prove that if $v(\phi(t)) \equiv \textbf{true}$ then $v(t) \in v(T, \varphi)$.

First we will prove that, by induction on the size of the set $parent_v(t) \cup \{t\}$, if $v(\phi^*(t)) \equiv \textbf{true}$, then there exists $t' \in (parent_v(t) \cup \{t\})$ such that $v(\varphi(t')) \equiv \textbf{true}$.

For the base case, let us suppose $|parent_v(t)| = 0$ and $v(\phi^*(t)) \equiv \textbf{true}$, where $\phi(t)$ is of the form 6.39. Because $|parent_v(t)| = 0$, it follows that there is no $t' \in parent(t)$ such that $v(t') = v(t)$. On the other hand, because each disjunct μ_i, $i \in [n]$ was obtained on line 10 of the algorithm by unifying $t_i \in parent(t)$ to t using mgu θ_i, it follows that μ_i is of the form $\lambda_i \wedge ⩕(\theta_i)$. But this means that $v(⩕(\theta_i)) \equiv \textbf{false}$, because θ_i is the mgu for t_i, t and v is not a unifier for t_i, t. Following that $v(\mu_i) \equiv \textbf{false}$ for all $i \in [n]$, because $v(\phi^*(t)) \equiv \textbf{true}$, it follows that it needs to be that $v(\varphi(t)) \equiv \textbf{true}$. This proves the base case.

For the induction step, we assume that, if for a tuple t we have $|parent_v(t)| = k$ and $v(\phi^*(t)) \equiv \textbf{true}$, then there exists tuple $t' \in (parent_v(t) \cup \{t\})$ such that $v(\varphi(t')) \equiv \textbf{true}$. Let us now consider $|parent_v(t)| = k + 1$ and $v(\phi^*(t)) \equiv \textbf{true}$. If $v(\varphi(t)) \equiv \textbf{true}$, the induction is proved. Let us suppose that $v(\varphi(t)) \equiv \textbf{false}$, thus there exists $i \in [n]$ such that $v(\mu_i) \equiv \textbf{true}$ (see formula 6.39). Where condition μ_i was added in step 10 of the algorithm due to tuple $t_i \in parent(t)$. From the construction of the disjunct μ_i, we have $\mu_i = \lambda_i \wedge ⩕(\theta)$, where $\lambda_i = \phi^*(t_i) \wedge \rho_1 \wedge \ldots \wedge \rho_h$. From the assumption we have $v(\mu_i) \equiv \textbf{true}$, thus $v(⩕(\theta)) \equiv \textbf{true}$ and $v(\lambda_i) \equiv \textbf{true}$ following that $v(\phi^*(t_i)) \equiv \textbf{true}$. On the other hand, from $v(⩕(\theta)) \equiv \textbf{true}$ it follows that $v(t) = v(t_i)$, thus $t_i \in parent_v(t)$. Meaning that $|parent_v(t_i)| < |parent_v(t)| = k + 1$ and having also $v(\phi^*(t_i)) \equiv \textbf{true}$. From the induction steps it follows that there exists a $t' \in (parent_v(t_i) \cup \{t_i\})$ such that $v(\varphi(t')) \equiv \textbf{true}$. This concludes the proof that if $v(\phi^*(t)) \equiv \textbf{true}$, then there exists $t' \in (parent_v(t) \cup \{t\})$ such that $v(\varphi(t')) \equiv \textbf{true}$. But from the main assumption we know that $v(\phi(t)) \equiv \textbf{true}$, thus $v(\phi^*(t)) \equiv \textbf{true}$. From the previous inductive proof it follows that there exists $t' \in parent_v(t)$ such that $v(\varphi(t')) \equiv \textbf{true}$. From the definition of $parent_v(t)$ sequence it follows that $v(t') = v(t)$, thus $v(t) \in v(T, \varphi)$. By this we showed that $(U, \phi) \sqsubseteq (T, \varphi)$.

It remains to be proved that $(T, \varphi) \sqsubseteq (U, \phi)$. Let us consider v a valuation and t a tuple in T such that $v(\varphi(t)) \equiv \textbf{true}$. We need to show that $v(t) \in v(U, \phi)$.

Similarly to the previous proof, first we will demonstrate by induction on the size of $child_v(t) \cup \{t\}$ that if $\phi^*(t) \equiv \textbf{true}$, then there exists $t' \in (child_v(t) \cup \{t\})$ such that $v(\phi(t')) \equiv \textbf{true}$. For the induction base case, let us suppose that $|child_v(t)| = 0$ and $v(\phi^*(t)) \equiv \textbf{true}$. For the ease of reading we rewrite below the general format of $\phi(t)$:

$$\phi(t) = ((\varphi(t) \vee \mu_1 \vee \ldots \vee \mu_n) \wedge \gamma_1 \wedge \ldots \wedge \gamma_m) \tag{6.40}$$

The assumption is $v(\phi^*(t)) \equiv \textbf{true}$, this means that either $v(\phi(t)) \equiv \textbf{true}$ or there exists a positive integer $j \in [m]$ such that $v(\gamma_j) \equiv \textbf{false}$. Towards a contradiction, let us suppose that there exists positive integer $i \in [m]$ such that $v(\gamma_j) \equiv \textbf{false}$. From the description of the formula 6.40 we know that each γ_j, $j \in [m]$ is generated on line 9 of the algorithm by unifying tuples t, t_j using an mgu θ_j. Thus γ_j is of the form $\neg ⩕(\theta_j)$.

Because $v(\gamma_i) \equiv$ **false**, it follows that $v(\curlywedge(\theta_j)) \equiv$ **true**, but because θ_j is an mgu for t, t_j, it follows that $v(t) = v(t_j)$, contradicting that $|child_v(t)| = 0$. Thus it must be that $v(\phi(t)) \equiv$ **true**, proving the base case.

For the induction step, let us suppose that for any tuple t if $|child_v(t)| = k$ and $v(\phi^*(t)) \equiv$ **true**, then there tuple $t' \in (child_v(t) \cup \{t\})$ such that $v(\phi(t')) \equiv$ **true**. Let us consider tuple t such that $|child_v(t)| = k + 1$ and $v(\phi^*(t)) \equiv$ **true**. If $v(\phi(t')) \equiv$ **true**, then the induction step is proved. Let us consider that we have $v(\phi(t')) \equiv$ **false**, thus from formula 6.40 it follows that there exists a smallest positive integer $j \in [m]$ such that $v(\gamma_j) \equiv$ **false** and for all $h \in [j]$ we have $v(\gamma_h) \equiv$ **true**. As shown in the induction basic case, γ_j is of the form $\neg \curlywedge (t_j)$, where θ_j is the mgu between tuples t and $t_j \in child(t)$. Because $v(\gamma_i) \equiv$ **false**, then $v(\curlywedge(\theta_j)) \equiv$ **true**. On the other hand, we have from line 10 of the algorithm that $\phi^*(t_j)$ is a disjunctive formula that contains the following disjunct $\lambda_j = (\phi^*(t) \wedge \gamma_1 \wedge \ldots \wedge \gamma_{j-1} \wedge \curlywedge(\theta_j))$. From the way j was chosen, it follows that, for all $h \in [j]$, we have that $v(\gamma_h) \equiv$ **true**. We also have, from the induction assumption, that $v(\phi^*(t)) \equiv$ **true** and we already showed that $v(\curlywedge(\theta_j)) \equiv$ **true**. Therefore, we have that $v(\lambda_i) \equiv$ **true**. Because λ_j is a disjunct in $\phi^*(t_j)$, it follows that $v(\phi^*(t_j)) \equiv$ **true**. Also we know that $v(\curlywedge(\theta_j)) \equiv$ **true**, where θ_j is the mgu for tuples t, t_j. It follows that $v(t_j) = v(t)$ meaning that $t_j \in child_v(t)$, thus $|child_v(t_j)| < |child_v(t)| = k + 1$. Adding that $v(\phi^*(t_j)) \equiv$ **true** from the inductive assumption, it follows that there exists a $t' \in (child_v(t_j) \cup \{t_j\})$ such that $v(\phi(t')) \equiv$ **true**. This concludes the inductive proof.

Now let t be a tuple from T such that $v(\varphi(t)) \equiv$ **true**. As $\phi(t)$ is a disjunctive formula containing the disjunct $\varphi(t)$, it follows that $v(\phi^*(t)) \equiv$ **true**. From the previous inductive proof, it follows that there exists a tuple $t' \in (child_v(t) \cup \{t\})$ with the property that $v(\phi(t')) \equiv$ **true**. But because $v(t') = v(t)$, it follows that $v(t) \in (U, \phi)$, proving that $(T, \varphi) \sqsubseteq (U, \phi)$. From this and the previous result, it follows that $(T, \varphi) \cong (U, \phi)$■

Example 47 *Let us now exemplify some of the algorithm steps by considering the* *same conditional table from Example 46 shown not to be a core:*

(T, φ)

t	$\varphi(t)$
$R(a, X_1, c)$	$X_5 = a$
$R(a, b, X_2)$	$X_6 = b$
$R(X_3, X_4, c)$	$X_7 = c$
$R(a, b, c)$	$X_8 = d$

Let us label the tuples as follows: $t_1 = R(a, X_1, c)$, $t_2 = R(a, b, X_2)$, $t_3 = R(X_3, X_4, c)$ *and* $t_4 = R(a, b, c)$.

Let us consider the following total order between tuples: $t_4 \lessdot t_2 \lessdot t_1 \lessdot t_3$. *The algorithm starts by setting* $(U, \phi) \leftarrow (T, \varphi)$. *Based on CORE-COMP algorithm presented in Figure 6.2, the tuples are unified in the descending order, that is:* t_3 *with* t_1, t_3 *with* t_2, t_3 *with* t_4, t_1 *with* t_2, t_1 *with* t_4 *and* t_2 *with* t_4.

The unifier $\theta_{31} = \{X_3/a, X_4/X, X_1/X\}$ *is an mgu for tuples* t_3 *and* t_1. *Based on formula 6.38 we have* $\curlywedge(\theta_{31}) = (X_4 = X_1 \wedge X_3 = a)$. *From line 9 and 10 of the CORE-COMP algorithm we change the local conditions of tuples* t_3 *and* t_1 *as follows:*

$$\begin{aligned}\phi(t_3) &= X_7 = c \wedge (X_4 \neq X_1 \vee X_3 \neq a)\\ \phi(t_1) &= X_5 = a \vee (X_7 = c \wedge X_4 = X_1 \wedge X_3 = a)\end{aligned}$$

Similarly for tuples t_3, t_2, *the mgu is* $\theta_{32} = \{X_3/a, X_4/b, X_2/c\}$, *thus changing the local conditions as follows:*

$$\begin{aligned}\phi(t_3) &= X_7 = c \wedge (X_4 \neq X_1 \vee X_3 \neq a) \wedge (X_3 \neq a \vee X_4 \neq b \vee X_2 \neq c)\\ \phi(t_2) &= X_6 = b \vee (X_7 = c \wedge (X_4 \neq X_1 \vee X_3 \neq a) \wedge (X_3 = a \wedge X_4 = b \wedge X_2 = c))\end{aligned}$$

We get for tuples t_3 *and* t_4*by by using mgu* $\theta_{34} = \{X_3/a, X_4/b\}$:

$$\begin{aligned}\phi(t_3) &= X_7 = c \wedge (X_4 \neq X_1 \vee X_3 \neq a) \wedge (X_3 \neq a \vee X_4 \neq b \vee X_2 \neq c)\\ &\qquad \wedge(X_3 \neq a \vee X_4 \neq b)\\ \phi(t_4) &= X_8 = d \vee (X_7 = c \wedge (X_4 \neq X_1 \vee X_3 \neq a) \wedge (X_3 \neq a \vee X_4 \neq b \vee X_2 \neq c)\\ &\qquad \wedge(X_3 = a \wedge X_4 = b))\end{aligned}$$

The unifier $\theta_{12} = \{X_1/b, X_2/c\}$ *is an mgu for* t_1 *and* t_2*, thus we have:*

$$\begin{aligned}\phi(t_1) &= (X_5 = a \vee (X_7 = c \wedge X_4 = X_1 \wedge X_3 = a)) \wedge (X_1 \neq b \vee X_2 \neq c)\\ \phi(t_2) &= X_6 = b \vee (X_7 = c \wedge (X_4 \neq X_1 \vee X_3 \neq a) \wedge (X_3 = a \wedge X_4 = b \wedge X_2 = c))\\ &\qquad \vee((X_5 = a \vee (X_7 = c \wedge X_4 = X_1 \wedge X_3 = a)) \wedge (X_1 = b \wedge X_2 = c))\end{aligned}$$

We have mgu $\theta_{14} = \{X_1/b\}$ *for tuples* t_1 *and* t_4*:*

$$\begin{aligned}\phi(t_1) &= (X_5 = a \vee (X_7 = c \wedge X_4 = X_1 \wedge X_3 = a)) \wedge (X_1 \neq b \vee X_2 \neq c) \wedge (X_1 \neq b)\\ \phi(t_4) &= X_8 = d \vee (X_7 = c \wedge (X_4 \neq X_1 \vee X_3 \neq a) \wedge (X_3 \neq a \vee X_4 \neq b \vee X_2 \neq c)\\ &\qquad \wedge(X_3 = a \wedge X_4 = b))\\ &\qquad \vee((X_5 = a \vee (X_7 = c \wedge X_4 = X_1 \wedge X_3 = a)) \wedge (X_1 \neq b \vee X_2 \neq c) \wedge (X_1 = b))\end{aligned}$$

Finally, the unifier $\theta_{24} = \{X_2/c\}$ *is the mgu for* t_2*,* t_4 *changing their local conditions to:*

$$\begin{aligned}\phi(t_2) &= (X_6 = b \vee (X_7 = c \wedge (X_4 \neq X_1 \vee X_3 \neq a) \wedge (X_3 = a \wedge X_4 = b \wedge X_2 = c))\\ &\qquad \vee((X_5 = a \vee (X_7 = c \wedge X_4 = X_1 \wedge X_3 = a)) \wedge (X_1 = b \wedge X_2 = c))) \wedge (X_2 \neq c)\\ \phi(t_4) &= X_8 = d \vee (X_7 = c \wedge (X_4 \neq X_1 \vee X_3 \neq a) \wedge (X_3 \neq a \vee X_4 \neq b \vee X_2 \neq c)\\ &\qquad \wedge(X_3 = a \wedge X_4 = b))\\ &\qquad \vee((X_5 = a \vee (X_7 = c \wedge X_4 = X_1 \wedge X_3 = a)) \wedge (X_1 \neq b \vee X_2 \neq c) \wedge (X_1 = b))\\ &\qquad \vee((X_6 = b \vee (X_7 = c \wedge (X_4 \neq X_1 \vee X_3 \neq a) \wedge (X_3 = a \wedge X_4 = b \wedge X_2 = c))\\ &\qquad \vee((X_5 = a \vee (X_7 = c \wedge X_4 = X_1 \wedge X_3 = a)) \wedge (X_1 = b \wedge X_2 = c))) \wedge (X_2 = c))\end{aligned}$$

Putting all together, the conditional table (U, ϕ) *returned by the* $CORE\text{-}COMP$ *algorithm has the same tuples as* (T, φ) *with the following local conditions:*

$$\begin{array}{rcl}
\phi(t_1) & = & (X_5 = a \vee (X_7 = c \wedge X_4 = X_1 \wedge X_3 = a)) \wedge (X_1 \neq b \vee X_2 \neq c) \wedge (X_1 \neq b) \\
\phi(t_2) & = & (X_6 = b \vee (X_7 = c \wedge (X_4 \neq X_1 \vee X_3 \neq a) \wedge (X_3 = a \wedge X_4 = b \wedge X_2 = c)) \\
& & \vee((X_5 = a \vee (X_7 = c \wedge X_4 = X_1 \wedge X_3 = a)) \wedge (X_1 = b \wedge X_2 = c))) \wedge (X_2 \neq c) \\
\phi(t_3) & = & X_7 = c \wedge (X_4 \neq X_1 \vee X_3 \neq a) \wedge (X_3 \neq a \vee X_4 \neq b \vee X_2 \neq c) \\
& & \wedge(X_3 \neq a \vee X_4 \neq b) \\
\phi(t_4) & = & X_8 = d \vee (X_7 = c \wedge (X_4 \neq X_1 \vee X_3 \neq a) \wedge (X_3 \neq a \vee X_4 \neq b \vee X_2 \neq c) \\
& & \wedge(X_3 = a \wedge X_4 = b)) \\
& & \vee((X_5 = a \vee (X_7 = c \wedge X_4 = X_1 \wedge X_3 = a)) \wedge (X_1 \neq b \vee X_2 \neq c) \wedge (X_1 = b)) \\
& & \vee((X_6 = b \vee (X_7 = c \wedge (X_4 \neq X_1 \vee X_3 \neq a) \wedge (X_3 = a \wedge X_4 = b \wedge X_2 = c)) \\
& & \vee((X_5 = a \vee (X_7 = c \wedge X_4 = X_1 \wedge X_3 = a)) \wedge (X_1 = b \wedge X_2 = c))) \wedge (X_2 = c))
\end{array}$$

Consider $v_1 = \{X_1/d, X_2/c, X_3/a, X_4/d, X_5/a, X_6/b, X_7/c, X_8/d\}$ *a valuation for* (U, ϕ). *In this case we have* $v_1(\phi_1) \equiv$ **true**, $v_1(\phi_2) \equiv$ **false**, $v_1(\phi_3) \equiv$ **false** *and* $v_1(\phi_4) \equiv$ **true**, *giving* $v_1(U, \phi) = \{R(a, d, c), R(a, b, c)\}$ *equal to* $v_1(T, \varphi)$. *Note that* $v_1(\varphi(t_4)) \equiv$ **false** *and* $v_1\phi(t_4)) \equiv$ **true**.

We conclude this section with another example of a conditional table that has isomorphic tuples:

Example 48 *Let* (T, φ) *be the following conditional table:*

(T, φ)

t	$\varphi(t)$
$R(a, X_1)$	**true**
$R(a, X_2)$	**true**
$R(a, X_3)$	**true**

The $CORE - COMP$ *algorithm with this input will return the conditional table* (U, ϕ) *with the following tabular specification:*

(U, ϕ)

t	$\phi(t)$
$R(a, X_1)$	$X_1 \neq X_2 \wedge X_1 \neq X_3$
$R(a, X_2)$	$X_1 = X_2 \wedge X_2 \neq X_3$
$R(a, X_3)$	$X_1 = X_3 \vee (X_1 = X_2 \wedge X_2 = X_3)$

6.4 The Conditional Chase

In this section we generalize the classical chase procedure to work on conditional tables. As we will prove, this new conditional chase will give a useful tool to compute a representation (in this case a conditional table) for all the possible consistent instances under the constructible models semantics. For this we need first to introduce the following central concept:

Definition 45 *Let Σ be a set of TGD's and T a tableau. A* conditional trigger for Σ on T *(or simply, a trigger) is a tuple $\tau = (\xi, \theta, T')$, where $\xi \in \Sigma$, and $\theta = (\theta_1, \theta_2)$ is an mgu that unifies body(ξ) with T', where $T' \subseteq T$ and T' is a θ_2-core. The set trigg$_\Sigma(T)$ contains the set of all triggers for Σ on T.*

Example 49 *Consider the following tableau $T = \{R(X_1, b), R(b, c), R(X_2, d)\}$ and let $\Sigma = \{\xi_1, \xi_2\}$, with ξ_1: $R(x, y), R(y, z) \rightarrow S(x, z)$ and ξ_2: $S(x, x) \rightarrow R(x, x)$. For this example we have body(ξ_1) $= \{R(X, Y), R(Y, Z)\}$. Thus the conditional triggers are trigg$_\Sigma(T) = \{\tau_1, \tau_2, \tau_3, , \tau_4, \tau_5\}$, where*

$$\begin{aligned}
\tau_1 &= (\xi_1, (\{X/X_1, Y/b, Z/c\}, \{\}), \{R(X_1, b), R(b, c)\}),\\
\tau_2 &= (\xi_1, (\{X/X_1, Y/b, Z/d\}, \{X_2/b\}), \{R(X_1, b), R(X_2, d)\}),\\
\tau_3 &= (\xi_1, (\{X/b, Y/c, Z/d\}, \{X_2/c\}), \{R(b, c), R(X_2, d)\}),\\
\tau_4 &= (\xi_1, (\{X/X_2, Y/d, Z/b\}, \{X_1/d\}), \{R(X_2, d), R(X_1, b)\}),\\
\tau_5 &= (\xi_1, (\{X/c, Y/c, Z/b\}, \{X_1/c\}), \{R(b, c), R(X_1, b)\}).
\end{aligned}$$

Note that ξ_2 does not generate any triggers as there are no tuples over relation name S in instance T.

Regarding the complexity of computing the set $trigg_{\Sigma}(T)$, we have the following:

Lemma 11 *The set $trigg_{\Sigma}(T)$ can be computed in polynomial time in the number of tuples in T.*

Proof: Let us consider $\Sigma = \{\xi_1, \xi_2, \dots, \xi_k\}$; also let $c \geq |body(\xi_i)|$ for all $i \in [n]$. For each dependency $\xi \in \Sigma$ there are a maximum of $\binom{|T|}{c}$ tableaux $T' \subseteq T$ that may contribute to a trigger. For each such T' and dependency ξ based on Proposition 18 we need time $O((2c)^{2c}c^2)$ to compute the set $mgu(body(\xi), T')$. We have k dependencies and, for each dependency, the number of subset tableaux of T that we need to check can be approximated by $|T|^c$. Thus, we need time $O(k|T|^c(2c)^{2c}c^2)$ to compute $trigg_{\Sigma}(T)$. As Σ is considered given, that is integers k and c are constants, it follows that the computing time is $O(|T|^c)$ in data complexity■

Because applying a trigger to a c-table will generate new null values (unless the TGD is full), we need a mechanism for controlling this generation in such a way that the new nulls are true representatives of the implicit skolemization taking place. Let $\tau = (\xi, (\theta_1, \theta_2), T')$ be a trigger, where $\xi = \alpha(\bar{x}, \bar{y}) \to \exists \bar{z}\ \beta(\bar{x}, \bar{z})$. For each z in $\bar{z}$, consider a function $Z \mapsto Z_\tau$, such that we have $Z_\tau = Z_\rho$ for all triggers ρ of the form $\rho = (\xi, (\theta_1, \gamma_2), T'')$. Clearly such a function exists (assuming that Δ_{V} and Δ_{N} can be well ordered).

Example 50 *For a better understanding of the previous notion, let us consider tableau $T = \{R(a, X_1), R(X_2, a)\}$ and dependency $\xi : R(x, x) \to T(x)$. For this configuration $trigg_{\{\xi\}}(T) = \{\tau_1, \tau_2\}$, where trigger $\tau_1 = (\xi, (\{X/a\}, \{X_1/a\}), \{R(a, X_1)\})$ and trigger $\tau_2 = (\xi, (\{X/a\}, \{X_2/a\}), \{R(X_2, a)\})$. In this case both triggers are defined on the same dependency and have the same first mapping as unifier. Therefore we have $X_{\tau_1} = X_{\tau_2}$.*

Before introducing the conditional-chase micro-step, let us clarify some useful notations. Let (T, φ) be a conditional table. Then we define:

$$⩕(T, \varphi) \;\; =_{\mathsf{def}} \;\; \bigwedge_{t \in T} \varphi(t), \tag{6.41}$$

Also, when θ is finite partial mapping from Δ_{N} to the set $\Delta_{\mathsf{N}} \cup \Delta_{\mathsf{C}}$, we shall use the abbreviation:

$$⩕(\theta) \;\; =_{\mathsf{def}} \;\; \bigwedge_{i} X_i = \theta(X_i), \tag{6.42}$$

where the X_i's are all the nulls in the domain of θ.

We are now ready to define the conditional-chase step on an conditional table with a total order on its tuples.

Definition 46 *Let (T, φ) be a conditional table that is a conditional core such that there exists a total order $\lessdot$ on its tuples. Let $\tau = (\xi, (\theta_1, \theta_2), T')$ be a trigger from $trigg_{\Sigma}(T)$. We denote by θ_1^{τ} the extension of θ_1 that maps each null Z from $head(\xi)$ associated with an existentially quantified variable z from ξ, to the variable Z_{τ}. We say that the conditional table (U, ϕ)* is obtained from (T, φ) by applying trigger τ, *if (U, ϕ) contains all the c-tuples from T together with their possibly modified local conditions and possibly a new c-tuple, as follows:*

<u>If</u> *T contains a tuple t syntactically equal to $\theta_1^{\tau}(head(\xi))$,*

<u>then</u> *for (U, ϕ) the local condition of t is changed to:*

$$\phi(t) =_{\mathsf{def}} \varphi(t) \vee \big(⩕\,(T', \varphi) \wedge ⩕(\theta_2)\big), \tag{6.43}$$

<u>else</u> *add the tuple t' : $\theta_1^{\tau}(head(\xi))$ with local condition:*

$$\phi(t') =_{\mathsf{def}} ⩕(T', \varphi) \wedge ⩕(\theta_2) \tag{6.44}$$

such that $t \lessdot t'$ for all $t \in T$.

If (U,ϕ) is obtained from (T,φ), in this way we write $(T,\varphi)\rightarrow_{\tau}(V,\psi)$, where (V,ψ) is the conditional table computed by CORE-COMP algorithm with input (U,ϕ). This transformation is called a conditional-chase micro-step *(or simply micro-step).*

Note that in this case the lexicographic order between tuples cannot be used as we may generate tuples that precede lexicographically the existing ones. In this case, an index based order may be used, that is to each tuple is assigned an index (numerical value) based on the total order in the initial conditional table and for each of the tuples added by the micro-step the index is increased by 1[1].

The intuition behind computing the core of the conditional table at each micro-step is given by the way constructible models semantics is defined (we want the conditional chase to mimic the constructible models semantics). To exemplify this case, consider the following example:

Example 51 *Let us consider the conditional table without any local conditions (that is a tableau) $T=\{R(a,b),R(a,c)\}$. Let $\Sigma=\{\xi_1,\xi_2\}$, where:*

$$\xi_1: \qquad R(x,y)\rightarrow\exists z\, S(x,z) \tag{6.45}$$

$$\xi_2: \qquad S(x,y)\rightarrow\exists z\, P(x,z) \tag{6.46}$$

Supposing that the core computation for the conditional table is no needed, then we have the following triggers in $trigg_{\Sigma}(T,\mathbf{true})$:

$$\begin{aligned}\tau_1 &= (\xi_1,(\{X/a,Y/b\},\{\}),\{R(a,b)\})\ \textit{and}\\ \tau_2 &= (\xi_1,(\{X/a,Y/c\},\{\}),\{R(a,c)\}).\end{aligned}$$

By applying these triggers according to Definition 46, it will create the following new tuples: $S(a,X_1)$ and $S(a,X_2)$ with tautological local condition. These new tuples

[1]Note that this is the reason why the *CORE-COMP* algorithm traverses the tuples in descending order.

will create new triggers:

$$\begin{aligned}\tau_3 &= (\xi_2, (\{X/a, Y/X_1\}, \{\}), \{S(a, X_1)\}) \text{ and}\\ \tau_4 &= (\xi_2, (\{X/a, Y/X_2\}, \{\}), \{S(a, X_2)\}).\end{aligned}$$

When applied, these new triggers will add the following two new tuples: $P(a, X_3)$ *and* $P(a, X_4)$*. It can be easily noted that there are no new triggers to be considered and, by applying any of the triggers* τ_1*,* τ_2*,* τ_3 *or* τ_4*, no new tuples will be added. Thus we get the following instance by conditional chasing* T *:*

(T', φ')

t	$\varphi'(t)$
$R(a,b)$	**true**
$R(a,c)$	**true**
$S(a,X_1)$	**true**
$S(a,X_2)$	**true**
$P(a,X_3)$	**true**
$P(a,X_4)$	**true**

Applying valuation $v = \{X_1/b, X_2/b, X_3/c, X_4/d\}$ *on the resulting conditional table, we obtain* $v(T', \varphi) \notin \Sigma(T)$*. Thus, by applying all the conditional micro-steps without adding the conditional core computation, it may generate inconsistent instances under the constructible models semantics. By considering the micro-step with the conditional core computation at each step (in this example we considered the tuples ordered lexicographically for the initial conditional table) and by applying the triggers in the order* τ_1*,* τ_2*,* τ_3 *and* τ_4*, we get the following conditional table:*

(T'', φ'')

t	$\varphi'(t)$
$R(a,b)$	**true**
$R(a,c)$	**true**
$S(a,X_1)$	$X_1 \neq X_2$
$S(a,X_2)$	**true**
$P(a,X_3)$	$X_1 \neq X_2 \wedge X_3 \neq X_4$
$P(a,X_4)$	**true**

The following lemma states, as expected, that a conditional micro-step is monotone in the $\sqsubseteq$ partial order.

Lemma 12 *If* $(T, \varphi) \rightarrow_{\tau} (U, \phi)$, *then* $(T, \varphi) \sqsubseteq (U, \phi)$.

Proof: The proof is immediate from Definition 46 and Proposition 54■

Having a table (T, φ) and a finite set Σ of TGD's, the set of triggers $trigg_{\Sigma}(T, \varphi)$ is obviously finite. Consider a sequence

$$(T, \varphi) \rightarrow_{\tau_1} (U_1, \varphi_1) \rightarrow_{\tau_2} \cdots \rightarrow_{\tau_n} (U_n, \varphi_n),$$

where $\tau_1, \tau_2, \ldots, \tau_n$ is an ordering of all the triggers from the set $trigg_{\Sigma}(T, \varphi)$. We call this a *conditional-chase micro-sequence for* (T, φ) *using* $trigg_{\Sigma}(T, \varphi)$.

Example 52 *Let us consider conditional table* (T, φ)*:*

(T, φ)

t	$\varphi(t)$
$R(a,b)$	$X_1 = c$
$R(b,a)$	$X_2 = d$
$P(a,b)$	$X_3 = e$

To this let us add dependencies $\Sigma = \{\xi_1, \xi_2\}$, *where*

$$\begin{aligned} \xi_1 : \quad & R(x,y) \rightarrow \exists z\, S(x,z) \\ \xi_2 : \quad & P(x,y) \rightarrow \exists R(x,y) \end{aligned}$$

For (T, φ) *and* Σ, *we have the following set of triggers* $trigg_{\Sigma}(T, \varphi)$*:*

$$\begin{aligned} \tau_1 &= (\xi_1, (\{X/a, Y/b\}, \{\}), \{R(a,b)\}) \\ \tau_2 &= (\xi_1, (\{X/b, Y/a\}, \{\}), \{R(b,a)\}) \\ \tau_3 &= (\xi_2, (\{X/a, Y/b\}, \{\}), \{P(b,a)\}) \end{aligned}$$

Let us consider the following sequence of micro-steps, obtained by applying the triggers τ_1 followed by τ_2 and τ_3:

$$(T, \varphi) \rightarrow_{\tau_1} (U_1, \psi_1) \rightarrow_{\tau_2} (U_2, \psi_2) \rightarrow_{\tau_3} (U_3, \psi_3).$$

where (U_1, ψ_1), (U_2, ψ_2) and (U_3, ψ_3) are the following conditional tables:

(U_1, ψ_1)

t	$\psi_1(t)$
$R(a,b)$	$X_1 = c$
$R(b,a)$	$X_2 = d$
$P(a,b)$	$X_3 = e$
$S(a,Y_1)$	$X_1 = c$

(U_2, ψ_2)

t	$\psi_2(t)$
$R(a,b)$	$X_1 = c$
$R(b,a)$	$X_2 = d$
$P(a,b)$	$X_3 = e$
$S(a,Y_1)$	$X_1 = c$
$S(b,Y_2)$	$X_2 = d$

(U_3, ψ_3)

t	$\psi_3(t)$
$R(a,b)$	$X_1 = c \vee X_3 = e$
$R(b,a)$	$X_2 = d$
$P(a,b)$	$X_3 = e$
$S(a,Y_1)$	$X_1 = c$
$S(b,Y_2)$	$X_2 = d$

Let us now consider another sequence of micro-steps obtained by applying first τ_2 followed by τ_3 and τ_1:

$$(T, \varphi) \rightarrow_{\tau_2} (V_1, \mu_1) \rightarrow_{\tau_3} (V_2, \mu_2) \rightarrow_{\tau_1} (V_3, \mu_3).$$

where (V_1, μ_1), (V_2, μ_2) and (V_3, μ_3) are the following conditional tables:

(V_1, μ_1)

t	$\mu_1(t)$
$R(a,b)$	$X_1 = c$
$R(b,a)$	$X_2 = d$
$P(a,b)$	$X_3 = e$
$S(b,Y_2)$	$X_2 = d$

(V_2, μ_2)

t	$\mu_2(t)$
$R(a,b)$	$X_1 = c \vee X_3 = e$
$R(b,a)$	$X_2 = d$
$P(a,b)$	$X_3 = e$
$S(b,Y_2)$	$X_2 = d$

(V_3, μ_3)

t	$\mu_3(t)$
$R(a,b)$	$X_1 = c \vee X_3 = e$
$R(b,a)$	$X_2 = d$
$P(a,b)$	$X_3 = e$
$S(b,Y_2)$	$X_2 = d$
$S(a,Y_1)$	$X_1 = c \vee X_3 = e$

Note that even by applying the same set of triggers on conditional table (T, φ) we have $rep_{\mathbf{C}}(U_3, \psi_3) \neq rep_{\mathbf{C}}(V_3, \mu_3)$.

As it can be noted in the previous example, the order in which the triggers are applied in a conditional micro-sequence does affect the outcome, but we shall see that in the end the order will not matter.

After a micro-sequence, it is not guaranteed that the representation of the result will satisfy the set of dependencies. It may be that additional triggers will be generated. We therefore abstract a micro-sequence into a macro-step, as follows:

Definition 47 *Let (T, φ) and (U, ϕ) be conditional tables with a total order on the set of tuples; and let Σ be a set of TGD's. We write $(T, \varphi) \Rightarrow_\Sigma (U, \phi)$, if (U, ϕ) is obtained from (T, φ) by applying all micro-steps generated from trigg$_\Sigma(T)$. The transformation from (T, φ) to (U, ϕ) is called a* conditional-chase macro-step using Σ.

For the conditional table and the set of dependencies from Example 52, we have the macro-steps $(T, \varphi) \Rightarrow_\Sigma (U_3, \psi_3)$ and $(T, \varphi) \Rightarrow_\Sigma (V_3, \mu_3)$. It is a corollary of Lemma 12 that the macro-step is monotone.

Corollary 9 *If $(T, \varphi) \Rightarrow_\Sigma (U, \phi)$, then $(T, \varphi) \sqsubseteq (U, \phi)$.*

We are now ready to introduce the concept of a conditional-chase sequence.

Definition 48 *Let (T, φ) be a c-table and Σ a set of TGD's. A sequence*

$$(T_0, \varphi_0), (T_1, \varphi_1), \ldots, (T_n, \varphi_n), \ldots$$

of c-tables, inductively constructed as

$$\begin{array}{rcl} (T_0, \varphi_0) & = & (T, \varphi), \\ (T_{i+1}, \varphi_{i+1}) & = & (U, \phi), \text{ where } (T_i, \varphi_i) \Rightarrow_\Sigma (U, \phi), \\ (T_\omega, \varphi_\omega) & = & \bigsqcup_{i \in \omega} (T_i, \varphi_i), \end{array}$$

is called a conditional-chase sequence *associated with (T, φ) and Σ. We denote $chase_\Sigma^{\mathbf{cond}}(T_0, \varphi_0) = (T_\omega, \varphi_\omega)$.*

Note that the separation of micro and macro sequences guarantees that the overall sequence will be fair, as all applicable trigger is fired in each micro sequence.

The following lemma follows directly from the definition of the triggers:

Lemma 13 *If $T \subseteq U$, then $trigg_\Sigma(T) \subseteq trigg_\Sigma(U)$.*

The next result gives us the necessary and sufficient condition for the conditional chase to terminate.

Lemma 14 *If in the conditional-chase sequence we have $(T_i, \varphi_i) \cong (T_{i+1}, \varphi_{i+1})$ for some $i \in \omega$, then $(T_i, \varphi_i) \cong (T_j, \varphi_j)$ for all $j > i$.*

Proof: In this proof we will suppose that there are no unifiable tuples in T_i, thus the core computing step will not change the conditional table at any of the micro-steps. This is not a restriction as it can be very easily seen that all the steps from this proof hold for the computed core too by doing an induction, similar to the one in the proof of Theorem 54, on the size of the set $child_v$.

Let $i \in \omega$ such that $(T_i, \varphi_i) \cong (T_{i+1}, \varphi_{i+1})$. From this it follows that $T_i = T_{i+1}$ and that for each valuation v we have $v(T_i, \varphi_i) = v(T_{i+1}, \varphi_{i+1})$. Towards a contradiction, let us suppose that $(T_{i+1}, \varphi_{i+1}) \ncong (T_{i+2}, \varphi_{i+2})$, thus it must be that either $T_{i+1} \neq T_{i+2}$, or there exists valuation v such that $v(T_{i+1}, \varphi_{i+1}) \neq v(T_{i+2}, \varphi_{i+2})$. Consider first that $T_{i+1} \neq T_{i+2}$ from macro-step monotonicity Corollary 9 follows that there exists a tuple $t \in T_{i+2} \setminus T_{i+1}$. Thus, there exists trigger $\tau \in trigg_\Sigma(T_{i+1})$, where Σ is the set of TGD's considered that generates the new tuple t. But from the assumption we know that $T_i = T_{i+1}$, so $\tau \in trigg_\Sigma(T_i)$, meaning that tuple t needs to be also in T_{i+1} giving a contradiction with the assumption that $t \in T_{i+2} \setminus T_{i+1}$. From this it follows that it needs to be that $T_i = T_{i+1} = T_{i+2}$, thus also $trigg_\Sigma(T_i) = trigg_\Sigma(T_{i+1}) = trigg_\Sigma(T_{i+2})$. Next let us suppose that there exists valuation v such that $v(T_{i+1}, \varphi_{i+1}) \neq v(T_{i+2}, \varphi_{i+2})$. Let us now unfold the macro-steps into micro-steps as follows, considering $n = |trigg_\Sigma(T_i)|$:

$$(T_i, \varphi_i) \rightarrow_{\tau_1} (U_1, \phi_1) \rightarrow_{\tau_2} \dots (U_n, \phi_n) \rightarrow_{\tau_n} (T_{i+1}, \varphi_{i+1}) \tag{6.47}$$

$$(T_{i+1}, \varphi_{i+1}) \rightarrow_{\tau'_1} (V_1, \psi_1) \rightarrow_{\tau'_2} \dots (V_n, \psi_n) \rightarrow_{\tau'_n} (T_{i+2}, \varphi_{i+2}) \tag{6.48}$$

From the chase monotonicity property we have that there exists valuation v such that $v(T_{i+2}, \varphi_{i+2}) \setminus v(T_{i+1}, \varphi_{i+1}) \neq \varnothing$. That is there exists a smallest integer k such that $(V_{k-1}, \psi_{k-1}) \rightarrow_{\tau'_k} (V_k, \psi_k)$ and there exists a tuple $t \in v(V_k, \psi_k) \setminus v(V_{k-1}, \psi_{k-1})$, so that $t \in v(T_{i+2}, \varphi_{i+2}) \setminus v(T_{i+1}, \varphi_{i+1})$. From this we have that there exists a tuple $t_T \in T_i$ such that $v(t_T) = t$ and $v(\psi_k(t_T)) \equiv$ **true**. Because k is the smallest integer with this property, it follows that $v(\psi_l(t_T)) \equiv$ **false** for all $l \in [k-1]$ and also that $v(\phi_l(t_T)) \equiv$ **false** for all $l \in [n]$. As the set of triggers in both sequences 6.47 and 6.48 is the same, it naturally follows that there exists an integer $j \in [n]$ such that $\tau_j = \tau'_k$. Let $\tau'_k = (\xi, (\theta_1, \theta_2), T')$. Now, because $v(\psi_k(t_T)) \equiv$ **true** and $v(\psi_{k-1}(t_T)) \equiv$ **false**, it follows that, for the condition $\mu_k = \curlywedge(T', \psi_{k-1}) \wedge \curlywedge(\theta_2)$ induced by τ'_k, we have $v(\mu_k) \equiv$ **true**. Meaning that $v(\curlywedge(T', \psi_{k-1})) \equiv$ **true** and $v(\curlywedge(\theta_2)) \equiv$ **true**. On the other hand, the condition induced by trigger τ_j in sequence 6.47 is $\mu_j = \curlywedge(T', \phi_{j-1}) \wedge \curlywedge(\theta_2)$, and from our assumption that $t \notin v(U_j, \phi_j)$, it must be that $v(\mu_j) \equiv$ **false**. Thus, because $v(\curlywedge(\theta_2)) \equiv$ **true** it follows that $v(\curlywedge(T', \phi_{j-1})) \equiv$ **false**. This means that there exists a tuple $t^* \in T'$, such that $v(\phi_j(t^*)) \equiv$ **false**. On the other hand, we know that $v(\psi_k(t^*)) \equiv$ **true**. From this we have that one of the following needs to hold:

1. exists an integer $j < l \leq n$, such that $v(\phi_l(t^*)) \equiv$ **true**, or

2. exists an integer $0 \leq l < k$, such that $v(\psi_l(t^*)) \equiv$ **true**.

If the first case holds, because $v(\phi_l(t^*)) \equiv$ **true** and from the monotonicity corollary, it follows that $v(\varphi_{i+1}(t^*)) \equiv$ **true** and that $v(\varphi_i(t^*)) \equiv$ **false**. But this is a contradiction with the assumption that $(T_i, \varphi_i) \cong (T_{i+1}, \varphi_{i+1})$. So it needs to be that the second condition holds, that is $v(\phi_l(t^*)) \equiv$ **true** for a positive $l < k$. But again, this is a contradiction with the way we choose k to be minimal as $v(t^*) \in v(V_l, \varphi_l) \setminus v(T_{i+1}, \varphi_{i+1})$. From this it follows directly that it needs to be that $(T_{i+2}, \varphi_{i+2}) \cong (T_{i+1}, \varphi_{i+1})$∎

From the previous lemma it follows that, if for some $i \in \omega$, $(T_i, \varphi_i) \cong (T_{i+1}, \varphi_{i+1})$, then we have $(T_\omega, \varphi_\omega) \cong (T_i, \varphi_i)$. In this case we say that the chase sequence *converges in the finite.* From [3] we know that testing for two conditional tables if $(T_\omega, \varphi_\omega) \cong (T_i, \varphi_i)$ is Π_2^P-complete in general. In Section 6.5 we will present a class of dependencies that ensures the conditional-chase termination in polynomial time without needing to test the previous condition at each macro-step.

Example 53 *Continuing with the conditional table and the triggers from Example 52 we have that $(T, \varphi) \Rightarrow_\Sigma (T_1, \varphi_1)$, where (T_1, φ_1) is the conditional table (U_3, ψ_3) from the given example. Applying the sets of triggers on (T_1, φ_1), we get the following micro-step sequence:*

$$(T_1, \varphi_1) \rightarrow_{\tau_1} (U_4, \psi_4) \rightarrow_{\tau_2} (U_5, \psi_5) \rightarrow_{\tau_3} (T_2, \varphi_2).$$

Where (T_2, φ_2) is the following conditional table:

(T_2, φ_2)

t	$\varphi_2(t)$
$R(a, b)$	$X_1 = c \vee X_3 = e \vee X_3 = e$
$R(b, a)$	$X_2 = d$
$P(a, b)$	$X_3 = e$
$S(a, Y_1)$	$X_1 = c \vee (X_1 = c \vee X_3 = e)$
$S(b, Y_2)$	$X_2 = d \vee X_2 = d$

It is easy to check that $(T_1, \varphi_1) \ncong (T_2, \varphi_2)$. Applying the triggers in the same order yet another time, we get $(T_2, \varphi_2) \Rightarrow_\Sigma (T_3, \varphi_3)$, where (T_3, φ_3) is the following conditional table:

(T_3, φ_3)

t	$\varphi_3(t)$
$R(a,b)$	$X_1 = c \lor X_3 = e \lor X_3 = e \lor X_3 = d$
$R(b,a)$	$X_2 = d$
$P(a,b)$	$X_3 = e$
$S(a,Y_1)$	$X_1 = c \lor (X_1 = c \lor X_3 = e) \lor (X_1 = c \lor X_3 = e \lor X_3 = e)$
$S(b,Y_2)$	$X_2 = d \lor X_2 = d \lor X_2 = d$

In this case we have that $(T_2, \varphi_2) \cong (T_3, \varphi_3)$, so that $(T_\omega, \varphi_\omega) = (T_2, \varphi_2)$. Thus the result of the conditional chase was computed after three conditional macro-steps. Note that if we would apply the triggers in the second order from the same Example 52, then we would only need two macro steps to get to a congruent table to the one previously obtained.

Next we proceed with a series of lemmas which allow us to prove the main property of the conditional chase, namely that the representation of the conditional table returned by the chase process is the same with the instance form the constructible models semantics, i.e. $rep_{\mathbf{C}}(chase_{\Sigma}^{\mathbf{cond}}(T, \varphi)) = \Sigma(rep_{\mathbf{C}}(T, \varphi))$.

Lemma 15 *Let (T_0, φ_0) be a c-table, Σ a set of TGD's , v a valuation for (T_0, φ_0), $I_0 = v(T_0, \varphi_0)$ and $G_0 = ground(\Sigma)$. Also consider sequence:*

$$(I_0, G_0) \Rightarrow_{\alpha_1 \to \beta_1} (I_1, G_1) \Rightarrow_{\alpha_2 \to \beta_2} \cdots \Rightarrow_{\alpha_n \to \beta_n} (I_n, G_n) \Rightarrow_{\alpha_{n+1} \to \beta_{n+1}} \cdots \tag{6.49}$$

and the unfolding of a conditional-chase sequence:

$$(T_0, \varphi_0) \to_{\tau_1} (T_1, \varphi_1) \to_{\tau_2} \cdots \to_{\tau_n} (T_n, \varphi_n) \to_{\tau_{n+1}} \cdots \tag{6.50}$$

Then there exists a valuation v', extending v, such that the following holds:

1. *for any $i \in \omega$ (or $i \in [n]$ if sequence 6.49 is finite of size n) there exists an integer k such that $I_i \subseteq v'(T_k, \varphi_k)$, and*

2. *for any $j \in \omega$ (or $j \in [m]$ if sequence 6.50 is finite of size m) there exists an integer k such that $v'(T_j, \varphi_j) \subseteq I_k$.*

Proof: First we will inductively construct sequences $v_0 \Subset v_1 \Subset \ldots \Subset v_n \Subset \ldots$ and $(m_0, m_1, \ldots, m_n, \ldots)$ such that for any integer i we have $(I_i, \mathbf{true}) \subseteq v_i(T_{m_i}, \varphi_{m_i})$. Intuitively, m_i will be the index of the last micro-step from the unfold conditional-chase sequence 6.50, used to compute valuation v_i which is an extension of v_{i-1}.

First we set $v_0 = v$ and $m_0 = 0$ having $I_0 = v_0(T_o, \varphi_0)$. Let us suppose we constructed sequences $v_0 \Subset v_1 \Subset \ldots \Subset v_{k-1}$ and $(m_0, \ldots, m_{k-1})$ such that for all $i \in [k-1]$ we have $(I_i, \mathbf{true}) \subseteq v_i(T_{m_i}, \varphi_{m_i})$.

We construct now v_k as an extension of v_{k-1} as follows: let $\alpha_k \to \beta_k \in ground(\xi)$ be the k-th grounding applied in sequence 6.49. From the way a grounding is applied in the constructible models semantics (see Section 6.2), we have that $\alpha_k \subseteq I_{k-1}$. But from the inductive assumption and from the monotonicity of the conditional micro-step we have that $(I_{k-1}, \mathbf{true}) \subseteq v_{k-1}(T_j, \varphi_j)$ for any $j \geq k-1$. From this it follows that there exist an integer $j \geq m_{k-1}$ and a trigger $\tau_j = (\xi, (\theta_1, \theta_2), T')$ (guaranteed to exist because the way macro-steps are defined), such that (θ_1, θ_2) is a more general unifier than (w, v_{k-1}), where $w(body(\xi)) = \alpha_k$, $v_{k-1}(T', \varphi_{j-1}) = \alpha_k$ and T' is a v_{k-1}-core. Let $\theta_1^{\tau_j}(head(\xi))$ be the new tuples generated by trigger τ_j. This means that there exists a valuation v' such that $v'(v_{k-1}(\theta_1^{\tau_j}(head(\xi)))) = \beta_k$. Otherwise, if such a valuation would not exist, it would mean that trigger τ_j was already used in the inductive process with grounding $\alpha_l \to \beta_l$, $l < k$ where $\alpha_l = \alpha_k$. But this is a contradiction since, after applying the grounding $\alpha_l \to \beta_l$, the set of grounding G_l does not contain any groundings of the form $\alpha_l \to *$. Let (U_j, ψ_j) be the c-table obtained from c-table (T_{j-1}, φ_{j-1}) by applying trigger τ_j, without computing the conditional core. Thus $(T_j, \varphi_j) = \textit{CORE-COMP}(U_j, \psi_j)$. We know that $v_{k-1}(T', \varphi_{j-1}) \equiv \mathbf{true}$ from the way (U_j, ψ_j) is constructed (see Definition 46), so it follows that $v_{k-1}(\bigwedge\!\!\!\bigwedge(T', \varphi_{j-1})) \equiv \mathbf{true}$. Also, because (θ_1, θ_2) is a more general unifier than (w, v_{k-1}), it follows that $v_{k-1}(\bigwedge\!\!\!\bigwedge(\theta_2)) \equiv \mathbf{true}$. That is we have $\beta_k \subseteq v'(v_{k-1}(U_j, \psi_j))$, thus because from the construction we have $(U_j, \psi_j) \cong (T_j, \varphi_j)$ it follows that $\beta_k \in v'(v_{k-1}(T_j, \varphi_j)) = v'(v_{k-1}(U_j, \psi_j))$. We now set $v_k = v' \circ v_{k-1}$ and $m_k = j$. Clearly we have $v_{k-1} \Subset v_j$, $m_k > m_{k-1}$ and, because $I_k = I_{k-1} \cup \beta_k$, we have $(I_k, \mathbf{true}) \subseteq v_k(T_{m_k}, \varphi_{m_k})$.

Let $v' = v_0 \circ v_1 \circ \ldots \circ v_n \circ \ldots$. We first show that for any integer k, if $v'(T_k, \varphi_k)$ is a ground instance, then there exists an integer j such that $v'(T_k, \varphi_k) \subseteq I_j$. We will prove this by induction on k. For the base chase suppose $k = 0$. From the assumption we have $v'(T_0, \varphi_0) = v_0(T_0, \varphi_0) = I_0$. Now let us suppose $k > 0$, $v'(T_k, \varphi_k)$ is a ground instance and for any $l < k$ there exists an integer j_l such that $v'(T_l, \varphi_l) \subseteq I_{j_l}$. Towards a contradiction, let us suppose that $v'(T_k, \varphi_k) \notin I_i$ for any $i \geq 0$. Let $\tau_k = (\xi, (\theta_1, \theta_2), T')$ be the trigger from the chase sequence 6.50 used to obtain c-table (T_k, φ_k). Because, from the induction assumption, $v'(T_{k-1}, \varphi_{k-1}) \subseteq I_{j_l}$, it follows that it must be that $v'(\theta_2^{\tau_k}(head(\xi))) \notin I_i$ for any $i > j_l$. Let $v'(\theta_2^{\tau_k}(head(\xi))) = \beta$ and $v'(T', \varphi_{k-1}) = \alpha$, clearly $\alpha \subseteq I_{j_l}$. Because (T', φ_{k-1}) is a core, it follows that for any $T'' \subseteq T_{k-1}$, $T'' \neq T'$ it cannot be that $v'(T'', \varphi_{k-1}) = \alpha$. Thus, because $\alpha \to \beta$ is never applied in sequence 6.49, it follows that there exists β', different than β, such that $\alpha \to \beta'$ is applied in the constructible models sequence. But because there is no tableau $T'' \neq T'$ such that $v'(T'', \varphi_{k-1}) = \alpha$, it follows that, when constructing v', in the previous paragraph, for grounding $\alpha \to \beta'$ we used the trigger $\tau_n = \tau_k$ (it may be that we chose the trigger when applied at step n not at step k), thus making $v'(\theta_2^{\tau_n}(head(\xi))) = \beta'$. But this is a contradiction with the fact that $v'(\theta_2^{\tau_k}(head(\xi))) = \beta$ and $\beta \neq \beta'$.

In the previous paragraph we proved that, for any integer k, if $v'(T_k, \varphi_k)$ is a ground instance, then there exists an integer j such that $v'(T_k, \varphi_k) \subseteq I_j$. Now we will show that, for any integer k, the valuation $v'(T_k, \varphi_k)$ is a ground instance. Towards a contradiction let us suppose that there exists an integer k such that $v'(T_k, \varphi_k)$ is not a ground instance and $v'(T_{k-1}, \varphi_{k-1})$ is a ground instance. This means that, for trigger $\tau_k = (\xi, (\theta_1, \theta_2), T')$, we have that $v'(\theta_2^{\tau_k}(head(\xi)))$ contains nulls and $v'(\bigwedge\!\!\!\bigwedge(T', \varphi_{k-1}) \wedge \bigwedge\!\!\!\bigwedge(\theta_2)) \equiv \mathbf{true}$. Let $\alpha = v'(T', \varphi_{k-1})$. This means that, for any β, such that $\alpha \to \beta \in ground(\xi)$, the grounding $\alpha \to \beta$ was not applied in the constructible models sequence. But because $v'(T_{k-1}, \varphi_{k-1})$ is a ground instance, we previously proved that there exists an integer j such that $v'(T_{k-1}, \varphi_{k-1}) \subseteq I_j$, thus $\alpha \in I_j$. From the fairness applicability of the groundings in the constructible models sequence, it follows that there must be a grounding of the form $\alpha \to \beta' \in ground(\xi)$ that is applied in the sequence. But this is a contradiction with our assumption that such grounding is never applied. Thus $v'(T_k, \varphi_k)$ is a ground instance for any $k \geq 0$.

Adding all together, we constructed valuation v' such that for all $j \geq 0$ there exists a $k \geq 0$ such that $I_j \subseteq v'(T_k, \varphi_k)$, proving point (1) of the lemma. We also proved that, for all $k > 0$, $v'(T_k, \varphi_k)$ is a ground instance and, for any $i \geq 0$, there exists $k \geq 0$ such that $v'(T_i, \varphi_i) \subseteq I_k$, proving point (2) of the lemma∎

Lemma 16 *Let (T_0, φ_0) be a c-table and Σ a set of TGD's Also consider the unfolding of a conditional-chase sequence:*

$$(T_0, \varphi_0) \to_{\tau_1} (T_1, \varphi_1) \to_{\tau_2} \cdots \to_{\tau_n} (T_n, \varphi_n) \to_{\tau_{n+1}} \cdots \tag{6.51}$$

and valuation v over $\sqcup_{i \in \omega}(T_i, \varphi_i)$, $I_0 = v_0(T_0, \varphi_0)$.

Then there exists a constructible models sequence

$$(I_0, G_0) \Rightarrow_{\alpha_1 \to \beta_1} (I_1, G_1) \Rightarrow_{\alpha_2 \to \beta_2} \cdots \Rightarrow_{\alpha_n \to \beta_n} (I_n, G_n) \Rightarrow_{\alpha_{n+1} \to \beta_{n+1}} \cdots \tag{6.52}$$

and an ascending sequence $(p_0, p_1, \ldots, p_m, \ldots)$ such that, for any integer $i \in \omega$ (or $i \in [l]$ if sequence 6.51 is finite of size l), $v(T_i, \varphi_i) = I_{p_i}$. Even more, if sequence 6.51 is finite of size l, then sequence 6.52 is finite of size p_l.

Proof: We will inductively construct the sequences 6.52 and $(p_0, p_1, \dots, p_m, \dots)$ based on the number of micro-steps in 6.51. For the base case $k = 0$, we have $I_0 = v(T_0, \varphi_0)$, thus $p_0 = 0$. Let us suppose that $k > 0$. We have the ascending sequence $(p_0, p_1, \dots, p_{k-1})$ such that $I_{p_i} = v(T_i, \varphi_i)$, $i \in [k-1]$. If $v(T_k, \varphi_k) = v(T_{k-1}, \varphi_{k-1})$, then we set $p_k = p_{k-1}$. Thus, by induction assumption, we have $I_{p_k} = I_{p_{k-1}} = v(T_{k-1}, \varphi_{k-1}) = v(T_k, \varphi_k)$. Let us now consider that $v(T_k, \varphi_k) \neq v(T_{k-1}, \varphi_{k-1})$, from the monotonicity of the conditional micro-steps it follows that $v(T_{k-1}, \varphi_{k-1}) \subset v(T_k, \varphi_k)$. Let $\tau_k = (\xi, (\theta_1, \theta_2), T')$ be the trigger used in 6.51 to generate c-table (T_k, φ_k) from (T_{k-1}, φ_{k-1}). Le us denote with $\alpha = v(T', \varphi_{k-1})$ and $\beta = v(\theta_2^{\tau_k}(head(\xi)))$. Clearly we have $v(T_k, \varphi_k) = v(T_{k-1}, \varphi_{k-1}) \cup \beta$, and $\alpha \in v(T_{k-1}, \varphi_{k-1}) = I_{m_{k-1}}$. It remains to be proved that the grounding $\alpha \to \beta$ from $ground(\xi)$ is applicable on $(I_{p_{k-1}}, G_{p_{k-1}})$. That is, we need to prove that $\alpha \to \beta \in G_{p_{k-1}}$. Towards a contradiction, let us suppose that $\alpha \to \beta \notin G_{m_{k-1}}$, thus in the sequence:

$$(I_0, G_0) \Rightarrow_{\alpha_1 \to \beta_1} (I_1, G_1) \Rightarrow_{\alpha_2 \to \beta_2} \cdots \Rightarrow_{\alpha_{p_{k-1}} \to \beta_{p_{k-1}}} (I_{p_{k-1}}, G_{p_{k-1}}) \tag{6.53}$$

there exists an integer $j < k$ such that $\alpha_{p_j} = \alpha$. Let $\tau_j = (\xi, (\theta'_1, \theta'_2), T'')$ be the trigger used at step j in the chase sequence 6.51. Clearly $\tau_j \neq \tau_k$, otherwise they would generate the same output instance. Following that, it needs to be that $T'' \neq T'$. Thus from the monotonicity of the micro-step, it needs to be that there exists a tuple $t' \in T' \setminus T''$. On the other hand, we also have that $\alpha = v(T', \varphi_k) = v(T'', \varphi_k)$, but from the micro-step definition we have that all new tuples generated in T' succeeds (based on $\lessdot$ order) the tuples in T''. Because $v(T', \varphi_k) = v(T'', \varphi_k)$ it follows that there exists a tuple $t'' \in T''$, such that $v(t') = v(t'')$ and $v(\varphi_k(t')) \equiv \textbf{true}$ and $v(\varphi_j(t'')) \equiv \textbf{true}$. Thus t' and t'' are unifiable and $t'' \lessdot t'$. On the other hand, because $t'' \lessdot t'$, the $CORE\text{-}COMP$ algorithm adds a new conjunction to the local condition t' not allowing it to be equal with t''. Following that, it cannot be that $v(\varphi_k(t')) \equiv \textbf{true}$, beacuse it is a contradiction with our previous assumption on tuple t'. From this, it follows that $\alpha \to \beta$ is applicable on

$(I_{p_{k-1}}, G_{p_{k-1}})$. We set $p_k = p_{k-1} + 1$ and $I_{p_k} = I_{p_{k-1}} \cup \beta$.

Let us now suppose that the conditional chase terminates after a chase sequence of size l. Towards a contradiction let us suppose that there exists $\alpha \to \beta \in ground(\xi)$ such that $\alpha \to \beta \in G_{p_l}$ and $\beta \notin I_{p_l}$. Because $v(T_l, \varphi_l) = I_{p_l}$, this would mean that there exists trigger $\tau = (\xi, (\theta_1, \theta_2), T')$ such that $v(\bigwedge\!\!\!\!\bigwedge(T', \varphi_l) \wedge \bigwedge\!\!\!\!\bigwedge(\theta_2)) \equiv \mathbf{true}$ and $v(T', \varphi_l) = \alpha$. On the other hand, because in the chase sequence we would add another step $(T_l, \varphi_l) \to_\tau (U, \psi)$, we would have that $v(T_l, \varphi_l) = v(U, \psi)$, meaning that the trigger $\tau = (\xi, (\theta_1, \theta_2), T')$ from the sequence 6.51 was already applied at step $k \leq l$. Consequently, in the constructible models sequence we would have eliminated $\alpha \to *$ from G_k, which means that G_l cannot have a grounding of the form $\alpha \to *$. This contradicts with our assumption that such grounding exists. Therefore it must be that the constructible models sequence is also finite and it has size p_l ■

From the construction of the sequence $(p_0, p_1, \ldots, p_m, \ldots)$ in the previous proof, it can be noted that sequence $(1, 2, \ldots, m, \ldots)$ is a subsequence of $(p_0, p_1, \ldots, p_m, \ldots)$. This is because it may be that $p_i = p_{i+1}$ for some i.

We are now ready to state the main theorem of this chapter, which says that the result of the conditional chase represents exactly the constructible models.

Theorem 55 *Let (T_0, φ_0) be a conditional table and Σ a set of TGD's, then:*

$$\Sigma(rep_{\mathbf{C}}(T, \varphi)) = rep_{\mathbf{C}}(chase^{\mathbf{cond}}_{\Sigma}(T, \varphi)).$$

Proof: First we will show that $\Sigma(rep_{\mathbf{C}}(T, \varphi)) \subseteq rep_{\mathbf{C}}(chase^{\mathbf{cond}}_{\Sigma}(T, \varphi))$. For this let us consider $I_0 \in rep_{\mathbf{C}}(T, \varphi)$, so there exists a valuation v such that $v(T, \varphi) = I_0$. Now let $J \in \Sigma(I_0)$, this means there exists a sequence:

$$(I_0, G_0) \Rightarrow_{\alpha_1 \to \beta_1} (I_1, G_1) \Rightarrow_{\alpha_2 \to \beta_2} \cdots \Rightarrow_{\alpha_n \to \beta_n} (I_n, G_n) \Rightarrow_{\alpha_{n+1} \to \beta_{n+1}} \cdots$$

such that $J = \bigcup_{i \in \omega} I_i$. Let us now consider the unfolding of the conditional-chase sequence, where $(T_0, \varphi_0) = (T, \varphi)$:

$$(T_0, \varphi_0) \rightarrow_{\tau_1} (T_1, \varphi_1) \rightarrow_{\tau_2} \cdots \rightarrow_{\tau_n} (T_n, \varphi_n) \rightarrow_{\tau_{n+1}} \cdots$$

From Lemma 15 we know that there exists a valuation v such that, for any $i \in \omega$, there exists a $j \in \omega$ such that $I_i \subseteq v(T_j, \varphi_j)$, and, for any $j \in \omega$, there is an $i \in \omega$ such that $v(T_j, \varphi_j) \subseteq I_i$. From this it follows that $\bigcup_{i \in \omega} I_i = \bigcup_{j \in \omega} v(T_j, \varphi_j)$. On the other hand, we know that the valuation is continuous with respect to the union on c-tables, $\bigcup_{j \in \omega} v(T_j, \varphi_j) = v(\bigsqcup_{j \in \omega}(T_j, \varphi_j))$. Thus we have:

$$J = \bigcup_{i \in \omega} I_i = v(\bigsqcup_{j \in \omega}(T_j, \varphi_j)) = v(chase_{\Sigma}^{\mathbf{cond}}(T, \varphi))$$

Following that, $J \in rep_{\mathbf{C}}(chase_{\Sigma}^{\mathbf{cond}}(T, \varphi))$. So by generalization it directly follows that $\Sigma(rep_{\mathbf{C}}(T, \varphi)) \subseteq rep_{\mathbf{C}}(chase_{\Sigma}^{\mathbf{cond}}(T, \varphi))$.

Let us now prove the other direction, that is: $rep_{\mathbf{C}}(chase_{\Sigma}^{\mathbf{cond}}(T, \varphi)) \subseteq \Sigma(rep_{\mathbf{C}}(T, \varphi))$. For this, let us consider the unfolded conditional-chase sequence:

$$(T_0, \varphi_0) \rightarrow_{\tau_1} (T_1, \varphi_1) \rightarrow_{\tau_2} \cdots \rightarrow_{\tau_n} (T_n, \varphi_n) \rightarrow_{\tau_{n+1}} \cdots$$

and let us also consider a valuation v on this sequence. From Lemma 16, it follows that there exists a constructible models sequence:

$$(I_0, G_0) \Rightarrow_{\alpha_1 \rightarrow \beta_1} (I_1, G_1) \Rightarrow_{\alpha_2 \rightarrow \beta_2} \cdots \Rightarrow_{\alpha_n \rightarrow \beta_n} (I_n, G_n) \Rightarrow_{\alpha_{n+1} \rightarrow \beta_{n+1}} \cdots$$

and an ascending sequence $(p_0, p_1, \ldots, p_m, \ldots)$ such that $v(T_k, \varphi_k) = I_{p_k}$. As noted, the sequence $(1, 2, \ldots, m, \ldots)$ is a subsequence of $(p_0, p_1, \ldots, p_m, \ldots)$. It then follows that $\bigcup_{i \in \omega} I_i = \bigcup_{j \in \omega} v(T_j, \varphi_j)$. From this it follows directly that $v(\bigsqcup_{j \in \omega}(T_j, \varphi_j)) = \bigcup_{i \in \omega} I_i$, thus $v(\bigsqcup_{j \in \omega}(T_j, \varphi_j)) \in \Sigma(I)$ and by generalization $rep_{\mathbf{C}}(chase_{\Sigma}^{\mathbf{cond}}(T, \varphi)) \subseteq \Sigma(rep_{\mathbf{C}}(T, \varphi))$■

A direct corollary of this theorem is that, by using conditional chase in data exchange, one may represent the target instance as a conditional table representing exactly the solution space under constructible models semantics.

Corollary 10 *Let $M = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ be a data exchange mapping and I be a ground instance. If the conditional chase terminates with input I and $\Sigma = \Sigma_{st} \cup \Sigma_t$, then:*

$$rep_{\mathbf{C}}(chase_{\Sigma}^{\mathbf{cond}}(I, \mathbf{true}))|_{\mathbf{T}} = Sol_M^{\mathtt{CM}}(I).$$

6.5 On Conditional-Chase Termination

In Chapter 4 we presented some classes of TGD's which guarantee the termination for different chase variations. In this section we will present a class of TGD's which guarantees the conditional-chase termination. Let us denote by $\mathsf{CT}_{\forall\forall}^{\mathbf{cond}}$ the class of TGD's such that for any $\Sigma \in \mathsf{CT}_{\forall\forall}^{\mathbf{cond}}$ the conditional-chase algorithm terminates with any c-tables (T, φ) and Σ.

The following proposition shows that the conditional chase does not terminate for a more larger class of TGD's than the oblivious chase.

Proposition 19 $\mathsf{CT}_{\forall\forall}^{\mathbf{cond}} \subsetneq \mathsf{CT}_{\forall\forall}^{\mathbf{obl}}$.

Proof: For the subset part, it is easy to see that any conditional-chase sequence on c-table (T, φ) with Σ set of TGD's is a superset for the oblivious-chase sequence on T and Σ. Thus, if the conditional chase terminates for all conditional tables on Σ, it follows that the oblivious chase on all instances will terminate on Σ too. For the strict inclusion part, let us consider $\Sigma = \{R(x, y, y) \rightarrow \exists z, v\ R(x, z, v)\}$. Using the technique in [59] it can easily be shown that the standard chase with $\widehat{\Sigma}$ terminates on all instances. From this and Theorem 3 it follows that the oblivious chase with Σ terminates on all instances. Consider then the conditional chase starting with c-table $\{R(a, b, b)\}$ (the local condition of $R(a, b, b)$ is **true**). The result of this chase can be represented by the infinite conditional table below:

t	$\varphi(t)$
$R(a, b, b)$	**true**
$R(a, X_1, X_2)$	**true**
$R(a, X_3, X_4)$	$X_1 = X_2$
$R(a, X_5, X_6)$	$(X_1 = X_2) \wedge (X_3 = X_4)$
...	...

This proves the strict inclusion part as well■

In order to obtain the sufficient condition for the conditional-chase termination, we introduce a new rewriting of a set of TGD's Σ, called *disjoining*, and denoted $\ddot{\Sigma}$. Given a TGD ξ, we denote with $\ddot{\xi}$ the same dependency, where for each variable x repeated in the body we replace all occurrences, except the first, of x in the body with a fresh new universally quantified variable. For instance, if ξ denotes the dependency $R(x, y, x), R(y, z, x) \rightarrow \exists v, w\ R(x, v, w)$, then $\ddot{\xi}$ will denote the dependency $R(x, y, x_1), R(y_1, z, x_2) \rightarrow \exists v, w\ R(x, v, w)$, where the subscripted variables are fresh. The set $\ddot{\Sigma}$ is then $\{\ddot{\xi} : \xi \in \Sigma\}$.

There is of course a difference between the conditional and the oblivious chase: the former may fire based on unifiers (θ_1, θ_2) and the latter based only on homomorphisms θ_1. From this observation we obtain the following lemma.

Lemma 17 *$chase_{\Sigma}^{\mathbf{cond}}(T, \varphi)$ terminates if $chase_{\ddot{\Sigma}}^{\mathbf{obl}}(T)$ terminates.*

From Definition 4 we observe that the extended dependency graph for $\ddot{\Sigma}$ can be obtained from the extended dependency graph of Σ by deleting edges, thus we have:

Lemma 18 *If Σ is a richly acyclic set of TGD's, then $\ddot{\Sigma}$ is also richly acyclic.*

We now have the main result of this section

Theorem 56 $\mathsf{RA} \subsetneq \mathsf{CT}^{\mathbf{cond}}_{\forall\forall}$ *and for any* $\Sigma \in \mathsf{RA}$ *and any conditional table* (T, φ) *there is a polynomial that bounds the execution time of the conditional-chase algorithm with* (T, φ) *and* Σ*.*

Proof. Let (T, φ) be a conditional table and Σ a richly acyclic set of TGD's. By Lemma 18, the set $\ddot{\Sigma}$ is also richly acyclic, and by Lemma 1, the set $\widehat{\ddot{\Sigma}}$ is weakly acyclic. By [27], $chase^{\mathbf{std}}_{\widehat{\ddot{\Sigma}}}(T)$ terminates in polynomial time. By Theorem 14, $chase^{\mathbf{obl}}_{\ddot{\Sigma}}(T)$ also terminates in polynomial time, and then by Theorem 3, $chase^{\mathbf{cond}}_{\Sigma}(T, \varphi)$ terminates in polynomial time in size of (T, φ).

For the strict inclusion part, consider $\Sigma = \{R(x, x) \rightarrow \exists y \; R(x, y)\}$. It is easy to see that $\Sigma \notin \mathsf{RA}$. On the other hand, for each tuple that is unified with the body of the dependency a new tuple is created. But this new tuple contributes in creating a new tuple if and only if it is the same as the initial tuple. Thus, the conditional-chase procedure will terminate in maximum n macro steps, where n is the size of the input conditional table∎

Chapter 7

Chasing with Second Order Dependencies

The notion of representation systems describes structures that are algebraically closed under queries. For example, complete instances are closed under FO queries. This means that given any ground instance and any FO query, one may represent the result of the query as a ground instance. Unfortunately, as clarified in Chapter 6, the previous property does not hold when dealing with incomplete databases. Therefore, the recent conclusions stipulate that the representation systems are also highly relevant in the context of data exchange. In the previous section we also have showed that if the data exchange mapping is specified by a richly acyclic set of TGD's, then for each source data specified by a conditional table, there exists a conditional table over the target such that it can be used to get the certain and possible answers to any FO query. Unfortunately, there are many mappings that cannot be specified by sets of TGD's and EGD's. In this chapter, we will develop the previous result showing that we can extend the previous class of mappings to *ordinary* SO dependencies which are specified by source-to-target second order dependencies and by richly acyclic target

TGD's and EGD's. Second order dependencies plays an important role in composing schema mappings. They were first introduced by Fagin et al. in [29] for source to target dependencies and it was proved that the class of SO-tgd's is the right language for mapping composition. Later on this language was extended by adding weakly acyclic target dependencies as well by Arenas, Fagin and Nash [9].

7.1 Second Order Dependencies

We introduced in Section 6.2 the constructible models semantics for dependencies specified by a set of TGD's. Here we extend this semantics for a higher class of dependencies *source-to-target second order dependencies.* For this, let us first present the notions of *term* and *source-to-target second order dependencies.*

Definition 49 *Let $\bar{x}$ be a sequence of variables and $\bar{f}$ be a sequence of function symbols. A* term *based on $\bar{x}$ and $\bar{f}$ is defined recursively as follows:*

- *every variable in $\bar{x}$ is a term;*
- *every 0-ary function symbol in $\bar{f}$ is a term;*
- *if f is a function symbol of arity k, then $f(t_1, \dots, t_k)$ is a term, where all t_i are terms for $i \in [k]$.*

Definition 50 *Given a source schema* **S** *and a distinct target schema* **T**, *a* source-to-target second order dependency *is a formula of the form:*

$$\exists f_1 \dots \exists f_m((\forall \bar{x}_1(\alpha_1 \to \beta_1)) \wedge \dots \wedge (\forall \bar{x}_n(\alpha_n \to \beta_n)))$$

where for each $i \in [m]$ and each $k \in [n]$:

- *f_i is a function symbol;*

- *α_k is a conjunction of atoms $R(\bar{y})$ or equalities $t_1 = t_2$. With $R \in \mathbf{S}$, $\bar{y}$ contains variables from $\bar{x}_k$ and t_1, t_2 are terms based on variables from $\bar{x}_i$ and functions $f_1, \dots, f_n$;*

- *β_k is a conjunction of atoms $S(\bar{t})$, where $S \in \mathbf{T}$ and $\bar{t}$ are terms based on $\bar{x}_k$ and $\{f_1, \dots, f_m\}$;*

- *each variable in $\bar{x}_k$ occurs in some relational atomic formula in α_k.*

We denote by st-SO dependencies we denote the class of all source-to-target second order dependencies.

The last item in the previous definition ensures the safeness of the dependency. Thus, the formula:

$$\exists f, \exists g\ (\forall x, y\ R(x,y), f(x) = g(y) \rightarrow S(x,y)) \wedge (\forall x, y\ R(x,x), f(x) = g(y) \rightarrow T(x,y))$$

is not st-SO dependencies as the second dependency contains universally quantified variable y which does not occur in any relational atomic formula in the body.

Given Σ a st-SO dependencies, a source instance I and a target instance J, we say that (I, J) is a model for Σ, denoted by $I \cup J \vDash \Sigma$, if $I \cup J$ satisfies Σ in the standard model theoretic sense.

Example 54 [29] *Consider the source schema $\mathbf{S} = \{Emp\}$, where Emp is a unary relation containing a list of employees; and a target schema $\mathbf{S} = \{Mgr, SelfMgr\}$, where Mgr is a binary relation for the employees; and their manager and relation $SelfMgr$ is a unary relation that maintains all the self managers. Let us also consider st-SO dependencies:*

$$\exists f \big(\forall e\ (Emp(e) \rightarrow Mgr(e, f(e))) \wedge \forall e\ (Emp(e) \wedge (e = f(e)) \rightarrow SelfMgr(e))\big)$$

Intuitively, the dependencies mention that for each employee in Emp there exists a manager given by the function f; and if for an employee the manager is the same as the employee ($e = f(e)$), then that employee is a self-manager. Consider, for example, the following instances:

I_1	J_1
$Emp(john)$	$Mgr(john, mike)$

I_2	J_2
$Emp(john)$	$Mgr(john, mike)$
$Emp(ann)$	$Mgr(ann, mike)$
$Emp(mike)$	$Mgr(mike, mike)$
	$SelfMgr(mike)$

I_3	J_3
$Emp(john)$	$Mgr(john, mike)$
$Emp(ann)$	$Mgr(ann, mike)$
	$Mgr(mike, mike)$

I_4	J_4
$Emp(john)$	$Mgr(john, mike)$
$Emp(ann)$	$Mgr(ann, mike)$
$Emp(mike)$	$Mgr(mike, mike)$

In this example $I_1 \cup J_1 \models \Sigma$, $I_2 \cup J_2 \models \Sigma$, $I_3 \cup J_3 \models \Sigma$, even if the tuple $Mgr(mike, mike)$ exists, there is no employee "mike" under the source Emp relation. Finally, $I_4 \cup J_4 \nvDash \Sigma$ because there exists an employee "mike" whose manager is himself, but there is no entry for him in the $SelfMgr$ relation.

The following example shows that the membership problem for st-SO dependencies is NP-hard (i.e. *Given I and Σ, does $I \models \Sigma$?*).

Example 55 [29] *Let us consider the followings schemata $\mathbf{S} = \{E\}$ and $\mathbf{T} = \{D\}$; and the following st-SO dependencies:*

$$\Sigma : \; \exists f \; (\forall x, y \; E(x,y) \rightarrow D(f(x), f(y)))$$

We will reduce now the problem of 3-colorability graph problem to the membership problem: Is instance $I \models \Sigma$? *Consider a graph $G = (V, E)$. We construct instance*

I as follows: for each edge (v_1, v_2) from G add tuple $E(v_1, v_2)$ in I. Also consider $D^I = \{(g,y),(y,g),(g,b),(b,g),(b,y),(y,b)\}$, where g, b and y stand for colors green, blue *and* yellow *respectively.*

It is obvious that, if there exists an interpretation for function f in Σ, then G is 3-colorable, as the interpretation for f gives the colors for the vertexes. Also clearly if G is 3-colorable, then there exists an interpretation of f such that $I \vDash \Sigma$.

Consider Δ_{F} a countable set of function names and function *arity* that assigns an integer for each function symbols in Δ_{F}. A term over Δ_{F} is defined recursively as follows:

- any constant $a \in \Delta_{\mathsf{C}}$ is a term,
- any null $a \in \Delta_{\mathsf{C}}$ is a term,
- any function symbol $f \in \Delta_{\mathsf{F}}$ with $arity(f) = 0$ is a term,
- for any function symbol $f \in \Delta_{\mathsf{F}}$ with $arity(f) = k$, $f(t_1, t_2, \ldots, t_k)$ is a term, where for any $i \in [k]$, t_i are terms.

We extend the notion of instance to Skolem instance by allowing as elements in the instance terms and not only constants and nulls.

Example 56 *Consider f, g two function symbols from Δ_{F} such that $arity(f) = 2$ and $arity(g) = 1$. The following is a Skolem instance using the function symbols f and g:*

I
$R(a, X, b, Y)$
$R(b, f(X, a), c, Y)$
$R(g(Y), f(a, a), c, f(a, g(X)))$

Let Σ be st-SO dependencies:

$$\exists f_1 \dots \exists f_m((\forall \bar{x}_1(\alpha_1 \to \beta_1)) \wedge \dots \wedge (\forall \bar{x}_n(\alpha_n \to \beta_n)))$$

For such st-SO dependencies we say that the size of Σ is n, denoted by $|\Sigma|$. For each dependency $\xi_i :\ \forall \bar{x}_i(\alpha_i \to \beta_i)$, we denote by *body*$(\xi_i)$ to be the Skolem instance that contains a tuple for each atom in $\hat{\alpha}_i$, where each universally quantified variable from $\bar{x}_i$ is mapped by a fresh new null value; where $\hat{\alpha}_i$ is the formula obtained from α_i by replacing each equality atom by a new binary relational symbol E, called equality relation. Similarly, we define the Skolem instance *head*(ξ_i).

Example 57 *Let Σ be st-SO dependencies from example 54:*

$$\exists f\big(\forall e\ (Emp(e) \to Mgr(e, f(e))) \wedge \forall e\ (Emp(e) \wedge (e = f(e)) \to SelfMgr(e))\big)$$

For this we have $|\Sigma| = 2$. If we consider $\xi_1 = \forall e\ (Emp(e) \to Mgr(e, f(e)))$ and $\xi_2 = \forall e\ (Emp(e) \wedge (e = f(e)) \to SelfMgr(e))$, then:

$$\begin{array}{rcl} body(\xi_1) & = & \{Emp(X)\} \\ head(\xi_1) & = & \{Mgr(X, f(X))\} \\ body(\xi_2) & = & \{Emp(Y), E(Y, f(Y))\} \\ head(\xi_2) & = & \{SelfMgr(Y)\} \end{array}$$

Note that even if in the st-SO dependencies both dependencies use universally quantified variable e, this variable is mapped to a new null according to its scope. Thus, in the scope of the first occurrence, e is mapped to null X and, in the scope of second occurrence, it is mapped to Y.

7.2 Extending Constructible Models Semantics

7.2.1 Adding EGD's

Before extending our constructible models semantics (introduced in Section 6.2) to include st-SO dependencies, we have to define the constructible models semantics for a more general set of dependencies, namely sets of TGD's and EGD's. For this, let us see how an EGD is applied on a node (I, G). Let $\forall \bar{x}\ \alpha(\bar{x}) \rightarrow x = y$ be an EGD, where x, y are variables in *seqx*. For this EGD, we define the set of groundings:

$$ground(\forall \bar{x}\ \alpha(\bar{x}) \rightarrow x = y) = \bigcup_{i \in \omega,\ v_i(x) \neq v_i(y)} \{\alpha(v_i(\bar{x})) \rightarrow \bot\}$$

where v_i, $i \in \omega$, is an enumeration of all valuations from the set of variables from $\bar{x}$ to the countable set Δ_{C}. Given a set $\Sigma = \{\xi_1, \ldots, \xi_n\}$ and a set of TGD's and EGD's, we define the set of groundings $ground(\Sigma) = ground(\xi_1) \cup \ldots \cup ground(\xi_n)$. Let I be a ground instance and G a set of finitely generated ground dependencies from Σ. Let I be an instance and $(\alpha \rightarrow \bot) \in G$ be a grounding of an EGD $\xi \in \Sigma$, if $\alpha \subseteq I$, then we say that (I, G) *fails* with the grounding $(\alpha \rightarrow \bot)$ and it is denoted as $(I, G) \Rightarrow_{\alpha \rightarrow a = b} \bot$. Consider now the sequence:

$$(I_0, G_0), (I_1, G_1), \ldots, (I_n, G_n), \ldots$$

where $((I_0, G_0) = (I, G)$ and $(I_i, G_i) \Rightarrow_{*} (I_{i+1}, G_{i+1})$, where $* \in G_i$ is either an TGD or EGD grounding. If there exists an integer i such that $(I_i, G_i) \Rightarrow_{\alpha \rightarrow \bot} \bot$, we say that the sequence is *failing*. For a non-failing sequence it is obvious that $I_i \subseteq I_{i+1}$ and that $G_i \supseteq G_{i+1}$ for each $i \in \omega$. For the non-falling sequence, we define the *limit of a chase sequence* as:

$$(\bigcup_{i \in \omega} I_i, \bigcap_{i \in \omega} G_i).$$

Similarly to the TGDcase, we use the notation $\mathcal{C}(I, G)$ for the set of the limits of *all* chase sequences originating from (I, G). With this we can define the set of all constructible models of I and Σ as:

$$\Sigma(I) = \{J \,:\, (J, G) \in \mathcal{C}\big(I, ground(\Sigma)\big), G \subseteq ground(\Sigma)\}.$$

Example 58 *Figure 7.1 represents the constructible models chase tree corresponding to the instance* $I = \{R(a), R(b)\}$ *and* Σ *containing the following TGD and EGD:*

$$\begin{aligned} R(x) &\rightarrow \exists y\, S(x, y) \\ S(x, y) &\rightarrow x = y. \end{aligned}$$

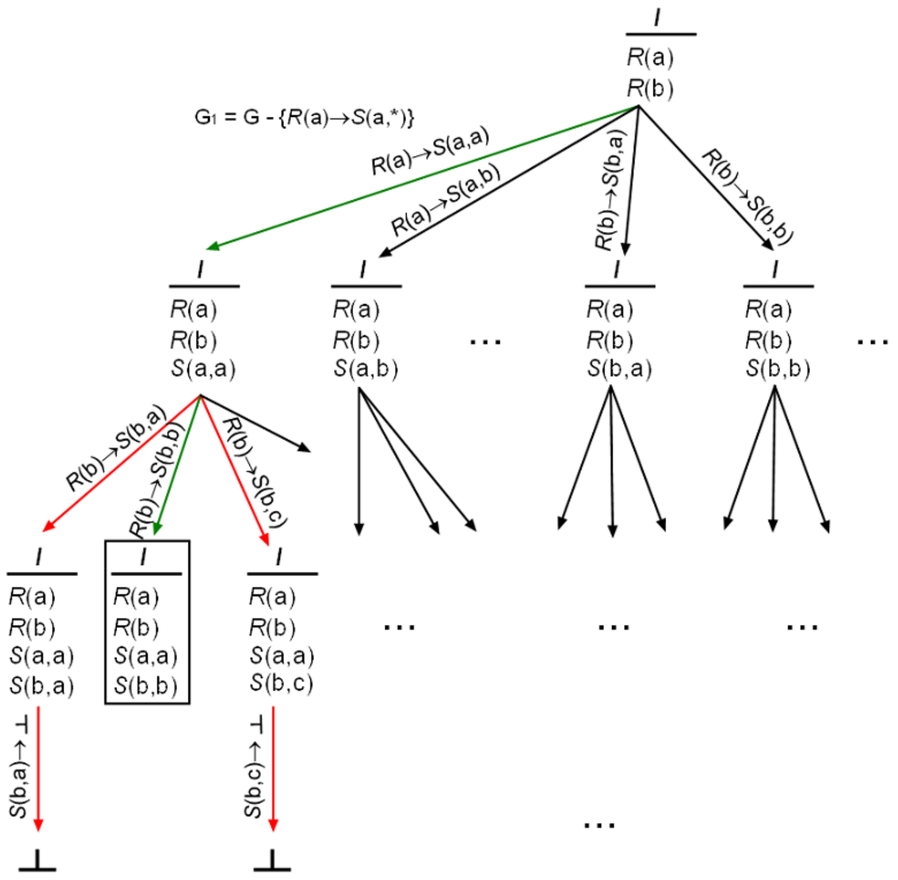

Figure 7.1: Chase tree with TGD's and EGD's

7.2.2 Adding st-SO dependencies

Now we are ready to extend even more the notion of constructible models semantics in such a way that it also allows st-SO dependencies. For this, let Σ be st-SO dependencies of the form:

$$\exists f_1 \ldots \exists f_m((\forall \bar{x}(\alpha_1 \to \beta_1)) \wedge \ldots \wedge (\forall \bar{x}_n(\alpha_n \to \beta_n))) \tag{7.1}$$

and let $F = \{f_1, \ldots, f_m\}$ be the set of function symbols therein. Let $F^\flat$ be an interpretation of F in the universe of Δ_C. We construct with $F^\flat$ the set of groundings $G_{F^\flat}$ as follows: for each $\forall \bar{x}_i\ \alpha_i \to \beta_i$ in 7.1, for $i \in [n]$ and for each valuation v_j, $j \in \omega$, for $\bar{x}_i$ with values in Δ_C add grounding $\alpha'_i \to \beta'_i$ to G. Where, α'_i and β'_i are obtained from α_i and β_i respectively by replacing recursively each variable $x \in \bar{x}_i$ with the value $v_i(x)$ and each term $f_k(\bar{a})$, where $\bar{a}$ is a sequence of constants, with the constant given by the corresponding interpretation for f_k in $F^\flat$. We will not consider in $G_{F^\flat}$ the groundings that have an equality of the form $a = b$ in the body of the grounding, with a, b as two distinct constants. Also, we will eliminate all the equalities of the form $a = a$ from the body of the grounding. Similarly, if there is a grounding $\alpha \to \beta$ in $G_{F^\flat}$ such that there exists an equality of the form $a = b$ in the head, with a, b two distinct constants, the grounding is replaced with $\alpha \to \bot$. Also, we eliminate from β all equalities of the form $a = a$. Thus, there will be no groundings with equalities in $G_{F^\flat}$.

Example 59 *Let us also consider the same st-SO dependencies from Example 54:*

$$\exists f \big(\forall e\ (Emp(e) \to Mgr(e, f(e))) \wedge \forall e\ (Emp(e) \wedge (e = f(e)) \to SelfMgr(e))\big)$$

Consider also $F_1^\flat$ that interprets f as identity function. The set $G_{F_1^\flat}$ is defined under this interpretation as the following TGD's and EGD's groundings:

$$\begin{aligned}G_{F_1^\flat} = \{&Emp(john) \rightarrow Mgr(john, john);\ Emp(john) \rightarrow SelfMgr(john)\\ &Emp(ann) \rightarrow Mgr(ann, ann); Emp(ann) \rightarrow SelfMgr(ann);\\ &Emp(mike) \rightarrow Mgr(mike, mike); Emp(mike) \rightarrow SelfMgr(mike), \ldots\}\end{aligned}$$

Clearly under this interpretation all employees are self managers. Now let us consider the interpretation that assigns $f(a) = mike$ to all constants a.

$$\begin{aligned}G_{F_2^\flat} = \{&Emp(john) \rightarrow Mgr(john, mike);\ Emp(ann) \rightarrow Mgr(ann, mike);\\ &Emp(mike) \rightarrow Mgr(mike, mike); Emp(mike) \rightarrow SelfMgr(mike), \ldots\}\end{aligned}$$

Note that we eliminated all the groundings in the $G_{F_2^\flat}$ of the form:

$$Emp(john), john = mike \rightarrow SelfMgr(john).$$

Given an instance I and $G_{F^\flat}$ a set of groundings associated with the interpretation $F^\flat$ and st-SO dependencies Σ, we then define the sequence:

$$(I_0, G_0), (I_1, G_1), \cdots, (I_n, G_n), \cdots$$

where $(I_0, G_0) = (I, G_{F^\flat})$ and for each $i > 0$ instance I_i is obtained from I_{i-1} by applying grounding $\alpha \rightarrow \beta$ with $\beta \neq \bot$, where $G_i = G_{i-1} \setminus \{\alpha \rightarrow *\}$ and $I_i = I_{i-1} \cup \beta$. If there exists an integer i and a grounding $(\alpha \rightarrow \bot) \in G_i$ such that $\alpha \subseteq I_i$, then we say that the *sequence fails.*

Similarly to the embedded dependencies case, we define the limit for all non failing sequences as:

$$(\bigcup_{i \in \omega} I_i, \bigcap_{i \in \omega} G_i).$$

The notation $\mathcal{C}(I, G_{F^\flat})$ will stand for the set of the limits of all chase sequences originating from $(I, G_{F^\flat})$. We are now ready to defined $\Sigma_{F^\flat}(I)$

$$\Sigma_{F^\flat}(I) = \{J \,:\, (J,G) \in \mathcal{C}\big(I, G_{F^\flat}\big)\}$$

Finally for Σ a st-SO dependencies and I we define $\Sigma(I)$ to be:

$$\Sigma(I) = \bigcup_{F^\flat \text{ - interpretation}} \Sigma_{F^\flat}(I)$$

Example 60 *Continuing with the st-SO dependencies from Example 59, with the interpretation $F_1^\flat$ and the instance $I = \{Emp(john), Emp(mike)\}$, we have:*

$$\Sigma_{F^\flat}(I) = \{\{Emp(john), Emp(mike), Mgr(john, john), Mgr(mike, mike), \\ SelfMgr(john), SelfMgr(mike)\}\}$$

Given a mapping $M = (\mathbf{S}, \mathbf{T}, \Sigma, \varnothing)$, where Σ is a st-SO dependencies, and given a source instance I, we extend the notion of constructible models solution space to st-SO dependencies as follows:

$$Sol_M^{\texttt{CM}}(I) = \{J \in Inst(\mathbf{T}) |\ \exists K \in \Sigma(I) \text{ and } J = K|_{\mathbf{T}}\}.$$

7.2.3 Adding st-SO dependencies and target TGD's

We need to extend the constructible models semantics to allow, beside st-SO dependencies, the target TGD's. In order to achieve this, let us consider $\Sigma = \Sigma_{stSO} \cup \Sigma_t$, where Σ_{stSO} represents the st-SO dependencies and Σ_t identifies a set of TGD's. Let us denote by $G_{\mathbf{t}}$ the set of all groundings corresponding to Σ_t, as described in Section 6.2. For each interpretation $F^\flat$ of the function symbols from Σ_{stSO}, we consider the set $G_{F^\flat}$ which corresponds to the source-to-target second order dependencies.

Given an instance I and the sets of groundings $G_{F^\flat}$ associated with an interpretation $F^\flat$ and Σ_{stSO} and $G_{\mathbf{t}}$, we define the sequence:

$$(I_0, G_0), (I_1, G_1), \cdots, (I_n, G_n) \cdots$$

where $(I_0, G_0) = (I, G_{F^\flat} \cup G_{\mathbf{t}})$ and for each $i > 0$ instance I_i is obtained from I_{i-1} by applying the grounding $\alpha \to \beta$, where $\beta \neq \bot$. In this case $G_i = G_{i-1} \smallsetminus \{\alpha \to *\}$ and $I_i = I_{i-1} \cup \beta$. In case there exist an integer $i \geq 0$ and a grounding $(\alpha \to \bot) \in G_i$ such that $\alpha \subseteq I_i$, then we say that the *sequence fails.* For a non failing sequence the limit is defined as:

$$(\bigcup_{i \in \omega} I_i, \bigcap_{i \in \omega} G_i).$$

The notation $\mathcal{C}(I, G_{F^\flat} \cup G_{\mathbf{t}})$ stands for the set of the limits of all chase sequences originating from $(I, G_{F^\flat} \cup G_{\mathbf{t}})$. For $\Sigma = \Sigma_{stSO} \cup \Sigma_t$, we define $\Sigma_{F^\flat}(I)$:

$$\Sigma_{F^\flat}(I) = \{J \; : \; (J, G) \in \mathcal{C}\big(I, G_{F^\flat} \cup G_{\mathbf{t}}\big)\}$$

Finally, for Σ and I we define $\Sigma(I)$ to be:

$$\Sigma(I) = \bigcup_{F^\flat \text{ - interpretation}} \Sigma_{F^\flat}(I)$$

The following proposition is a direct result of the fact that st-SO dependencies are non-recursive. For a set of instances $\mathcal{I}$, we define $\Sigma(\mathcal{I}) = \bigcup_{I \in \mathcal{I}} \Sigma(I)$.

Proposition 20 *Let* $\mathbf{S}$ *and* $\mathbf{T}$ *be two distinct schemata and let* Σ_{stSO} *st-SO dependencies and* Σ_t *be a set of target TGD's. If* $\Sigma = \Sigma_{stSO} \cup \Sigma_t$*, then:*

$$\Sigma(I) = \Sigma_t(\Sigma_{stSO}(I)).$$

Let $M = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ be a data exchange mapping, where Σ_{st} is specified by st-SO dependencies, Σ_t is specified by a set of target TGD's. Given a source instance I and by denoting $\Sigma = \Sigma_{st} \cup \Sigma_t$, we extend the notion of constructible models solution space to st-SO dependencies and target TGD's as follows:

$$Sol_M^{\mathtt{CM}}(I) = \{J \in Inst(\mathbf{T}) | \ \exists K \in \Sigma(I) \text{ and } J = K|_{\mathbf{T}}\}.$$

7.3 Global Conditional Tables

We present now an extension of the conditional tables (introduced in Section 6.3.1) called global conditional tables [35]. As we will see in the following section, this extension is needed in order to capture the constructible models semantics over st-SO dependencies.

A global conditional table (global c-table) is a pair (T, φ), where T is a tableau, and φ, beside associating a local condition $\varphi(t)$ with each tuple $t \in T$, it also associates a *global condition* $\varphi(T)$ to (T, φ). Similarly to local conditions, the global condition is a boolean formula built up from atoms of the form $x = y$, $x \neq y$, $x = a$, $x \neq a$, $a = b$ and $a \neq b$ for $x, y \in \Delta_{\mathsf{N}}$, and $a, b \in \Delta_{\mathsf{C}}$. An atomic equality of the form $a = a$ for $a \in \Delta_{\mathsf{C}}$ represents the logical constant **true**, while for two distinct constants a and b, the equality $a = b$ represents **false**.

A global conditional table (T, φ) represents a set of possible worlds. Let v be a valuation such that $v(\varphi(T)) \equiv \mathbf{true}$, then:

$$v(T, \varphi) = \{v(t) | \ t \in T, \ \text{ and } v\big(\varphi(t)\big) \equiv \mathbf{true}\}. \tag{7.2}$$

The set of possible worlds represented by global c-table (T, φ) is:

$$rep_{\mathbf{C}}(T, \varphi) = \bigcup_{v,\ v(\varphi(T)) \equiv \mathbf{true}} \{v(T, \varphi)\}. \tag{7.3}$$

Note that any global conditional table (T, φ) with tautological global condition, that is for all valuation v we have $v(\varphi(T)) \equiv \textbf{true}$, is equivalent to the conditional table (T, φ).

Example 61 *Consider the global conditional table (T, φ) with the tabular representation shown below:*

$\varphi(T) = (X = a \lor X = b \lor X = c)$

t	$\varphi(t)$
$R(a, X)$	**true**
$R(b, c)$	**true**
$R(a, c)$	$X = b$

The set of possible worlds is given by the set of valuations for (T, φ) that makes the global condition a tautology. The tabular representation of all the possible instances for (T, φ), where $I_i = v_i(T, \varphi)$, for $i \in \{1, 2, 3\}$, $v_1(X) = a$, $v_2(X) = b$ and $v_3(X) = c$ are presented below:

I_1	I_2	I_3
$R(a, a)$	$R(a, b)$	$R(a, c)$
$R(b, c)$	$R(b, c)$	$R(b, c)$
	$R(c, d)$	

The partial order "$\sqsubseteq$" is extended to the global conditional tables too. Let (T, φ) and (U, ψ) be global c-tables. By $(T, \varphi) \sqsubseteq (U, \phi)$ we mean that $T \subseteq U$, and that for all valuations v such that $v(\varphi(T)) \equiv \textbf{true}$, it is that $v(\varphi(U)) \equiv \textbf{true}$ and $v(T, \varphi) \subseteq v(U, \phi)$. If both $(T, \varphi) \sqsubseteq (U, \phi)$ and $(T, \varphi) \sqsubseteq (U, \phi)$, we write $(T, \varphi) \cong (U, \phi)$ and say that the global c-tables are *congruent*.

Two global c-tables (T, φ), (U, ϕ) are said to be equivalent, denoted $(T, \varphi) \equiv (U, \phi)$, if $rep_{\mathbf{C}}(T, \varphi) = rep_{\mathbf{C}}(U, \phi)$. Clearly Lemma 8 holds for global conditional tables as well. The class of all global conditional tables is denoted by GCOND.

In the natural way, we extend the notion of conditional table and global conditional table to the notions of Skolem conditional table and Skolem global conditional tables.

Example 62 *This is an example of Skolem global conditional table:*

$\varphi(T) = (X = a \vee f(X,c) = b \vee g(X) = c)$

t	$\varphi(t)$
$R(a, g(X))$	**true**
$R(b, c)$	**true**
$R(a, c)$	$X = b$

*The same as a global c-table, a Skolem global conditional table represents a set of possible worlds. Let (T, φ) be a Skolem global c-table. Let $F^\flat$ be an interpretation for the Skolem functions in (T, φ) and let v be a valuation such that $F^\flat(v(\varphi(T))) \equiv$ **true**, where $F^\flat(\psi)$, for a formula ψ, represents the formula obtained by replacing recursively each function symbol by its interpretation in $F^\flat$. Note that formula $F^\flat(v(\varphi(T)))$ is well defined because $v(\varphi(T))$ does not contain any null symbols. We can now define $v(T, \varphi)$ to be:*

$$F^\flat(v(T,\varphi)) = \{F^\flat(v(t)) \mid t \in T \textit{ and } F^\flat(v(\varphi(t))) \equiv \mathbf{true}\} \tag{7.4}$$

The set of possible worlds represented by Skolem global c-table (T, φ) is:

$$rep_{\mathbf{C}}(T,\varphi) = \bigcup_{F^\flat} \bigcup_{v,\, F^\flat(v(\varphi(T)))\equiv\mathbf{true}} \{F^\flat(v(T,\varphi))\}. \tag{7.5}$$

Let us now present the notion of Skolem term unification. Let $s_1 = f(t^1_1, t^1_2, \ldots, t^1_k)$ and $s_2 = f(t^2_1, t^2_2, \ldots, t^2_k)$ be two Skolem terms over the function symboldefined as f with $arity(f) = k$. Let $A = \{t^1_1, \ldots, t^1_k, t^2_1, \ldots, t^2_k\}$. A mapping θ from A to $A \cup \Delta_{\mathsf{N}}$ is said to be a unifier for s_1 and s_2 if:

1. θ is identity on constants, and
2. for each $\theta(t) = X$, where $X \notin A$, X is a fresh new null value from Δ_{N}, and
3. $\theta(s_1) = \theta(s_2)$.

A unifier θ_1 for s_1 and s_2 is said to be more general than a unifier θ_2, if there exists a non-isomorphic function g, identity on constants, such that $\theta_2 = g \circ \theta_1$. Finally, a unifier θ is said to be a most general unifier for the terms s_1 and s_2 if there does not exist a unifier ψ for s_1, s_2 that is more general than θ. Clearly any two most general unifiers for terms s_1 and s_2 are isomorphic. We denote by $mgu(s_1, s_2)$ a representative unifier from the equivalence class of the most general unifiers for s_1 and s_2. For a most general unifier θ we will use the abbreviation:

$$\wedge\!\!\!\wedge\,(\theta) \;=_{\mathsf{def}}\; \Big(\bigwedge_{t,s\notin\Delta_{\mathsf{C}},\, \theta(t)=\theta(s)} t = s\Big) \wedge \Big(\bigwedge_{a\in\Delta_{\mathsf{C}},\, \theta(t)=a} t = a\Big) \tag{7.6}$$

Example 63 *Let $s_1 = f(a, g(X_1, b), X_2, h(X_2))$ and $s_2 = f(X_1, h(X_2), f(a, X_2, b, X_3), c)$, then $mgu(s_1, s_2) = \{X_1/a, g(X_1, b)/c, h(X_2)/c, X_2/Y, f(a, X_2, b, X_3)/Y\}$, where Y is a fresh new null value. For this we have:*

$$\wedge\!\!\!\wedge(mgu(s_1, s_2)) = (X_2 = f(a, X_2, b, X_3) \wedge X_1 = a \wedge g(X_1, b) = c \wedge h(X_2)/c).$$

Given a Skolem global c-table (T, φ), we denote with $(T{\downarrow_f}, \varphi{\downarrow_f})$ the global conditional table obtained following these steps:

1. create table $TERMS$ that has an entry for each term over a functional symbol in T (i.e. we do not include constants and nulls);
2. construct conjunctive formula ψ with a conjunction $\wedge\!\!\!\wedge(mgu(t_1, t_2)) \rightarrow t_1 = t_2$ for each two unifiable t_1,t_2 in $TERMS$;
3. assign for each term in $TERMS$ a new fresh null value;
4. construct formula $\hat{\psi}$ by replacing each highest level term ψ with its corresponding null value from $TERMS$. Note that this process is not recursive, for example if in $TERMS$ we mapped $g(X_1)$ to Y and $f(a, g(X_1))$ to Z, then the formula $f(a, g(X_1)) = b$ is replaced with $Z = b$;
5. construct global c-table (U, ϕ) by replacing each highest level term in (T, φ) with its corresponding null in $TERMS$;

6. construct $(T{\downarrow_f}, \varphi{\downarrow_f})$ such that $T{\downarrow_f} = U$ and for all $t \in T{\downarrow_f}$ we have $\varphi{\downarrow_f}\,(t) = \phi(t)$ and $\varphi{\downarrow_f}\,(T{\downarrow_f}) = \phi(U) \wedge \hat{\psi}$.

Because $TERMS$ contains all the Skolem terms over the function symbols, it follows that $(T{\downarrow_f}, \varphi{\downarrow_f})$ does not contain any function symbols, thus $(T{\downarrow_f}, \varphi{\downarrow_f})$ is a global c-table.

Example 64 *Let us exemplify the previous construction by considering (T, φ) to be the following Skolemized global c-table:*

$\varphi(T) =$ **true**

t	$\varphi(t)$
$R(a, f(X_1, g(X_2)))$	$g(a) = f(a, b)$
$R(b, X_1)$	$f(a, f(a, b)) = c$
$R(f(a, g(X_3)), c)$	$X = b$

Under this setting the table $TERMS$ will contain the following set of terms:

$$\{f(X_1, g(X_2)), f(a, g(X_3)), f(a, f(a, b)), f(a, b), g(X_2), g(X_3), g(a)\}$$

Formula ψ obtained by unifying each term from $TERMS$ will be:

$$\begin{aligned}
\psi = {} & (X_1 = a \wedge g(X_2) = g(X_3) \rightarrow f(X_1, g(X_2)) = f(a, g(X_3))) \\
& \wedge(X_1 = a \wedge g(X_2) = f(a, b) \rightarrow f(X_1, g(X_2)) = f(a, f(a, b))) \\
& \wedge(X_1 = a \wedge g(X_2) = b \rightarrow f(X_1, g(X_2)) = f(a, b)) \\
& \wedge(g(X_3) = f(a, b) \rightarrow f(a, g(X_3)) = f(a, f(a, b))) \\
& \wedge(g(X_3) = b \rightarrow f(a, g(X_3)) = f(a, b)) \\
& \wedge(f(a, b) = b \rightarrow f(a, f(a, b)) = f(a, b)) \\
& \wedge(X_2 = X_3 \rightarrow g(X_2) = g(X_3)) \\
& \wedge(X_2 = a \rightarrow g(X_2) = g(a)) \\
& \wedge(X_3 = a \rightarrow g(X_3) = g(a))
\end{aligned}$$

Next we assign a fresh new variable for each term in terms as follows:

$TERMS$	
$f(X_1, g(X_2))$	Y_1
$f(a, g(X_3))$	Y_2
$f(a, f(a,b))$	Y_3
$f(a,b)$	Y_4
$g(X_2)$	Y_5
$g(X_3)$	Y_6
$g(a)$	Y_7

Due to these notations, the formula ψ *is transformed in* $\hat{\psi}$*:*

$$\begin{aligned}\psi = {} & (X_1 = a \wedge Y_5 = Y_6 \rightarrow Y_1 = Y_2) \wedge (X_1 = a \wedge Y_5 = Y_4) \rightarrow Y_1 = Y_3) \\ & \wedge (X_1 = a \wedge Y_5 = b \rightarrow Y_1 = Y_4) \wedge (Y_6 = Y_4 \rightarrow Y_2 = Y_3) \\ & \wedge (Y_6 = b \rightarrow Y_2 = Y_4) \wedge (Y_4 = b \rightarrow Y_3 = Y_4) \\ & \wedge (X_2 = X_3 \rightarrow Y_5 = Y_6) \wedge (X_2 = a \rightarrow Y_5 = Y_7) \wedge (X_3 = a \rightarrow Y_6 = Y_7)\end{aligned}$$

Thus, the global c-table $(T\downarrow_f, \varphi\downarrow_f)$ *is:*

$\varphi\downarrow_f (T\downarrow_f) = \hat{\psi}$	
t	$\varphi\downarrow_f (t)$
$R(a, Y_1)$	$Y_7 = Y_4$
$R(b, X_1)$	$Y_3 = c$
$R(Y_2, c)$	$X = b$

The following theorem tells us that by removing the function symbols as presented before, we do not loose any information.

Theorem 57 *Given* (T, φ) *a Skolemized global c-table, then*

$$rep_{\mathbf{C}}(T, \varphi) = rep_{\mathbf{C}}(T\downarrow_f, \varphi\downarrow_f).$$

Proof: Let $I \in rep_{\mathbf{C}}(T, \varphi)$. This means there exists an interpretation $F^\flat$ for the function symbols in (T, φ) and the valuation v such that $F^\flat(v(\varphi(T))) \equiv \textbf{true}$ and $I = F^\flat(v(T, \varphi))$. Let v' be the valuation that extends v as follows: for each term t

in $TERMS$ associated with the null value Y we set $v'(Y) = F^\flat(v(t))$. Let us also define $\varphi{\downarrow_f}(T) = \phi(U) \wedge \hat{\psi}$ (from the description of $(T{\downarrow_f}, \varphi{\downarrow_f})$). Now, from the used construction, we have $v'(\phi(U)) = F^\flat(\varphi(T))$, thus $v'(\phi(U)) \equiv \mathbf{true}$. Let us consider a conjunction $\alpha \rightarrow Y_i = Y_j$ from $\hat{\psi}$. Let us now replace Y_i and Y_j with the corresponding terms t_i and t_j from $TERMS$. Thus, the formula becomes $\bigwedge\!\!\!\bigwedge(mgu(t_i,t_j)) \rightarrow t_i = t_j$. But, from the construction of $\bigwedge\!\!\!\bigwedge(mgu(t_i,t_j))$, it naturally follows that for any mapping g such that $g(\bigwedge\!\!\!\bigwedge(mgu(t_i,t_j))) \equiv \mathbf{true}$, it is that $g(t_i) = g(t_j)$. On the other hand, if $v(\bigwedge\!\!\!\bigwedge(mgu(t_i,t_j))) \equiv \mathbf{true}$, from the construction of v' we have that $v'(\alpha) \equiv \mathbf{true}$, and also because $v(t_i) = v(t_j)$ and because $F^\flat$ is a well defined interpretation it follows that $v'(Y_i) = v'(Y_j)$. Thus, $v'(\alpha \rightarrow Y_i = Y_j) \equiv \mathbf{true}$, meaning that $v'(\varphi{\downarrow_f}(T{\downarrow_f})) \equiv \mathbf{true}$.

Now let $t \in T$. From the construction of $(T{\downarrow_f}, \varphi{\downarrow_f})$ it follows that there exists a tuple $t' \in T{\downarrow_f}$ obtained from t by replacing each function symbols terms with their corresponding null from $TERMS$ table. Clearly, from the construction of v', we have that $F^\flat(v(t)) = v'(t')$ and also that $F^\flat(v(\varphi(t))) = v'(\varphi{\downarrow_f}(t'))$. From this it directly follows that $F^\flat(v(T,\varphi)) \subseteq v'(T{\downarrow_f}, \varphi{\downarrow_f})$. Thus, by generalization, $rep_{\mathbf{C}}(T,\varphi) \subseteq rep_{\mathbf{C}}(T{\downarrow_f}, \varphi{\downarrow_f})$.

For the other direction, let us consider $I \in rep_{\mathbf{C}}(T{\downarrow_f}, \varphi{\downarrow_f})$. This means that there exists a valuation v' such that $v'(\varphi{\downarrow_f}(T{\downarrow_f})) \equiv \mathbf{true}$. Let us set v to be the restriction of v' to all null values except the ones from $TERMS$. And let us define the interpretation $F^\flat$ such that for each term t_i in $TERMS$ with the associated labeled null Y_i set $F^\flat(v(t_i)) = v'(Y_i)$. Because $v'(\hat{\psi}) \equiv \mathbf{true}$, it follows that $F^\flat$ is a well defined interpretation. From this it follows, similarly to the proof for the other direction, that $v'(T{\downarrow_f}, \varphi{\downarrow_f}) = F^\flat(v(T,\varphi))$ and, by generalization, that $rep_{\mathbf{C}}(T{\downarrow_f}, \varphi{\downarrow_f}) \subseteq rep_{\mathbf{C}}(T,\varphi)$■

7.4 Chasing with Second Order Dependencies

In this section we will extend the conditional table chase with TGD's to the global-conditional chase with st-SO dependencies. To achieve this, first we need to extend the tableau unification process to Skolem tableaux. For a Skolem tableau T, by $\hat{T}$ we denote the tableau T restricted to all relational symbols except E. We denote by $\Delta_{\mathsf{F}}(T)$ the set of all terms in T. The homomorphism notion is extended in the natural way to the Skolem tableaux too.

Definition 51 *Let T and U be two Skolem tableaux such that only T may contain the equality symbol E. A* unifier *for T and U, if it exists, is a pair (θ_1, θ_2), where θ_1 is a homomorphism from the set of terms $\Delta_{\mathsf{F}}(T)$ to the set $\Delta_{\mathsf{F}}(U) \cup \Delta_{\mathsf{C}}(dom(T))$ and θ_2 is a $\Delta_{\mathsf{C}}(dom(T))$-retraction for U, such that the following holds $\theta_1(\hat{T}) = \theta_2(U)$.*

Example 65 *Consider tableau $T = \{R(a, f(X_1), f(X_1), X_2), E(a, f(X_1))\}$ and tableau $U = \{R(Y_1, b, g(Y_2), Y_3)\}$. The pair (θ_1, θ_2), with the mappings $\theta_1 = \{f(X_1)/b, X_2/c\}$ and $\theta_2 = \{Y_1/a, g(Y_2)/b, Y_3/c\}$, is a unifier for T and U. Another unifier for T and U is (θ_1', θ_2'), where $\theta_1' = \{f(X_1)/b, X_2/Y_3\}$ and $\theta_2 = \{Y_1/a, g(Y_2)/b\}$.*

Definition 52 *A unifier (θ_1, θ_2) for the Skolem tableaux T and U is* more general *than a unifier (γ_1, γ_2), if there is a mapping f on $dom(U)$, $f \notin \mathsf{Id}$, identity on constants, such that $\gamma_1 = f \circ \theta_1$ and $\gamma_2 = f \circ \theta_2$.*

The more general unifier definition is extended to the most general unifier as follows:

Definition 53 *A unifier (θ_1, θ_2) is a* most general unifier *(mgu) for the Skolem instances T and U, if all unifiers (γ_1, γ_2) of T and U that are more general than (θ_1, θ_2) actually are isomorphic with (θ_1, θ_2). We denote by $mgu(T, U)$ the set of (representatives of the) equivalence classes of all mgu's of T and U.*

The same as in the standard-conditional chase case, we need to introduce the notion of Skolemized conditional trigger.

Definition 54 *Let Σ be st-SO dependencies of the form: $\exists \bar{f}\ \xi_1, \ldots, \xi_n$, where each ξ_i are of the form $\forall \bar{x}_i\ \alpha_i \to \beta_i$, for $i \in [n]$. Let T be a Skolem instance. A* Skolem conditional trigger for Σ on T *(or simply a trigger) is a tuple $\tau = (\xi_i, \theta, T')$, where $i \in [n]$ and $\theta = (\theta_1, \theta_2)$ is a mgu that unifies body(ξ_i) with T', where $T' \subseteq T$, and T' is a θ_2-core. The set trigg$_\Sigma(T)$ contains the set of all triggers for Σ on T.*

Example 66 *Going back to st-SO dependencies, from Example 54:*

$$\exists f \big(\forall e\ (Emp(e) \to Mgr(e, f(e))) \land \forall e\ (Emp(e) \land (e = f(e)) \to SelfMgr(e)) \big)$$

Let $\xi_1 = \forall e\ (Emp(e) \to Mgr(e, f(e)))$ and $\xi_2 = \forall e\ (Emp(e) \land (e = f(e)) \to SelfMgr(e))$. Consider a tableau $T = \{Emp(john), Emp(mike)\}$. For this configuration we have the following set of triggers:

$$\begin{aligned}
\tau_1 &= (\xi_1, (\{e/john\}, \{\}), \{Emp(john)\}) \\
\tau_2 &= (\xi_2, (\{e/john\}, \{\}), \{Emp(john)\}) \\
\tau_1 &= (\xi_1, (\{e/mike\}, \{\}), \{Emp(mike)\}) \\
\tau_2 &= (\xi_2, (\{e/mike\}, \{\}), \{Emp(mike)\})
\end{aligned}$$

Note that when dealing with st-SO dependencies there are no existentially quantified variables, so there is no need to create new null values during the chase process. It can also be observed that the presence of the Skolem functions ease the chase process as there is no need anymore to compute the core for the conditional table.

Before introducing the Skolem-conditional chase micro-step, let us introduce some notations similar to the ones presented for the conditional-chase process introduced in Section 6.4. Let (T, φ) be a Skolem conditional table. Then we define:

$$\mathop{\wedge\!\!\!\wedge}(T,\varphi) \;=_{\mathsf{def}}\; \bigwedge_{t\in T}\varphi(t), \tag{7.7}$$

When θ is a finite partial mapping from Δ_{N} to the set $\Delta_{\mathsf{N}} \cup \Delta_{\mathsf{C}}$, we shall use the abbreviation:

$$\mathop{\wedge\!\!\!\wedge}(\theta) \;=_{\mathsf{def}}\; \bigwedge_{i} X_i = \theta(X_i), \tag{7.8}$$

where the X_i's are all the nulls in the domain of θ.

For a Skolem trigger $\tau = (\xi, (\theta_1, \theta_2), T')$ we define:

$$\mathop{\wedge\!\!\!\wedge}_{\mathbf{L}}(\tau) \;=_{\mathsf{def}}\; \bigwedge_{E(t_1,t_2)\in body(\xi)} \theta_1(t_1) = \theta_1(t_2), \tag{7.9}$$

$$\mathop{\wedge\!\!\!\wedge}_{\mathbf{G}}(\tau) \;=_{\mathsf{def}}\; \bigwedge_{E(t_1,t_2)\in head(\xi)} \mathop{\wedge\!\!\!\wedge}(\theta_2) \wedge \mathop{\wedge\!\!\!\wedge}(T',\varphi) \wedge \mathop{\wedge\!\!\!\wedge}_{\mathbf{L}}(\tau) \to \theta_1(t_1) = \theta_1(t_2). \tag{7.10}$$

These new notations are now useful to define the conditional-chase step on the Skolem conditional tables:

Definition 55 *Let (T,φ) be a Skolem conditional table. Let $\tau = (\xi, (\theta_1, \theta_2), T')$ be a trigger from $trigg_\Sigma(T)$. We say that the Skolem conditional table (U,ϕ)* is obtained from (T,φ) by applying trigger τ, *if (U,ϕ) contains all the c-tuples from T together with their possibly modified local and global conditions, and possibly a new c-tuple, as follows:*

If *T contains a tuple t syntactically equal to $\theta_1(head(\xi))$,*

then *for (U, ϕ) the local condition of t is changed to:*

$$\phi(t) =_{\mathsf{def}} \varphi(t) \vee \big(\mathbin{\wedge\!\!\!\wedge}(T', \varphi) \wedge \mathbin{\wedge\!\!\!\wedge}(\theta_2) \wedge \mathbin{\wedge\!\!\!\wedge}_{\mathbf{L}}(\tau)\big), \tag{7.11}$$

else *add the tuple $t' : \theta_1(head(\xi))$ with local condition:*

$$\phi(t') =_{\mathsf{def}} \mathbin{\wedge\!\!\!\wedge}(T', \varphi) \wedge \mathbin{\wedge\!\!\!\wedge}(\theta_2) \wedge \mathbin{\wedge\!\!\!\wedge}_{\mathbf{L}}(\tau). \tag{7.12}$$

If (U, ϕ) is obtained from (T, φ) in this way and if $\boxed{\phi(U) = \varphi(T) \wedge \mathbin{\wedge\!\!\!\wedge}_{\mathbf{G}}(\tau)}$, then we write $(T, \varphi) \rightarrow_\tau (U, \phi)$. This transformation is called a Skolem-conditional chase micro-step *(or simply micro-step).*

We are now ready to define the result of chasing a Skolem conditional table with st-SO dependencies;

Definition 56 *Let (T, φ) be a Skolem conditional table Σ st-SO dependencies from schema $\mathbf{S}$ to schema $\mathbf{T}$. Let also $trigg_\Sigma(T) = \{\tau_1, \tau_2, \ldots, \tau_n\}$. The sequence:*

$$(T_0, \varphi_0), (T_1, \varphi_1), \cdots, (T_n, \varphi_n)$$

is said to be a Second-Order chase sequence *if $(T_0, \varphi_0) = (T, \varphi)$ and for all $i \in [n]$ it holds that $(T_{i-1}, \varphi_{i-1}) \rightarrow_{\tau_i} (T_i, \varphi_i)$.*

The following lemma follows directly from Definition 55 and the observation that st-SO dependencies are non-recursive dependencies.

Lemma 19 *Let (T, φ) be a Skolem conditional table and Σ st-SO dependencies, such that $|trigg_\Sigma(T)| = n$. Let also (T_n, φ_n) and (T'_n, φ'_n) be two Skolem c-tables from the n-th position in two Second Order chase sequences for (T, φ) with Σ, then:*

$$rep_{\mathbf{C}}(T_n, \varphi_n) = rep_{\mathbf{C}}(T'_n, \varphi'_n)$$

Intuitively, the previous lemma states that the order of applying the triggers when the dependencies are specified as st-SO dependencies does not matter.

The Skolem conditional table (T_n, φ_n) from Definition 56 is called the result of chasing (T, φ) with Σ and it is denoted by $chase^{\mathbf{cond}}_{\Sigma}(T, \varphi)$.

The following theorem shows that the interpretation of the Skolem conditional table gives exactly the set of constructible models for st-SO dependencies and the interpretation of the initial Skolem conditional table.

Theorem 58 *If (T, φ) is a Skolem conditional table and Σ st-SO dependencies such that (T, φ) and Σ does not share any skolem function names, then:*

$$\Sigma(rep_{\mathbf{C}}(T, \varphi)) = rep_{\mathbf{C}}(chase^{\mathbf{cond}}_{\Sigma}(T, \varphi))$$

Proof: Let $I \in \Sigma(rep_{\mathbf{C}}(T, \varphi))$, that is there exists an interpretation $F^{\flat}$ for the function symbols in (T, φ) and there exists a valuation v such that $J = F^{\flat}(v(T, \varphi))$ and $I \in \Sigma(J)$. This means that there exists an interpretation $K^{\flat}$ for the function symbols in Σ and that there exists a non failing constructible models sequence:

$$(J_0, G_0), (J_1, G_1), \cdots, (J_m, G_m)$$

such that $(J_0, G_0) = (J, G_{K^{\flat}})$ and for all $i \in [m]$ the instance J_i is obtained from the instance J_{i-1} by applying a grounding from G_{i-1}. There are no groundings in G_m to be applied on J_m and $J_m = I$.

We will prove by induction that there is a sequence of Skolem c-tables:

$$(T_0, \varphi_0), (T_1, \varphi_1), \cdots, (T_m, \varphi_m)$$

such that $(T_0, \varphi_0) = (T, \varphi)$ and for each $i \in [m]$, $K^\flat(F^\flat(v(T_i, \varphi_i))) = J_i$.

For the base case, let us consider $i = 0$, from the hypothesis $J = J_0 = F^\flat(v(T_0, \varphi_0))$. But since $(T_0, \varphi_0) = (T, \varphi)$ and because there are no common Skolem functions in Σ and (T, φ), it follows that $J_0 = K^\flat(F^\flat(v(T_0, \varphi_0)))$, proving the base case.

For the induction step, let us consider that for any integer $i \in [k-1]$, for a $k \in [m]$, the induction assumption holds. We need to prove that $K^\flat(F^\flat(v(T_k, \varphi_k))) = J_k$.

Now, let $(J_{k-1}, G_{k-1}) \Rightarrow_{\alpha \to \beta} (J_k, G_k)$. Thus $(\alpha \to \beta) \in \mathit{ground}(\xi)$, $\alpha \subseteq J_{k-1}$ and $J_k = J_{k-1} \cup \beta$. Because $\alpha \subseteq J_{k-1}$ and from the induction assumption we have that $J_{k-1} = K^\flat(F^\flat(v(T_{k-1}, \varphi_{k-1})))$. Therefore, there exists a subset minimal $T' \subseteq T_{k-1}$ such that $\alpha = K^\flat(F^\flat(v(T', \varphi_{k-1})))$. Moreover, α is $K^\flat(v'(\mathit{body}(\xi)))$ restricted to all relation names except the E relation, where v' is a valuation for the nulls in $v'(\mathit{body}(\xi))$. The consequence is that $(K^\flat \circ v', K^\flat \circ F^\flat \circ v)$ is a unifier for $\mathit{body}(\xi)$ and tableau T'. Thus, there exists a most general unifier (θ_1, θ_2) such that $K^\flat \circ v' = g \circ \theta_1$ and $K^\flat \circ F^\flat \circ v = g \circ \theta_2$. That is $\tau_k = (\xi, (\theta_1, \theta_2), T')$ is a trigger in $\mathit{trigg}_\Sigma(T_{k-1})$. On the other hand, because Σ is a source-to-target set of dependencies, it follows that $\tau_k \in \mathit{trigg}_\Sigma(T)$. Based on Definition 55, we create (T_k, φ_k) such that $(T_{k-1}, \varphi_{k-1}) \to_{\tau_k} (T_k, \varphi_k)$. By the micro-step definition, we have $\varphi_k(T_k) = \varphi_{k-1}(T_{k-1}) \wedge \curlywedge_{\mathbf{G}}(\tau_k)$ and by the induction step we have $K^\flat(F^\flat(v(\varphi_{k-1}(T_{k-1})))) \equiv \textbf{true}$. Towards a contradiction, let us suppose that $K^\flat(F^\flat(v(\curlywedge_{\mathbf{G}}(\tau_k)))) \equiv \textbf{false}$, thus there exist terms t_1 and t_2 that are equated in the head of ξ, such that $K^\flat(F^\flat(v(\theta_1(t_1)))) \neq K^\flat(F^\flat(v(\theta_1(t_2))))$, because the local conditions and unifications are evaluated to **true**. But this cannot be true as, on the contrary, it will follow that $(\alpha \to \bot) \in G_{k-1}$. Thus, the sequence would be a failing one. This means that it must be that $K^\flat(F^\flat(v(\varphi_k(T_k)))) \equiv \textbf{true}$. Similarly, because, from the induction assumption, $\alpha = K^\flat(F^\flat(v(T', \varphi_{k-1})))$, it follows that the local condition for the tuples added/modified needs to be valuated to **true** by $K^\flat \circ F^\flat \circ v$. Following that, $K^\flat(F^\flat(v(T_k, \varphi_k))) = K^\flat(F^\flat(v(T_{k-1}, \varphi_{k-1}))) \cup \beta$. Thus $K^\flat(F^\flat(v(T_k, \varphi_k))) = J_k$, proving the induction step.

With this we demonstrated that for the constructible models sequence:

$$(J_0, G_0), (J_1, G_1), \cdots, (J_m, G_m)$$

there exists a Second Order conditional-chase sequence:

$$(T_0, \varphi_0), (T_1, \varphi_1), \cdots, (T_m, \varphi_m)$$

such that $K^\flat(F^\flat(v(T_i, \varphi_i))) = J_i$ for all $i \in [m]$. We will now have to prove that there is no trigger $\tau \in \mathit{trigg}_\Sigma(T)$ such that $\tau \notin \{\tau_1, \ldots, \tau_m\}$, with $(T_m, \varphi_m) \rightarrow_\tau (T_{m+1}, \varphi_{m+1})$ and $K^\flat(F^\flat(v(T_m, \varphi_m))) \neq K^\flat(F^\flat(v(T_{m+1}, \varphi_{m+1})))$. Again, towards a contradiction, let us suppose that such a trigger exists. Let us define $\tau = (\xi, (\theta_1, \theta_2), T')$ and let us also denote $\alpha = K^\flat(F^\flat(v(\mathit{body}(\xi))))$ and $\beta = K^\flat(F^\flat(v(\mathit{head}(\xi))))$. On the other hand, because $K^\flat(F^\flat(v(T_m, \varphi_m))) \neq K^\flat(F^\flat(v(T_{m+1}, \varphi_{m+1})))$ and from the monotonicity property of the micro-chase step, it follows that $K^\flat(F^\flat(v(\varphi_{m+1}(T_{m+1})))) \equiv \mathbf{true}$. This means, from the construction of the global condition, that $\beta \neq \bot$. From this, it directly results that $(\alpha \rightarrow \beta) \in \mathit{trigg}_\Sigma(T)$. Finally, because $\tau \notin \{\tau_1, \ldots, \tau_m\}$ and Σ are source-to-target dependencies, it follows that $\alpha \rightarrow \beta'$ was not applied on the constructible models sequence, that is $\alpha \rightarrow \beta \in G_m$. In other words, it is applicable on (J_m, G_m) and $(J_m, G_m) \Rightarrow_{\alpha \rightarrow \beta} (J_{m+1}, G_{m+1})$ with $J_m \neq J_{m+1}$. This gives a contradiction with the assumption that in the constructible models there are no other applicable groundings on (J_m, G_m). Thus, it needs to be that there is no such trigger τ. This means that the instance I is part of $\mathit{rep}_\mathbf{C}(\mathit{chase}_\Sigma^{\mathbf{cond}}(T, \varphi))$ and, by generalization, that $\Sigma(\mathit{rep}_\mathbf{C}(T, \varphi)) \subseteq \mathit{rep}_\mathbf{C}(\mathit{chase}_\Sigma^{\mathbf{cond}}(T, \varphi))$.

For the other direction, let us suppose $I \in \mathit{rep}_\mathbf{C}(\mathit{chase}_\Sigma^{\mathbf{cond}}(T, \varphi))$. There exists the interpretation $H^\flat$ of the function symbols in $\mathit{chase}_\Sigma^{\mathbf{cond}}(T, \varphi)$ and the valuation v such that $I = H^\flat(v(\mathit{chase}_\Sigma^{\mathbf{cond}}(T, \varphi)))$. Let $F^\flat$ be the interpretation $H^\flat$ restricted to the function symbols form (T, φ) and $K^\flat$ the interpretation $H^\flat$ restricted to the function symbols from Σ. Clearly, $H^\flat = F^\flat \sqcup K^\flat$. Let $trigg_\Sigma(T) = \{\tau_1, \ldots, \tau_m\}$ and consider the sequence:

$$(T_0, \varphi_0), (T_1, \varphi_1), \cdots, (T_m, \varphi_m)$$

where $(T_{i-1}, \varphi_{i-1}) \rightarrow_{\tau_i} (T_i, \varphi_i)$, for all $i \in [m]$. Lemma 19 stipulates directly that $chase_\Sigma^{\mathbf{cond}}(T, \varphi) = (T_m, \varphi_m)$. Consider now the instance $J = F^\flat(v(T, \varphi))$. We will show that $I \in \Sigma_{K^\flat}(J)$. Similarly to the previous case, we can prove by induction that there exists a constructible models sequence:

$$(J_0, G_0), (J_1, G_1), \cdots, (J_m, G_m)$$

such that $(J_0, G_0) = (J, G_{K^\flat})$ with $J_i = H^\flat(v(T_i, \varphi_i))$ and that there are no other applicable groundings $(\alpha \rightarrow \beta) \in G_m$ such that $(J_m, G_m) \Rightarrow_{\alpha \rightarrow \beta} (J_{m+1}, G_{m+1})$ with $J_m \neq J_{m+1}$. Meaning that $J_m \in \Sigma_{K^\flat}(J)$ and $J_m = H^\flat(v(T_m, \varphi_m))$. The consequence is that $J_m = H^\flat(v(chase_\Sigma^{\mathbf{cond}}(T, \varphi))) = I$, so $I \in \Sigma_{K^\flat}(J)$ and, by generalization, $rep_{\mathbf{C}}(chase_\Sigma^{\mathbf{cond}}(T, \varphi)) \subseteq \Sigma(rep_{\mathbf{C}}(T, \varphi))$■

7.5 Strong Representation Systems

The notion of representation systems started lately to get more attention mainly in concordance with data exchange mappings [10]. Before presenting the notion of strong representation system, let us introduce some useful notions. In Section 2.2 we presented the certain answers and possible answers in the context of incomplete databases. We extend these notions with the exact answer to a query q for an incomplete database $\mathcal{I}$, denoted by $q(\mathcal{I})$, to be the set $\bigcup_{I \in \mathcal{I}} \{q(I)\}$.

Now we are ready to define the strong representation systems.

Definition 57 *Let $\mathcal{T}$ be a class of tables, REP an interpretation and $\mathcal{C}$ a class of queries. Then the triple $(\mathcal{T}, REP, \mathcal{Q})$ is a* strong representation system, *if, for each $T \in \mathcal{T}$ and $q \in \mathcal{Q}$, there exists a table $U \in \mathcal{T}$ such that $REP(U) = Rep(T)$.*

Example 67 *Consider G to be the class of ground instances and FO the class of safe first order queries. Then $(\mathsf{G}, rep^{CWA}, \mathsf{FO})$ is a strong representation system. That is for any ground instance I and for any safe query $q \in \mathsf{FO}$ there exists a ground instance J such that $rep^{CWA}(J) = q(rep^{CWA}(I))$. As we know that $rep^{CWA}(I) = \{I\}$ for any ground instance I, the previous formula becomes $\{J\} = q(\{I\})$ or, equivalently, $J = q(I)$. Let us consider the query $q(x) \leftarrow R(x,y), \neg S(y)$ and the ground instance $I = \{R(a,b), R(c,d), S(d)\}$. In this case $q(I)$ can be represented by ground instance $J = \{q(a)\}$. Note that $(\mathsf{G}, rep^{OWA}, \mathsf{FO})$ is also a strong representation system. On the other hand, if we denote by I the class of instances (not necessarily ground), then $(\mathsf{I}, rep^{CWA}, \mathsf{FO})$ is not a strong representation system. Even if we only consider the class of CQ queries, $(\mathsf{I}, rep^{CWA}, \mathsf{CQ})$ is not a strong representation system. To show this, consider instance $I = \{R(a,X)\}$ and query $x \leftarrow R(x,y), R(y,x)$. In this case, I is the set containing either the empty instance or the instance containing tuple (a) (in case $X = a$). Clearly this set cannot be represented by using a single instance.*

It is well known [1; 49] that there are first order queries q and instances I such that no instance J with $rep^{CWA}(J) = q(rep^{CWA}(I))$ exists (see the second query from the previous example). It turns out that we need *conditional tables* to obtain a strong representation system for first order queries.

Imielinski and Lipski [49] showed that conditional tables under closed world interpretation are a strong representation for FO-queries. We denote the class of conditional tables as COND .

Theorem 59 [49] *$(\mathsf{COND}, rep_{\mathbf{C}}, \mathsf{FO})$ is a strong representation system.*

Before introducing the representation systems for data exchange, let us recall the definition of a data exchange mapping, as specified in Section 5.1.2. A mapping M from a source schema $\mathbf{S}$ to a distinct target schema $\mathbf{T}$ is defined as a set of pairs (I, J), where $I \in Inst(\mathbf{S})$ and $J \in Inst(\mathbf{T})$. With this definition the notion of data exchange solution set can be presented as $Sol_M(I) = \{J| \, (I, J) \in M\}$. In the previous chapters we considered that the data exchange mappings are specified by a set of TGD's. Unfortunately, as shown in the following example, not all mappings can be specified by TGD's or/and EGD's.

Example 68 *Consider the source schema $\mathbf{S} = \{V, E\}$, where V represents the set of vertexes in a graph and E is a binary relation representing the set of edges in a graph. The schema $\mathbf{T} = \{C\}$ contains only the unary relation C representing a set of colors. Then we define mapping M from $\mathbf{S}$ to $\mathbf{T}$ as: $(I, J) \in M$, iff the vertexes of the graph represented by I can be colored with only using colors from J in such a way that each adjacent vertexes are colored with a different color. It is well known that the problem of testing if a graph is k-colorable for a fixed number k, with $k \geq 3$, is an* NP*-complete problem* [32]*. On the other hand, it is easy to verify that the membership problem of mappings specified by a set of TGD's and EGD's is polynomial. From this it follows that there are mappings that cannot be specified by a set of TGD's and EGD's.*

In Example 55 we showed a reduction of the graph 3-colorability problem to the membership problem for st-SO dependencies.

Definition 58 *Let $\mathcal{T}$ be a class of tables, REP a function that assigns a set of possible worlds to each $T \in \mathcal{T}$. Let $\mathcal{M}$ be a class of mappings and $\mathcal{Q}$ a class of queries. Then a quadruple $(\mathcal{T}, REP, \mathcal{Q}, \mathcal{M})$ is a* strong data exchange system*, if for each $T \in \mathcal{T}$, $M \in \mathcal{M}$, and $q \in \mathcal{Q}$, there exists a table $U \in \mathcal{T}$ such that:*

$$Rep(U) = \{q(J) |\ J \in Sol_M(I),\ I \in Rep(T)\}.$$

Let us denote by $\mathsf{fullTGD}^{CM}$ to be the class of mappings defined as follows: $M \in \mathsf{fullTGD}^{CM}$ if there exists Σ as a set of source-to-target and target full TGD's and $(I, J) \in M$ iff $J \in Sol^{\mathtt{CM}}_{\Sigma}(I)$. With this definition we have our first strong data exchange system:

Theorem 60 *$(\mathsf{G}, rep^{CWA}, \mathsf{FO}, \mathsf{fullTGD}^{\mathsf{CM}})$ is a strong data exchange system.*

Proof: Let I be a ground instance, that is $I \in \mathsf{G}$ and Σ a set of full source-to-target and target TGD's and EGD's. It is obvious that, for $Sol^{\mathtt{CM}}_{\Sigma}(I)$, the constructible models chase tree is isomorphic with the execution tree for the standard-chase algorithm when it runs with I and the set of full TGD's and EGD's Σ. In Section 3.1.2 we showed that the standard-chase algorithm returns the same instance on every execution branch. From this it follows that the set $Sol^{\mathtt{CM}}_{\Sigma}(I)$ will contain exactly one ground instance. The result now follows directly from this and the strong representation system presented in Example 67■

Unfortunately, the previous theorem does not hold if the mappings are specified by TGD's that contain existentially quantified variables. The same theorem will not work either if we change the class G by the class of general instances.

Let us denote TGD^{CM} to be the class of mappings defined as: $M \in \mathsf{TGD}^{CM}$, iff there exists Σ as a set of source-to-target and target TGD's and $(I, J) \in M$, iff $J \in Sol^{\mathtt{CM}}_{\Sigma}(I)$. With this definition from Corollary 10 we can infer the following result:

Theorem 61 *$(\mathsf{COND}, rep_{\mathbf{C}}, \mathsf{FO}, \mathsf{TGD}^{\mathsf{CM}})$ is a strong data exchange system.*

The following theorem extends the previous result to the class ordSO, where mapping $M \in \mathsf{ordSO}$ if the dependencies in M can be specified by st-SO dependencies and the target richly acyclic TGD's and $(I, J) \in M$, iff $J \in Sol^{\mathtt{CM}}_{\Sigma}(I)$.

Theorem 62 $(\mathsf{GCOND}, rep_{\mathbf{C}}, \mathsf{FO}, \mathsf{ordSO})$ *is a strong data exchange system.*

Proof: Let (T, φ) be a global conditional table and $M \in \mathsf{ordSO}$, thus $M = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$, where Σ_{st} are st-SO dependencies and Σ_t is specified by a richly acyclic set of TGD's. By Σ we denote the set $\Sigma_{st} \cup \Sigma_t$. Let (U, ϕ) be the Skolemized c-table $chase^{\mathbf{cond}}_{\Sigma_{st}}(T, \varphi)$. From Theorem 58 we have that $\Sigma_{st}(rep_{\mathbf{C}}(T, \varphi)) = rep_{\mathbf{C}}(U, \phi)$. On the other hand, we know from Theorem 57 that there exists a global conditional table $(U{\downarrow_f}, \phi{\downarrow_f})$ such that $rep_{\mathbf{C}}(U, \phi) = rep_{\mathbf{C}}(U{\downarrow_f}, \phi{\downarrow_f})$. Thus, we have $\Sigma_{st}(rep_{\mathbf{C}}(T, \varphi)) = rep_{\mathbf{C}}(U{\downarrow_f}, \phi{\downarrow_f})$ (1). From Theorem 56 we know that, because Σ_t is a richly acyclic set of TGD's, the conditional chase $chase^{\mathbf{cond}}_{\Sigma_t}(U{\downarrow_f}, \phi{\downarrow_f})$ will terminate. Let us now denote by (V, ψ) the global conditional table resulting by this chase procedure. Theorem 55 tells us that $\Sigma_t(rep_{\mathbf{C}}(U{\downarrow_f}, \phi{\downarrow_f})) = rep_{\mathbf{C}}(V, \psi)$ (2). Finally, from Proposition 20 we have $\Sigma(rep_{\mathbf{C}}(T, \varphi)) = \Sigma_t(\Sigma_{st}(rep_{\mathbf{C}}(T, \varphi)))$. From this and (1), it obviously follows that $\Sigma(rep_{\mathbf{C}}(T, \varphi)) = \Sigma_t(rep_{\mathbf{C}}(U{\downarrow_f}, \phi{\downarrow_f}))$. At the same time, from this and (2) we have $\Sigma(rep_{\mathbf{C}}(T, \varphi)) = rep_{\mathbf{C}}(V, \psi)$ (3). By restricting the instances from (3) to schema $\mathbf{T}$, it follows the statement from the theorem$\blacksquare$

We may say now, in conclusion, that the main focus of this section was the introduction of a large class of dependencies which are closed under data exchange. To achieve this, we showed, as it can be seen from the previous theorem, that the class ordSO, the class of first order queries together with the global conditional tables represent a strong data exchange system.

Chapter 8

Conclusions

The chase based procedures gained a lot of attention lately due to its applicability in newly introduced problems: data integration, ontologies, data repairs, data exchange, peer data exchange and data correspondence. Based on the studied research problem, there were different variations of the chase procedure. A first contribution of this thesis is to align together and compare the main chase variations: standard, oblivious, semi-oblivious, restricted and core chase. As presented, for all these variations when run with the same input values, if they terminate, the resulting instances are homomorphically equivalent, so it can be used to get the certain answers to any query in the UCQ class. We also extended the existing results by making a clear delimitation between the chase variations based on their termination. Thus, for each chase variation four classes of dependencies were identified:

- $\mathsf{CT}_{\forall\forall}$ - the class of dependencies for which the chase algorithm terminates on all branches for all instances;
- $\mathsf{CT}_{\forall\exists}$ - the class of dependencies for which the chase algorithm terminates for all instances on at least one branch;

- $\mathsf{CT}_{I,\forall}$ - the class of dependencies for which the chase algorithm terminates with input I on all branches;
- $\mathsf{CT}_{I,\exists}$ - the class of dependencies for each the chase algorithm terminates with input I on at least one branch.

The first two cases generate the instance independent termination classes, whereas the latter ones give the instance dependent termination classes. In this thesis we focused on the instance independent classes and we showed that these collapse for all of the chase variations except for the standard chase case. In Chapter 3 we clearly delimited these instance independent classes for each of the chase variations. In the following chapter we turned our attention to the decidable classes of dependencies introduced so far that ensure the chase termination for all instances. We compared these termination criteria and checked if for each chase variation the criterion ensures the termination. For this we introduced a rather nice characterization of the oblivious and semi-oblivious chase based on the standard-chase termination. This characterization served as a tool to delimit these classes of dependencies.

Next we focused on three main problems that take advantage of the chase process in order to compute different types of solutions. The problems we considered are: data exchange, data repair and data correspondence. In this dissertation we formally introduced the data correspondence problem as a new database problem of high practical interest. For the data repair and data correspondence problems we introduce new complexity boundaries for the *solution-existence* and *solution-checking* problems. These boundaries are proved by giving a general characterization theorem for the symmetric-difference solution based on a new notion of Σ-*satisfying homomorphism.*

The query answering problem on incomplete databases was always a challenge in the research community. Unfortunately, the homomorphic equivalence property of the instances returned by the presented chase variations does not help when dealing with general queries. This is why a closed world semantics needs to be adopted in order to get more natural answers. We started Chapter 6 by presenting the most prominent closed world semantics and showed, based on examples, that these are not “good“ candidates when dealing with the data exchange problem. The next step was then to introduce a more natural closed-world semantics which, even if it is not closed under logical equivalence, is argued to be a good semantics for data exchange. Also it was shown that, under this semantics, there exists a conditional-chase procedure able to compute a representation, in this case a conditional table, for the solution space introduced by the new semantics. As far as the termination of the conditional chase concerns, we showed that any richly acyclic set of dependencies ensures the conditional-chase termination (even more, we proved that the class of richly acyclic sets of dependencies ensure the chase termination for oblivious chase as well).

Finally, in the last chapter we studied the problem of representation systems and data exchange representation systems. As it is well known that the composition between mappings is not closed under sets of TGD's, we focused our attention on a higher class of dependencies, namely st-SO dependencies, for which we extended our closed world semantics. We also proved that even in this case there exists a conditional-chase procedure able to compute a conditional table with a global condition which represents the solution space under constructible models semantics.

Most of the time in this thesis, except for Chapter 7, we only considered dependencies specified by sets of TGD's. As a further work, we may also include the implication of mixing EGD's with TGD's. Clearly the EGD's addition may make the chase algorithm terminate earlier by *failure*, so it may be that there exist classes of mixed dependencies for which, because of the presence of the EGD's, may give us larger classes of dependencies that ensure the instance dependent chase termination. Another problem that still remains open, to the best of the author's knowledge, is if it is decidable to test if for a given set of TGD's one of the chase algorithm variations terminates on all instances (though this problem is considered to be highly undecidable). For now it is known that it is undecidable to test if for a given instance and a given set of TGD's the chase process terminates. This means that it may even be possible that the problem of chase termination on all instances could be decidable making all the classes that ensure sufficient condition for the chase termination dispensable.

For the data correspondence problem, uniform and non-uniform cases, it will be interesting to view the data complexity on getting the consistent answers to different query types. Similar work was recently done by Cate, Fontaine and Kolaitis in [72] for the database repair problem.

As presented in this thesis the conditional chase can be a good candidate for a closed world semantics chase procedure. Still, there is more work to do regarding the data complexity problems related to the resulting conditional tables. For example, testing if a tuple is part of a possible world from the representation of a conditional table is NP-hard in general [2]. Still, one may find some classes of dependencies such that when conditional chasing a ground instance with a set of TGD's from this class, the problem will become tractable. The same may apply for certain and possible query answering problems.

References

[1] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases.* Addison-Wesley, 1995.

[2] Serge Abiteboul, Paris C. Kanellakis, and Gösta Grahne. On the representation and querying of sets of possible worlds. In *SIGMOD Conference*, pages 34–48, 1987.

[3] Serge Abiteboul, Paris C. Kanellakis, and Gösta Grahne. On the representation and querying of sets of possible worlds. *Theor. Comput. Sci.*, 78(1):158–187, 1991.

[4] Foto N. Afrati and Phokion G. Kolaitis. Repair checking in inconsistent databases: algorithms and complexity. In *ICDT*, pages 31–41, 2009.

[5] Foto N. Afrati, Chen Li, and Vassia Pavlaki. Data exchange in the presence of arithmetic comparisons. In *EDBT*, pages 487–498, 2008.

[6] A. V. Aho, C. Beeri, and J. D. Ullman. The theory of joins in relational databases. *ACM Trans. Database Syst.*, 4(3):297–314, 1979.

[7] Marcelo Arenas, Pablo Barceló, Ronald Fagin, and Leonid Libkin. Locally consistent transformations and query answering in data exchange. In *PODS*, pages 229–240, 2004.

[8] Marcelo Arenas, Leopoldo E. Bertossi, and Jan Chomicki. Consistent query answers in inconsistent databases. In *PODS*, pages 68–79, 1999.

[9] Marcelo Arenas, Ronald Fagin, and Alan Nash. Composition with target constraints. In *ICDT*, pages 129–142, 2010.

[10] Marcelo Arenas, Jorge Pérez, and Juan L. Reutter. Data exchange beyond complete data. In *PODS*, pages 83–94, 2011.

[11] Renée J. Miller Ariel Fuxman, Phokion G. Kolaitis and Wang Chiew Tan. Peer data exchange. In *PODS*, pages 160–171, 2005.

[12] François Bancilhon, David Maier, Yehoshua Sagiv, and Jeffrey D. Ullman. Magic sets and other strange ways to implement logic programs. In *PODS*, pages 1–15, 1986.

[13] Catriel Beeri and Moshe Y. Vardi. A proof procedure for data dependencies. *J. ACM*, 31(4):718–741, 1984.

[14] Philip A. Bernstein, Alon Y. Halevy, and Rachel Pottinger. A vision of management of complex models. *SIGMOD Record*, 29(4):55–63, 2000.

[15] Andrea Calì, Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Data integration under integrity constraints. *Inf. Syst.*, 29(2):147–163, 2004.

[16] Andrea Calì, Georg Gottlob, and Michael Kifer. Taming the infinite chase: Query answering under expressive relational constraints. In *KR*, pages 70–80, 2008.

[17] Andrea Calì, Georg Gottlob, and Thomas Lukasiewicz. Datalog$^{\pm}$: a unified approach to ontologies and integrity constraints. In *ICDT*, pages 14–30, 2009.

[18] Andrea Calì, Georg Gottlob, and Thomas Lukasiewicz. A general datalog-based framework for tractable query answering over ontologies. In *PODS*, pages 77–86, 2009.

[19] Andrea Calì, Georg Gottlob, and Andreas Pieris. Query answering under non-guarded rules in datalog+/-. In *RR*, pages 1–17, 2010.

[20] Edward P. F. Chan. A possible world semantics for disjunctive databases. *IEEE Trans. Knowl. Data Eng.*, 5(2):282–292, 1993.

[21] Jan Chomicki and Jerzy Marcinkowski. On the computational complexity of minimal-change integrity maintenance in relational databases. In *Inconsistency Tolerance*, pages 119–150, 2005.

[22] Jan Chomicki, Jerzy Marcinkowski, and Slawomir Staworko. Computing consistent query answers using conflict hypergraphs. In *CIKM*, pages 417–426, 2004.

[23] Alin Deutsch, Alan Nash, and Jeffrey B. Remmel. The chase revisited. In *PODS*, pages 149–158, 2008.

[24] Alin Deutsch, Lucian Popa, and Val Tannen. Query reformulation with constraints. *SIGMOD Record*, 35(1):65–73, 2006.

[25] Alin Deutsch and Val Tannen. Reformulation of xml queries and constraints. In *ICDT*, pages 225–241, 2003.

[26] Ronald Fagin. Inverting schema mappings. *ACM Trans. Database Syst.*, 32(4), 2007.

[27] Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa. Data exchange: semantics and query answering. *Theor. Comput. Sci.*, 336(1):89–124, 2005.

[28] Ronald Fagin, Phokion G. Kolaitis, and Lucian Popa. Data exchange: getting to the core. *ACM Trans. Database Syst.*, 30(1):174–210, 2005.

[29] Ronald Fagin, Phokion G. Kolaitis, Lucian Popa, and Wang Chiew Tan. Composing schema mappings: Second-order dependencies to the rescue. *ACM Trans. Database Syst.*, 30(4):994–1055, 2005.

[30] Ronald Fagin, Phokion G. Kolaitis, Lucian Popa, and Wang Chiew Tan. Reverse data exchange: Coping with nulls. *ACM Trans. Database Syst.*, 36(2):11, 2011.

[31] Ronald Fagin and Moshe Y. Vardi. The theory of data dependencies - an overview. In *ICALP*, pages 1–22, 1984.

[32] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[33] Georg Gottlob and Alan Nash. Data exchange: computing cores in polynomial time. In *PODS*, pages 40–49, 2006.

[34] Georg Gottlob and Alan Nash. Efficient core computation in data exchange. *J. ACM*, 55(2), 2008.

[35] Gösta Grahne. *The Problem of Incomplete Information in Relational Databases*, volume 554 of *Lecture Notes in Computer Science*. Springer, 1991.

[36] Gösta Grahne and Victoria Kiricenko. Towards an algebraic theory of information integration. *Inf. Comput.*, 194(2):79–100, 2004.

[37] Gösta Grahne and Adrian Onet. Data correspondence, exchange and repair. In *ICDT*, pages 219–230, 2010.

[38] Gösta Grahne and Adrian Onet. Closed world chasing. In *LID*, pages 7–14, 2011.

[39] Gösta Grahne and Adrian Onet. On conditional chase termination. In *AMW*, 2011.

[40] Gösta Grahne and Adrian Onet. Representation systems for data exchange. In *ICDT*, pages 219–230, 2012.

[41] Sergio Greco, Francesca Spezzano, and Irina Trubitsyna. Stratification criteria and rewriting techniques for checking chase termination. *PVLDB*, 4(11):1158–1168, 2011.

[42] Pavol Hell and Jaroslav Nesetril. The core of a graph. *Discrete Mathematics*, 109(1-3):117–126, 1992.

[43] Pavol Hell and Jaroslav Nesetril. *Graphs And Homomorphisms.* Oxford University Press, 2004.

[44] André Hernich. Answering non-monotonic queries in relational data exchange. In *ICDT*, pages 143–154, 2010.

[45] André Hernich. *Foundations of query answering in relational data exchange.* PhD thesis, 2010.

[46] André Hernich. Computing universal models under guarded tgds. In *ICDT*, page to appear, 2012.

[47] André Hernich and Nicole Schweikardt. Cwa-solutions for data exchange settings with target dependencies. In *PODS*, pages 113–122, 2007.

[48] Tomasz Imielinski and Witold Lipski Jr. Incomplete information in relational data bases. *ICS PAS Report 475*, Polish Academy of Sciences, 1982.

[49] Tomasz Imielinski and Witold Lipski Jr. Incomplete information in relational databases. *J. ACM*, 31(4):761–791, 1984.

[50] David S. Johnson and Anthony C. Klug. Testing containment of conjunctive queries under functional and inclusion dependencies. *J. Comput. Syst. Sci.*, 28(1):167–189, 1984.

[51] Gregory Karvounarakis and Val Tannen. Conjunctive queries and mappings with unequalities. In *Technical Reports (CIS)*, pages 1–15, 2008.

[52] Phokion G. Kolaitis, Jonathan Panttaja, and Wang Chiew Tan. The complexity of data exchange. In *PODS*, pages 30–39, 2006.

[53] Maurizio Lenzerini. Data integration: A theoretical perspective. In *PODS*, pages 233–246, 2002.

[54] Leonid Libkin. Data exchange and incomplete information. In *PODS*, pages 60–69, 2006.

[55] Leonid Libkin. Incomplete information and certain answers in general data models. In *PODS*, pages 59–70, 2011.

[56] Leonid Libkin and Cristina Sirangelo. Open and closed world assumptions in data exchange. In *Description Logics*, 2009.

[57] Andrei Lopatenko and Leopoldo E. Bertossi. Complexity of consistent query answering in databases under cardinality-based and incremental repair semantics. In *ICDT*, pages 179–193, 2007.

[58] David Maier, Alberto O. Mendelzon, and Yehoshua Sagiv. Testing implications of data dependencies. *ACM Trans. Database Syst.*, 4(4):455–469, 1979.

[59] Bruno Marnette. Generalized schema-mappings: from termination to tractability. In *PODS*, pages 13–22, 2009.

[60] Bruno Marnette and Floris Geerts. Static analysis of schema-mappings ensuring oblivious termination. In *ICDT*, pages 183–195, 2010.

[61] Michael Meier, Michael Schmidt, and Georg Lausen. On chase termination beyond stratification. *PVLDB*, 2(1):970–981, 2009.

[62] Michael Meier, Michael Schmidt, and Georg Lausen. On chase termination beyond stratification [technical report and erratum]. Website, 2009. http://arxiv.org/abs/0906.4228.

[63] Michael Meier, Michael Schmidt, Fang Wei, and Georg Lausen. Semantic query optimization in the presence of types. In *PODS*, pages 111–122, 2010.

[64] Alberto O. Mendelzon. Database states and their tableaux. *ACM Trans. Database Syst.*, 9(2):264–282, 1984.

[65] Renée J. Miller, Mauricio A. Hernández, Laura M. Haas, Ling-Ling Yan, C. T. Howard Ho, Ronald Fagin, and Lucian Popa. The clio project: Managing heterogeneity. *SIGMOD Record*, 30(1):78–83, 2001.

[66] Jack Minker. On indefinite databases and the closed world assumption. In *CADE*, pages 292–308, 1982.

[67] Adrian Onet. The chase procedure and its applications in data exchange. In *DEIS*, to appear.

[68] Raymond Reiter. On closed world data bases. In *Logic and Data Bases*, pages 55–76, 1977.

[69] Thomas J. Schaefer. The complexity of satisfiability problems. In *STOC*, pages 216–226, 1978.

[70] Francesca Spezzano and Sergio Greco. Chase termination: A constraints rewriting approach. *PVLDB*, 3(1):93–104, 2010.

[71] Slawomir Staworko and Jan Chomicki. Priority-based conflict resolution in inconsistent relational databases. *CoRR*, abs/cs/0506063, 2005.

[72] Balder ten Cate, Gaëlle Fontaine, and Phokion G. Kolaitis. On the data complexity of consistent query answering. In *ICDT*, pages 22–33, 2012.

[73] Balder ten Cate and Phokion G. Kolaitis. Structural characterizations of schema-mapping languages. In *ICDT*, pages 63–72, 2009.

[74] Moshe Y. Vardi. Inferring multivalued dependencies from functional and join dependencies. *Acta Inf.*, 19:305–324, 1983.

[75] Jef Wijsen. Database repairing using updates. *ACM Trans. Database Syst.*, 30(3):722–768, 2005.

Index