

UNIVERSITÀ DELLA CALABRIA
Facoltà di Ingegneria
Dipartimento di Elettronica, Informatica e Sistemistica

Dottorato di Ricerca in Ingegneria dei Sistemi ed Informatica
XXIV ciclo
Settore Scientifico Disciplinare ING/INF-05

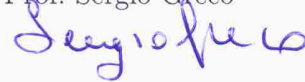
Tesi di Dottorato

ON THE PROBLEM OF CHECKING
CHASE TERMINATION

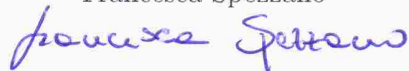
Coordinatore
Prof. Luigi Palopoli



Supervisore
Prof. Sergio Greco



Francesca Spezzano



Anno Accademico 2010/2011

UNIVERSITÀ DELLA CALABRIA
Facoltà di Ingegneria
Dipartimento di Elettronica, Informatica e Sistemistica

Dottorato di Ricerca in Ingegneria dei Sistemi ed Informatica
XXIV ciclo
Settore Scientifico Disciplinare ING/INF-05

Tesi di Dottorato

ON THE PROBLEM OF CHECKING
CHASE TERMINATION

Coordinatore
Prof. Luigi Palopoli

Supervisore
Prof. Sergio Greco

Francesca Spezzano

Anno Accademico 2010/2011

Preface

Several database areas such as data exchange and data integration share the problem of fixing database instance violations with respect to a set of integrity constraints. The chase algorithm solves such violations by inserting tuples and setting the value of nulls. Unfortunately, the chase algorithm may not terminate and the problem of deciding whether the chase process terminates is undecidable. In recent years, the problem known as chase termination has been investigated. It consists in the detection of sufficient conditions, derived from the structural analysis of dependencies, guaranteeing that the chase fixpoint terminates independently from the database instance. Several criteria introducing sufficient conditions for chase termination have been recently proposed (weak acyclicity, stratification, super-weak acyclicity, inductive restriction, etc.). Most of these criteria concentrate on tuple generating dependencies (TGDs).

The aim of this thesis is to present more general criteria and techniques for chase termination.

We first present extensions of the well-known stratification criterion and introduce a new criterion, called local stratification (LS), which generalizes both super-weak acyclicity and stratification-based criteria (including the class of constraints which are inductively restricted).

Next the thesis presents a rewriting algorithm transforming the original set of constraints Σ into an ‘equivalent’ set Σ^α and verifying the structural properties for chase termination on Σ^α . The rewriting of constraints allows to recognize larger classes of constraints for which chase termination is guaranteed. In particular, we show that if Σ satisfies chase termination condition C , then the rewritten set Σ^α satisfies C as well, but the vice versa is not true, that is there are significant classes of constraints for which Σ^α satisfies C and Σ does not.

A more general rewriting algorithm producing as output an equivalent set of dependencies and a boolean value stating whether a sort of cyclicity has been detected is also proposed. The new rewriting technique and the checking of acyclicity allow us to introduce the class of acyclic constraints (AC), which

generalizes LS and guarantees that all chase sequences are finite with a length polynomial in the size of the input database.

Finally, a system prototype that allows users to design data dependencies and apply different criteria and algorithms for checking chase termination is presented. The tool is also able to execute the chase procedure in order to repair the possible incomplete database provided by the user, and to execute (restricted) SQL queries over the repaired database.

Acknowledgments

My full gratitude goes to my supervisor Prof. Sergio Greco for his excellent and always present guidance during my doctoral course and all opportunities offered to me.

A special thank to my colleague and friend Irina Trubitsyna for her help and good suggestions.

I am also grateful to Prof. Phokion Kolaitis for giving me the opportunity to spend a year abroad at UCSC, and have interesting research meetings with him.

Last but not the least, I would like to thank for their support my Mum and Dad, my sister Annachiara and my love Edoardo.

Rende,

November 2011

Francesca Spezzano

Contents

1	Introduction	1
2	Theoretical Background	9
2.1	Relational Model	9
2.2	Relational Dependencies	10
2.3	Homomorphisms and Universal Solutions	12
3	The Chase Algorithm	15
3.1	Standard Chase Algorithm	15
3.2	Other Kinds of Chase	16
3.2.1	Oblivious Chase	16
3.2.2	Core Chase	18
3.3	Chase Termination Conditions	18
3.4	Relationship among Chase Termination Conditions	25
4	Applications	29
4.1	Data Dependencies Implication Problem	29
4.2	Database Design	30
4.3	Query Containment under Constraints	31
4.4	Query Optimization	32
4.5	Query Answering on Incomplete Data	33
4.6	Data Integration	35
4.7	Data Exchange	37
5	New Chase Termination Conditions	39
5.1	WA-Stratification	39
5.2	Local Stratification	43
6	Checking Chase Termination by Constraints Rewriting	47
6.1	Constraints Rewriting	47
6.2	Cyclicity Detection during Rewriting Process	52

VIII Contents

7	The <i>ChaseTEQ</i> System Prototype	61
7.1	System Description	61
7.2	Implementation	64
7.3	Application Scenario	66
8	Conclusions	69
	References	71

Introduction

The Chase is a fixpoint algorithm enforcing satisfaction of data dependencies in databases. It has been proposed more than thirty years ago [ABU79, MMS79] and has received an increasing attention in recent years in both database theory and practical applications. Indeed, the availability of data coming from different sources easily results in inconsistent or incomplete data (i.e. data not satisfying data dependencies) and, therefore, techniques for fixing inconsistencies are crucial [Ber06, Cho07]. The chase algorithm is used, directly or indirectly, on an everyday basis by people who design databases, and it is used in commercial systems to reason about the consistency and correctness of a data design. New applications of the chase in meta-data management, ontological reasoning, data exchange and data cleaning have been proposed as well [BKL11, CGP10, DLLR07].

The execution of the chase algorithm involves the insertion of tuples with possible null values and the changing of nulls which can be made equal to constants or other null values. However, the insertion of tuples with new (null) values could result in a non-terminating execution. The following example shows a case where a given database does not satisfy a set of data dependencies (also called constraints) and the application of the chase algorithm produces a new consistent database by adding tuples with nulls.

Example 1.1. Consider the set of constraints $\Sigma_{1.1}$:

$$\begin{aligned}\forall x N(x) &\rightarrow \exists y E(x, y) \\ \forall x \forall y S(y) \wedge E(x, y) &\rightarrow N(y)\end{aligned}$$

where the relations N and S store normal nodes and special nodes, respectively, whereas E stores edges. The second constraint states that if there exists an edge from x to y and y is a special node, then y must also be a (normal) node. The first constraint states that every normal node must have an outgoing edge.

Assume that the database contains the tuples $S(a), E(b, a)$. Since the second constraint is not satisfied, the tuple $N(a)$ is inserted. This update ope-

ration fires the first constraint to insert the tuple $E(a, n_1)$, where n_1 is a new labeled null. At this point the chase terminates since the database is consistent, i.e. the second constraint cannot be fired because n_1 is not in the relation S . The output database is consistent as both dependencies are satisfied. \square

However, it is important to observe that if we delete the atom $S(y)$ from the second constraint, the chase will never terminate as an infinite number of tuples will be added to the database.

The problem recently investigated, known as *chase termination*, consists in the identification of sufficient conditions, based on structural properties of the input set of constraints, guaranteeing that the chase fixpoint terminates independently from the database instance.

Several criteria guaranteeing chase termination for all chase sequences and for all database instances have been recently defined.

Fagin et al. [FKMP05] introduced the class of *weakly acyclic* sets of constraints (*WA*). Informally, weak acyclicity checks that the set of constraints does not present cyclic conditions for which a new null value forces (directly or indirectly) the introduction of another null in the same position. In the previous example we have that the presence of a value in N_1 , i.e. in the first position of the predicate N , forces the introduction of a new null value in position E_2 , i.e. in the second position of the predicate E , and it is denoted as $N_1 \xrightarrow{*} E_2$; this value is then introduced in position N_1 (denoted as $E_2 \rightarrow N_1$) and next a new null value is introduced in E_2 . The cycle going through the special edge $N_1 \xrightarrow{*} E_2$ means that an infinite number of nulls could be introduced.

The class of weakly acyclic sets of constraints has been generalized in several works [DNR08, Mar09, MSL09a].

Deutsch et al. proposed an extension of weak acyclicity called *stratification* (*Str*) [DNR08]. The idea behind stratification is to decompose, by constructing a *chase graph*, the set of constraints into independent subsets, where each subset consists of constraints that may fire each other, and to check each component separately for weak acyclicity. However, in [MSL09b] it has been shown that stratification is not able to check termination of all chase sequences, but it is sufficient to state that if a set of constraints is stratified, then there is at least one terminating chase sequence which can be determined from the chase graph. Thus, a variant of stratification, called *c-stratification* (*CStr*) [MSL09b], has been proposed to guarantee the termination of all chase sequences.

Meier et al. proposed a different extension of weak acyclicity called *safety* (*SC*) [MSL09a]. The improvement is based on the fact that only the effective propagation of null values should be considered in the graph. Thus, a variable can propagate nulls only if all its occurrences appear in ‘affected’ positions, i.e. positions which may actually contain null values [CGK08]. The Example 1.1 presents a set of constraints which is both safe and stratified, but not weakly acyclic. In fact, according to the safety condition, we have that E_2 does not

propagate null values to N_1 as the variable y also appears in the relation S which does not contain null values. So, position N_1 cannot contain nulls, i.e. it is not affected. From the motivation that S cannot contain null values, we have that firing r_1 cannot cause r_2 to fire, and then $\Sigma_{1.1}$ is also stratified.

Stratification and safety are not comparable (i.e. there are sets of constraints which only satisfy one of the two criteria), but are both generalized by the *inductive restriction* (*IR*) criterium [MSL09a, MSL09b].

A different extension of weak acyclicity, not comparable with inductive restriction, has been introduced in [Mar09] under the name of *super-weak acyclicity* (*SwA*). Basically, *SwA* takes into account the fact that variables may appear more than once in body atoms of constraints and, therefore, when different nulls are inserted in positions associated with the same variable, constraints are not fired.

The idea underlying stratification, also used in its variation and extensions (*CStr* and *IR*) and in the super-weak acyclicity, is to consider, in the propagation of nulls, how constraints may fire each other. However, there are simple cases where current criteria are not able to understand that all chase sequences are finite.

Example 1.2. Consider the following set of constraints $\Sigma_{1.2}$ consisting of the TGD

$$\forall x \forall y E(x, y) \wedge E(y, x) \rightarrow \exists z E(y, z) \wedge E(z, x)$$

We can distinguish two cases. If we suppose to have a database instance $D_1 = \{E(a, b), E(b, a)\}$ we have that the TGD is not satisfied, but the constraint will be applied only once by the chase as the resulting database $D'_1 = \{E(a, b), E(b, a), E(b, \eta_1), E(\eta_1, a)\}$ is consistent. Otherwise, if we suppose to have a database instance $D_2 = \{E(a, a)\}$, it is already consistent, and no chase step will be applied. It is easy to see that the chase is always terminating for all database instances, but none of the existing criteria guaranteeing the termination of all chase sequences is able to recognize it as terminating. \square

In order to cope with this problem, we propose extensions of the well-known stratification criterion and then introduce a new criterion, called *local stratification* (*LS*), which generalizes both super-weak acyclicity and stratification-based criteria (including the class of constraints which are inductively restricted) and guarantees the termination of all chase sequences, for all database instances, in polynomial time.

Nevertheless, despite the previously mentioned results, there are still important classes of terminating data dependencies which are not identified by none of the previous mentioned criteria.

Example 1.3. Consider the set of constraints $\Sigma_{1.3}$:

$$\begin{aligned} \forall x N(x) &\rightarrow \exists y E(x, y) \\ \forall x \forall y S(x) \wedge E(x, y) &\rightarrow N(y) \end{aligned}$$

that is a variant of Example 1.1 where the second constraint states that if there exists an edge from x to y and x is a special node, then y must be a (normal) node. Assume that the database contains the tuples $S(a), N(a)$. The first step of the chase algorithm will add the tuple $E(a, n_1)$, where n_1 is a new labeled null, to the database. At this point, since the second constraint is no more satisfied, the second constraint fires to insert the tuple $N(n_1)$ which in turn fires the first constraint so that the tuple $E(n_1, n_2)$ is added to the database. At this point the chase terminates (since the the tuple $S(n_1)$ is not in the database), and both dependencies are satisfied. \square

Thus, we present an orthogonal technique which enlarges the classes of dependencies recognized by the above criteria. This technique rewrites the input set of constraints Σ into an equivalent, but more informative, set Σ^α and checks classical criteria satisfaction on Σ^α . The rewriting of constraints allows to recognize larger classes of constraints for which chase termination is guaranteed. In particular, we show that if Σ satisfies chase termination conditions C , then the rewritten set Σ^α satisfies C as well, but the vice versa is not true, that is there are significant classes of constraints for which Σ^α satisfies C and Σ does not.

In the presence of constraints that are TGDs and EGDs, the previous described criteria and technique are quite weak, as it may happen that a solution can be found even if the set of TGDs are non-terminating.

Example 1.4. Consider the set of constraints $\Sigma_{1.4}$, stating that each node has an outgoing edge (r_1), each edge ends in a node (r_2) and the resulting graph must contain at most one node (EGD r_3).

$$\begin{aligned} r_1 : & \quad \forall x N(x) \rightarrow \exists y E(x, y) \\ r_2 : & \quad \forall x \forall y E(x, y) \rightarrow N(y) \\ r_3 : & \quad \forall x \forall y N(x) \wedge N(y) \rightarrow x = y \end{aligned}$$

The alternating application of constraints r_1 and r_2 ($r_1, r_2, r_1, r_2, \dots$) could be not terminating, whereas the alternating application of constraints r_1, r_2 and r_3 (r_1, r_2, r_3, \dots) always terminates. Indeed, assuming that the database D contains only the tuple $N(a)$, a solution for D and Σ is obtained by introducing the tuple $E(a, a)$. This solution is obtained as follows: by applying the first constraint which is not satisfied, the tuple $E(a, \eta_1)$ is introduced. Then, from the second constraint, the tuple $N(\eta_1)$ must also be inserted. At this point the application of the third constraint updates the null value η_1 to a . \square

Thus, the right selection of dependencies to be applied could inhibit the firing of constraints which may cause the cyclic introduction of null values, guaranteeing the presence of at least one terminating chase sequence.

Contributions

The main contributions of this thesis are the following:

- We first analyze the relationship among current criteria and show that super-weak acyclicity is not comparable with (c-)stratification, but it extends safety.
- We analyze the stratification conditions and propose a new stratification criterion, called *WA-stratification*, which builds a different chase graph $\Gamma(\Sigma)$, called *firing graph*, and overcomes some limitations of both stratification and c-stratification; since WA-stratification checks weak acyclicity over strong components of $\Gamma(\Sigma)$, two further, more powerful, conditions checking safety and super-weak acyclicity over $\Gamma(\Sigma)$ are defined. These criteria are called *SC-stratification* and *SwA-stratification*.
- SwA-stratification, the most general of the above new criteria, is not comparable with inductive restriction. Thus, we propose a further criterion, called *Local stratification (LS)* which uses more refined conditions of SwA during the construction of the firing graph. *LS* generalizes SwA-stratification and is more powerful than *IR*.
- As a minor result, since our criteria are based on the construction of a firing graph, an algorithm solving the firing problem (i.e. checking whether a constraint r_1 could fire a constraint r_2) is presented. It is shown that, although its complexity is exponential in the number of atoms in the body of r_2 , because it depends on the number of atoms in the head of r_1 unifying with each single atom in the body of r_2 , for some standard classes of constraints (e.g. multivalued dependencies, inclusion dependencies) it performs very well (recall that the firing problem for the (c-)stratification criterion is in \mathcal{NP} [DNR08, MSL09a]).
- Next, we present a technique for rewriting a set of constraints into an ‘equivalent’ set by adorning predicate symbols and show that the checking of termination criteria over the target set allows to detect larger classes of source constraints for which all chase sequences terminate;
- We extend the rewriting technique to capture even larger classes of constraints for which the chase terminates by analyzing affected positions (positions which may hold nulls); this is carried out by adorning predicates and using different adorning symbols in cases where we can statically establish that two affected positions cannot contain the same null value;
- The rewriting technique permits us to introduce a new class of terminating constraints consisting of a set of dependencies which are detected as acyclic by our rewriting algorithm. This class, called *Acyclic (AC)* generalizes *LS* and, considering static criteria for checking chase termination, is the most general criterion so far proposed ¹.
- Checking termination criteria and rewriting techniques have been implemented into a system prototype downloadable from the web. This system also allows to execute the chase fixpoint to repair and querying a given database on the base of the input dependencies.

¹ In this thesis we will use calligraphic style \mathcal{C} in order to denote the terminating class of constraints recognized by criterion C .

Organization

This thesis is organized as follows.

Chapter 2 introduces the basic notions that we shall use throughout the thesis.

Chapter 3 presents the chase algorithm and its variants and gives a survey on the well-known chase termination criteria proposed in the literature. In particular, weak acyclicity, safety, (c-)stratification, super-weak acyclicity, safe restriction and inductive restriction are explained.

Chapter 4 discusses several problems and applications using the chase algorithm, namely implication of data dependencies, database design, query optimization, query answering and containment under constraints, data integration and data exchange.

Chapter 5 presents some improvements for termination conditions discussed in Chapter 3 and then describes the class of locally stratified dependencies, that generalizes previously known classes, for which termination of the chase algorithm is guaranteed.

Chapter 6 presents a technique for checking chase termination based on rewriting the original set of TGDs into an ‘equivalent’ set Σ^α , whose structural properties allow to detect larger classes of source constraints for which all chase sequences terminate. Section 6.2 extends the rewriting technique by analyzing affected positions (positions which may hold nulls) and by introducing cyclicity detection in the rewriting process. This approach allows us to further enlarge the class of terminating dependencies and obtain the most general criterion so far proposed.

Chapter 7 briefly presents *ChaseTEQ*, a system prototype available on the web which implements some of the criteria and techniques for checking chase termination presented in this thesis, and allow the user to repair and querying the possible incomplete database.

Finally, in Chapter 8 conclusions are drawn.

Summary of Publications

Some of the results presented in this thesis appeared in the conference papers [GS10], presented at the *36th International Conference on Very Large Data Bases (VLDB 2010)* and [GST11] presented at the *37th International Conference on Very Large Data Bases (VLDB 2011)*. The system prototype has been presented in [DST11] at the *19th Italian Symposium on Advanced Database Systems (SEBD 2011)* and in [DGST11] at the *5th International Conference on Scalable Uncertainty Management (SUM 2011)*.

Theoretical Background

In this chapter we define the basic notions that we shall use throughout the thesis.

2.1 Relational Model

We introduce the following disjunct sets of symbols: (i) an infinite set *Consts* of *constants*, (ii) an infinite set *Nulls* of *labeled nulls* and (iii) an infinite set *Vars* of *variables*.

A *relational schema* \mathcal{R} is a pair $\langle \mathbf{R}, \Sigma \rangle$ where:

- \mathbf{R} is a set of relational predicates R , each with its associated arity $ar(R)$ that indicates the number of its attributes X , and
- Σ is a set of (*integrity*) *constraints* expressed on the relations in \mathbf{R} , i.e. assertion on the relations in \mathbf{R} that are intended to be satisfied by database instances.

When no integrity constraints are defined in \mathcal{R} , we simply denote the relational schema with \mathbf{R} .

An *instance* of a relational predicate R of arity n is a set of ground atoms in the form $R(c_1, \dots, c_n)$, where $c_i \in Consts \cup Nulls$. Such (ground) atoms are also called tuples or facts. A *database instance*, or simply an instance or a database, for a relational schema $\mathcal{R} = \langle \mathbf{R}, \Sigma \rangle$ is a set of instances for the relations in \mathbf{R} . A database instance for a schema \mathcal{R} is said to be *consistent* with \mathcal{R} if it satisfies all constraints expressed on \mathcal{R} . We denote by D a database instance constructed on *Consts* and by J, K the database instances constructed on $Consts \cup Nulls$. Given an instance K , $Nulls(K)$ (resp. $Consts(K)$) denotes the set of labeled nulls (resp. constants) occurring in K . An *atomic formula* (or *atom*) is of the form $R(t_1, \dots, t_n)$ where R is a relational predicate, t_1, \dots, t_n are terms belonging to the domain $Consts \cup Vars$ and $n = ar(R)$.

Let K be a database over a relational schema \mathbf{R} and $S \subseteq \mathbf{R}$, then $K[S]$ denotes the subset of K consisting of instances whose predicates are in S (clearly $K = K[\mathbf{R}]$). Analogously, if we have a collection of databases $K_C = \{K_1, \dots, K_n\}$ where each K_i is defined over a schema \mathbf{R}_i and let $S \subseteq \bigcap_{i \in [1..n]} \mathbf{R}_i$, then $K_C[S] = \{K_1[S], \dots, K_n[S]\}$.

A *position* R_i is a pair (R, i) , where R is a relation predicate belonging to the schema \mathbf{R} and i denotes the i -wise attribute of R .

Given a relation schema $R(X, Y)$ with sets of attributes X and Y , we denote by $R[X]$ the *projection* of the relation R onto attributes X .

An n -ary *conjunctive query* (CQ) over a schema \mathbf{R} is a formula of the form

$$Q(\mathbf{x}) \leftarrow \exists \mathbf{y} \Phi(\mathbf{x}, \mathbf{y})$$

where Q is a predicate not appearing in \mathbf{R} , and $\Phi(\mathbf{x}, \mathbf{y})$ is a conjunction of atoms constructed with predicates from \mathbf{R} . The arity of a query is the arity of its head predicate Q : if Q has arity 0, then the query is called *Boolean*. The answers to a query Q evaluated on the database instance K is denoted as $Q(K)$.

2.2 Relational Dependencies

The set of integrity constraints that we consider are *tuple generating dependencies* (TGDs) and *equality generating dependencies* (EGDs).

Given a relational schema \mathbf{R} , a tuple generating dependency over \mathbf{R} is a formula of the form

$$r : \forall \mathbf{x} \forall \mathbf{z} \phi_{\mathbf{R}}(\mathbf{x}, \mathbf{z}) \rightarrow \exists \mathbf{y} \psi_{\mathbf{R}}(\mathbf{x}, \mathbf{y}) \quad (2.1)$$

where $\phi_{\mathbf{R}}(\mathbf{x}, \mathbf{z})$ and $\psi_{\mathbf{R}}(\mathbf{x}, \mathbf{y})$ are conjunctions of atomic formulas over \mathbf{R} ; $\phi_{\mathbf{R}}(\mathbf{x}, \mathbf{z})$ is called the *body* of r , denoted as $Body(r)$, while $\psi_{\mathbf{R}}(\mathbf{x}, \mathbf{y})$ is called the *head* of r , denoted as $Head(r)$.

An equality generating dependency over \mathbf{R} is a formula of the form

$$\forall \mathbf{x} \phi_{\mathbf{R}}(\mathbf{x}) \rightarrow (x_1 = x_2) \quad (2.2)$$

where x_1 and x_2 are among the variables in \mathbf{x} .

Tuple and equality generating dependencies will be also called *embedded dependencies* or simply *(data) dependencies*. A constraint is said to be *full* if all variables are universally quantified.

In the following we will omit the subscript \mathbf{R} from formulas, whenever the database schema is understood and the universal quantification, since we assume that variables appearing in the body are universally quantified and variables appearing only in the head are existentially quantified. In some cases we also assume that the head and body conjunctions are sets of atoms.

Special classes of data dependencies

A *functional dependency*, or FD [Cod72] is a constraint between two sets of attributes in a relation from a database. Given a relation schema $R(X, Y, Z)$, a set of attributes X in R is said to *functionally determine* another attribute Y , also in R , (written $X \rightarrow Y$) if, and only if, each X value is associated with precisely one Y value. We call X the *determinant set* and Y the *dependent attribute*. Thus, given a tuple and the values of the attributes in X , one can determine the corresponding value of the Y attribute. In simple words, if X value is known, Y value is certainly known. For the purposes of simplicity, given that X and Y are sets of attributes in R , $X \rightarrow Y$ denotes that X functionally determines each of the members of Y (in this case Y is known as the *dependent set*). A functional dependency FD: $X \rightarrow Y$ is called *trivial* if Y is a subset of X .

A *key dependency*, or KD is a minimal set of attributes that functionally determine all of the attributes in a relation.

A functional dependency $X \rightarrow Y$ over a relation schema $R(X, Y, Z)$, where Y is a single attribute, is a special case of EGD as it is a binary unirelational EGD and can be expressed using the form (2.2) as

$$R(\mathbf{x}, y, \mathbf{z}), R(\mathbf{x}', y', \mathbf{z}') \rightarrow y = y'$$

Functional dependencies are certainly the most important and widely-studied integrity constraints for relational databases.

Another important integrity constraint is the *inclusion dependency*, or ID [Fag81]. As an example, an inclusion dependency can say that every MANAGER entry of the R relation appears as an EMPLOYEE entry of the S relation. More generally, an inclusion dependency can say that the projection onto a given m columns of the R relation are a subset of the projection onto a given m columns of the S relation. Hence, IDs are valuable for database design, since they permit us to selectively define what data must be duplicated in what relations.

Consider two relation schemes $R_i(X, Z)$ and $R_j(Y, W)$ (not necessarily distinct), if X is a set of k distinct attributes of R_i , and if Y is a set of k attributes of R_j , then an inclusion dependency is a constraint of the form $R_i[X] \subseteq R_j[Y]$.

An inclusion dependency $R_i[X] \subseteq R_j[Y]$ is a special case of TGDs, as it can be expressed using the form (2.1) as

$$R_i(\mathbf{x}, \mathbf{z}) \rightarrow \exists \mathbf{w} R_j(\mathbf{x}, \mathbf{w})$$

A *foreign key* is a field in a relational table that matches a key of another table. The foreign key can be used to cross-reference tables.

Constraints skolemization

Given a TGD r of the form (2.1) we denote with

$$sk(r) : \forall \mathbf{x} \forall \mathbf{z} \phi(\mathbf{x}, \mathbf{z}) \rightarrow \psi(\mathbf{x}, sk(\mathbf{y}))$$

the skolemized version of r , where each existentially quantified variable $y_i \in \mathbf{y}$ is replaced by a skolem term $f_{y_i}^r(\mathbf{w})$ where $f_{y_i}^r$ is a skolem function and \mathbf{w} denotes the set of universally quantified variables in r defining the scope of the variables \mathbf{y} ¹. In order to clearly identify universally quantified variables denoting the scope of existentially quantified variables we use parenthesis. For instances, in a TGD r of the form $\forall \mathbf{x}(\forall \mathbf{z} \phi(\mathbf{x}, \mathbf{z}) \rightarrow \exists y_1, \dots, y_n \psi(\mathbf{x}, y_1, \dots, y_n))$ variables in \mathbf{x} denote the scope of existentially quantified variables and, therefore, $sk(r)$ (obtained after the rewriting of r in prenex normal form, the skolemization of existentially quantified variables and the re-rewriting of the constraint with the implication operator) is equal to $\forall \mathbf{x}(\forall \mathbf{z} \phi(\mathbf{x}, \mathbf{z}) \rightarrow \psi(\mathbf{x}, f_{y_1}^r(\mathbf{x}), \dots, f_{y_n}^r(\mathbf{x})))$, whereas if r is of the form $\forall \mathbf{x} \forall \mathbf{z} (\phi(\mathbf{x}, \mathbf{z}) \rightarrow \exists y_1, \dots, y_n \psi(\mathbf{x}, y_1, \dots, y_n))$, the corresponding skolemized dependency $sk(r)$ is equal to $\forall \mathbf{x} \forall \mathbf{z} (\phi(\mathbf{x}, \mathbf{z}) \rightarrow \psi(\mathbf{x}, f_{y_1}^r(\mathbf{x}, \mathbf{z}), \dots, f_{y_n}^r(\mathbf{x}, \mathbf{z})))$. For full data dependency r (including EGDs), $sk(r) = r$. Since the resulting set of skolemized dependencies $sk(\Sigma)$ is satisfiable with respect to a database instance if and only if the original set Σ is, constraints in $sk(\Sigma)$ can be applied to database instances and ground skolemized terms appearing in the derived facts are mapped to database values (i.e. terms in $Consts \cup Nulls$).

2.3 Homomorphisms and Universal Solutions

Definition 2.1 (Homomorphism). Let K and J be two instances over relational schema \mathbf{R} with values in $Consts \cup Nulls$. A *homomorphism* $h : K \rightarrow J$ is a mapping from $Consts(K) \cup Nulls(K)$ to $Consts(J) \cup Nulls(J)$ such that: (1) $h(c) = c$, for every $c \in Consts(K)$, and (2) for every fact $R_i(t)$ of K , we have that $R_i(h(t))$ is a fact of J (where, if $t = (a_1, \dots, a_s)$, then $h(t) = (h(a_1), \dots, h(a_s))$). K is said to be *homomorphically equivalent* to J if there is a homomorphism $h : K \rightarrow J$ and a homomorphism $h' : J \rightarrow K$. \square

Similar to homomorphisms between instances, a homomorphism h from a conjunctive formula $\phi(\mathbf{x})$ to an instance J is a mapping from the variables \mathbf{x} to $Consts(J) \cup Nulls(J)$ such that for every atom $R(x_1, \dots, x_n)$ of $\phi(\mathbf{x})$ the fact $R(h(x_1), \dots, h(x_n))$ is in J .

A homomorphism $h : K \rightarrow J$ such that $J \subseteq K$ and $h(x) = x$ for each x in J is called *retraction*. A retraction is *proper* if it is not surjective. An instance is a *core* if it has no proper retractions. A core of an instance K , denoted as $core(K)$, is a retract of K which is a core. Cores of an instance K are unique up to isomorphism.

For any database instance D and set of constraints Σ over a database schema \mathbf{R} , a *solution* for (D, Σ) is an instance J such that $D \subseteq J$ and $J \models \Sigma$

¹ If the set of variables \mathbf{w} is empty, then the skolem function of arity 0 results in a skolem constant.

(i.e. J satisfies all constraints in Σ). A *universal solution* J is a solution such that for every solution J' there exists a homomorphism $h : J \rightarrow J'$. The set of solutions for (D, Σ) will be denoted by $Sol(D, \Sigma)$, whereas the set of universal solutions for (D, Σ) will be denoted by $USol(D, \Sigma)$.

All universal solutions have the same core (up to isomorphism) which is the smallest universal solution. The complexity and the efficient computation of the core of a universal solution has been studied in [FKP05, GN08]. Methods for directly computing the core by SQL queries in a data exchange framework where schema mappings are specified by source-to-target TGDs has been presented in [tCCKT09, MPR09].

The Chase Algorithm

If a data dependencies is not satisfied by a database instance, it is possible to repair the database instance by extending it with new atoms, or by re-naming nulls values. The procedure that enforces the validity of a set of data dependencies is called the chase.

3.1 Standard Chase Algorithm

Definition 3.1 (Chase step [FKMP05]). *Let K be a database instance.*

1. *Let r be a TGD $\phi(\mathbf{x}, \mathbf{z}) \rightarrow \exists \mathbf{y} \psi(\mathbf{x}, \mathbf{y})$. Let h be a homomorphism from $\phi(\mathbf{x}, \mathbf{z})$ to K such that there is no extension of h to a homomorphism h' from $\phi(\mathbf{x}, \mathbf{z}) \wedge \psi(\mathbf{x}, \mathbf{y})$ to K . We say that r can be applied to K with homomorphism h . Let K' be the union of K with the set of facts obtained by: (a) extending h to h' such that each variable in \mathbf{y} is assigned a fresh labeled null, followed by (b) taking the image of the atoms of ψ under h' . We say that the result of applying r to K with h is K' , and write $K \xrightarrow{r, h} K'$.*
2. *Let r be an EGD $\phi(\mathbf{x}) \rightarrow x_1 = x_2$. Let h be a homomorphism from $\phi(\mathbf{x})$ to K such that $h(x_1) \neq h(x_2)$. We say that r can be applied to K with homomorphism h . More specifically, we distinguish two cases.*
 - (a) *If both $h(x_1)$ and $h(x_2)$ are in Consts the result of applying r to K with h is “failure”, and $K \xrightarrow{r, h} \perp$.*
 - (b) *Otherwise, let K' be K where we identify $h(x_1)$ and $h(x_2)$ as follows: if one is a constant, then the labeled null is replaced everywhere by the constant; if both are labeled nulls, then one is replaced everywhere by the other. We say that the result of applying r to K with h is K' , and write $K \xrightarrow{r, h} K'$. \square*

Definition 3.2 (Chase [FKMP05]). *Let Σ be a set of TGDs and EGDs, and let K be an instance.*

- A chase sequence of K with Σ is a sequence (finite or infinite) of chase steps $K_i \xrightarrow{r, h_i} K_{i+1}$, with $i = 0, 1, \dots$, $K_0 = K$ and r a dependency in Σ .
- A finite chase of K with Σ is a finite chase sequence $K_i \xrightarrow{r, h_i} K_{i+1}$, $0 \leq i < m$, with the requirement that either (a) $K_m = \perp$ or (b) there is no dependency r of Σ and there is no homomorphism h_m such that r can be applied to K_m with h_m . We say that K_m is the result of the finite chase. We refer to case (a) as the case of a failing finite chase and we refer to case (b) as the case of a successful finite chase. \square

The chase of K with respect to a set of dependencies Σ , denoted by $\text{chase}(D, \Sigma)$, is the instance obtained by applying all applicable chase steps exhaustively to K .

Two very important properties of the chase algorithm have been individuated.

The first one regards the fact that the chase algorithm produces a universal solution. More specifically, in [FKMP05] it has been shown that, for any instance D and set of constraint Σ : (i) if J is the result of some successful finite chase of (D, Σ) , then J is a universal solution; (ii) if some failing finite chase of (D, Σ) exists, then there is no solution.

The second one deals with the undecidability on the chase algorithm, since, as seen in the Introduction, there are cases in which the chase is non-terminating.

Theorem 3.3. [DNR08] Consider an instance J and a set Σ of TGDs.

1. It is undecidable whether some chase sequence of J with Σ terminates;
2. It is undecidable whether all chase sequences of J with Σ terminate.

The undecidability holds even over a fixed schema, and even if J is an empty instance. \square

Because of the undecidability of the chase termination problem, several sufficient criteria have been defined to ensure the termination of the algorithm (see Section 3.3). Moreover, in this thesis, new and more general conditions and techniques guaranteeing chase termination will be proposed (see Chapter 5 and Chapter 6).

3.2 Other Kinds of Chase

3.2.1 Oblivious Chase

Generally, in the literature two different chase procedures are considered: *standard* and *oblivious*. Intuitively, a standard chase step applies only when there exists a mapping from the body of a constraint to the database instance and the head of the constraint is not satisfied, while an oblivious one always applies when there exists the mapping from the body to the instance, even if the constraint is satisfied. In [PS11] it has been shown that the problem of

checking if $K \models r$, where r is a TGD, is Π_2^P -complete in the size of r . Two different types of oblivious chase have been used in the literature: *naive* [CGK08, MSL09a, tCCKT09] and *skolem* [Mar09].

Example 3.4. Consider the constraint

$$r : E(x, z) \rightarrow \exists y E(x, y)$$

and the database $D = \{E(a, b)\}$. Under the standard chase, the constraint is satisfied and the chase terminates without any application of a chase step. Under the oblivious skolem chase only a tuple $E(a, n_1)$ is added, whereas under the oblivious naive chase an infinite number of tuples $E(a, n_1), E(a, n_2), E(a, n_3), \dots$ is added. Consider now the set of TGDs

$$\begin{aligned} r_1 : E(x, z) &\rightarrow \exists y E(x, y) \\ r_2 : E(x, z) &\rightarrow E(z, x) \end{aligned}$$

and the above database D . In this case, under standard chase only the tuple $E(b, a)$ is added to D , whereas under the oblivious skolem chase an infinite number of tuples is added. \square

Definition 3.5 (TGD oblivious chase step). *Let K be an instance, r a TGD $\phi(\mathbf{x}, \mathbf{z}) \rightarrow \exists \mathbf{y} \psi(\mathbf{x}, \mathbf{y})$ and h a homomorphism from $\phi(\mathbf{x}, \mathbf{z})$ to K . Then, we say that r can be applied to K with homomorphism h . Let K' be the union of K with the set of facts obtained by: (a) extending h to h' such that each variable in \mathbf{y} is assigned a fresh labeled null, followed by (b) taking the image of the atoms of ψ under h' . We say that the result of applying r to K with h is K' , and write $K \xrightarrow{*r, h} K'$. \square*

Since the oblivious chase analyzes only the body of a constraints, the oblivious chase step for an EGD remains the same as in Definition 3.1 (with the only difference that we add an $*$ on the arrow that indicate the step, as in the case of a TGD).

In [CGK08] an important property of the oblivious chase has been explicated, i.e. the fact that there always exists a homomorphism from the oblivious chase to the standard chase. As a consequence, the oblivious chase also produces a universal solution and can be used in lieu of the standard one.

A different kind of oblivious chase is the *oblivious skolem chase* proposed in [Mar09]. Let Σ be a set of TGDs. Then, $\mathcal{P}(\Sigma)$ is the logic program obtained from Σ by skolemizing each TGD of the form (2.1) $r \in \Sigma$, i.e. by replacing each existentially quantified variable $y_i \in \mathbf{y}$ in r with a skolem function $f_{y_i}^r(\mathbf{x})$. The result of a oblivious skolem chase sequence of an instance D w.r.t. a set of constraints Σ is given by the least fixed-point of $D \cup \mathcal{P}(\Sigma)$. It worth noting that this kind of chase introduces, instead of null symbols, skolem terms built upon the constants presents in D and the function symbols present in $\mathcal{P}(\Sigma)$. Moreover, it has been proved that if the oblivious skolem chase terminates, it terminates in polynomial time in the size of D .

3.2.2 Core Chase

In [DNR08] it has been shown that, even if the result of a chase sequence is a universal solution, the chase is not a complete algorithm for finding universal solutions. In fact, we have that, whenever several alternative chase steps could be applied, the chase picks one nondeterministically so that in some cases there is not a unique canonical universal solution, whereas in other cases there is no finite chase; thus, there are instances and sets of constraints for which certain choices lead to terminating chase sequences, while others to non-termination and, in some cases, we cannot produce a universal solution by the chase, as all chase sequences are non-terminating, although a finite solution does exist.

In order to identify a universal solution whenever it exists, a variant of the chase, called *core chase*, has been introduced [DNR08].

Definition 3.6 (Core chase step [DNR08]). *Let Σ be a set of constraints and let I, J, K be three database instances defined over a database schema \mathbf{R} . We say that I is derived from K through a parallel chase step and write $K \xrightarrow{\Sigma} I$ if *i*) $K \not\models \Sigma$ and *ii*) $I = \bigcup_{r \in \Sigma, K \xrightarrow{r} K'} K'$. Moreover, we say that J is derived from K through a core chase step and write $K \xrightarrow{\Sigma \downarrow} J$ if $K \xrightarrow{\Sigma} I$ and $J = \text{core}(I)$. \square*

The definition of core chase sequence derives from the chase sequence by using a core chase step instead of a chase step. Observe that core chase sequences are determined up to isomorphism as well.

In addition, it has been shown that if D is a database instance and Σ is a set of TGDs and EGDs, then there exists a universal solution for Σ and D if and only if the core chase of D with Σ terminates and yields such a solution, that is the core chase is complete for finding universal solutions.

3.3 Chase Termination Conditions

As said in the Introduction, several criteria identifying sufficient conditions for chase termination have been defined in the recent literature: weak acyclicity, safety, stratification, c-stratification, safe restriction, inductive restriction and super-weak acyclicity.

Weak Acyclicity

The first and basic criterion concerning the identification of sufficient conditions, determined by the structure of TGDs, guaranteeing chase termination, is known as *weak acyclicity (WA)*; it was given in [FKMP05, DT03] and inspired by [HY90]. The criterion is based on the structural properties of a graph $\text{dep}(\Sigma)$, called *dependency graph*, derived from the input set of TGDs Σ .

Let Σ be a set of TGDs over a database schema \mathbf{R} , then $\text{pos}(\Sigma)$ denotes the set of positions R_i such that R denotes a relational predicate of \mathbf{R} and there is an R -atom appearing in Σ .

Definition 3.7 (Weakly acyclic set of TGDs [FKMP05]). Let Σ be a set of TGDs over a fixed schema. Construct a directed graph $\text{dep}(\Sigma) = (\text{pos}(\Sigma), E)$, called the dependency graph, whose nodes correspond to the positions in $\text{pos}(\Sigma)$ and the set E of edges is obtained as follows: for every TGD $\phi(\mathbf{x}, \mathbf{z}) \rightarrow \exists \mathbf{y} \psi(\mathbf{x}, \mathbf{y})$ in Σ and for every x in \mathbf{x} that occurs in ϕ in position R_i :

1. for every occurrence of x in ψ in position S_j , add an edge $R_i \rightarrow S_j$ (if it does not already exist).
2. for every existentially quantified variable y and for every occurrence of y in ψ in position T_k , add a special edge $R_i \xrightarrow{*} T_k$ (if it does not already exist).

Then, Σ is weakly acyclic if the corresponding dependency graph $\text{dep}(\Sigma)$ has no cycle going through a special edge. \square

Weak acyclicity criterion guarantees that all chase sequences terminate. Clearly, the problem of checking whether a set of TGDs is weakly acyclic is polynomial in the size of Σ . In [FKMP05] it has been shown that if Σ is the union of a weakly acyclic set of TGDs with a set of EGDs, and D is a database instance, then there exists a polynomial in the size of D that bounds the length of every chase sequence of D with Σ .

Stratification

The idea behind stratification criterion (*Str*) is to decompose the set of constraints into independent subsets, where each subset consists of constraints that may fire each other, and to check each component separately for weak acyclicity.

Definition 3.8 (Precedence relation [DNR08]). Given a set of constraints Σ and two constraints $r_1, r_2 \in \Sigma$, we say that $r_1 \prec r_2$ iff there exists a relational database instance K and two homomorphisms h_1 and h_2 such that

- i) $K \xrightarrow{r_1, h_1} J$,
- ii) $J \not\models h_2(r_2)$ and
- iii) $K \models h_2(r_2)$. \square

Intuitively, $r_1 \prec r_2$ means that firing r_1 can cause the firing of r_2 .

Definition 3.9 (Stratified constraints [DNR08]). The chase graph $G(\Sigma) = (\Sigma, E)$ of a set of constraints Σ contains a directed edge (r_1, r_2) between two constraints iff $r_1 \prec r_2$. We say that Σ is stratified iff the constraints in every cycle of $G(\Sigma)$ are weakly acyclic. \square

Example 3.10. [DNR08] Consider the set $\Sigma_{3,10}$ consisting of the constraint

$$r : E(x, y) \wedge E(y, x) \rightarrow \exists z \exists w E(y, z) \wedge E(z, w) \wedge E(w, x)$$

stating that each node involved in a cycle of length 2 is also involved in a cycle of length 4 and the two cycles share an edge. Since $r \not\prec r$ (i.e. r does not fire itself), $G(\Sigma_{3,10})$ is acyclic and, therefore, $\Sigma_{3,10}$ is stratified. \square

Observe that the set $\Sigma_{1.1}$ from Example 1.1 is also stratified since the chase graph $G(\Sigma_{1.1})$ contains the unique edge $r_2 \rightarrow r_1$ and, consequently, is acyclic. Indeed, $r_1 \not\prec r_2$ since the new labeled null value introduced in the second position of the predicate E cannot be present in the relation S .

In [DNR08] it has been shown that the problem of deciding whether a set of constraints is stratified is in $co\mathcal{NP}$ and that stratification strictly generalizes weak acyclicity criterion.

C-stratification

Stratification guarantees, as shown in [MSL09b], that, for every database D , there is a chase sequence (but not all) which terminates in polynomial time in the size of D . The following example shows such a case.

Example 3.11. Consider the following set of constraints $\Sigma_{3.11}$:

$$\begin{aligned} r_1 &: N(x) \rightarrow \exists y E(x, y) \\ r_2 &: N(x) \rightarrow E(x, x) \\ r_3 &: E(x, y) \wedge E(x, x) \rightarrow N(x) \end{aligned}$$

$\Sigma_{3.11}$ is stratified since $r_3 \prec r_1$, $r_3 \prec r_2 \prec r_3$ (but $r_1 \not\prec r_3$) and the set of constraints $\{r_2, r_3\}$ is weakly acyclic. Moreover, assuming that the database only contains the tuple $R(a)$, the chase firing repeatedly r_1, r_2 and r_3 never terminates ($R(a), E(a, \eta_1), E(a, a), R(\eta_1), \dots$), while the chase which never fires r_1 terminates successfully. \square

In order to cope with this problem, a variation of stratification, called *c-stratification* criterion (*CStr*), has been proposed by [MSL09b]. Basically, c-stratification defines a different chase graph and applies a constraint whenever its body is satisfied (i.e. it uses the oblivious naive chase).

Definition 3.12 (C-Stratified constraints [MSL09b]). *Given two constraints $r_1, r_2 \in \Sigma$, we say that $r_1 \prec_c r_2$ iff there exists a relational database instance K and two homomorphisms h_1 and h_2 such that:*

- i) $K \xrightarrow{*, r_1, h_1} J$,
- ii) $J \not\models h_2(r_2)$ and
- iii) $K \models h_2(r_2)$

The c-chase graph $G_c(\Sigma) = (\Sigma, E)$ of a set of constraints Σ contains a directed edge (r_1, r_2) between two constraints iff $r_1 \prec_c r_2$. We say that Σ is c-stratified iff the constraints in every cycle of $G_c(\Sigma)$ are weakly acyclic. \square

Clearly, the class of c-stratified constraints is strictly included in the set of stratified ones. Considering the two previous examples we have that the set of constraints $\Sigma_{3.11}$ in Example 3.11 is stratified, but not c-stratified, whereas the set of Constraints $\Sigma_{3.10}$ of Example 3.10 is c-stratified.

The problem of checking whether a set of constraints is c-stratified is in $co\mathcal{NP}$ (as well as stratification). As well as weak acyclicity, c-stratification guarantees that for every database D there exists a polynomial in the size of D that bounds the length of every chase sequence of D with Σ [MSL09b].

Safety

A different extension of weak acyclicity, called *safety criterion (SC)*, which takes into account only affected positions has been proposed in [MSL09a]. An affected position denotes a position which could be associated with null values, that is it can also take values from *Nulls*.

Definition 3.13 (Affected positions [CGK08]). *Let Σ be a set TGDs. The set of affected positions $aff(\Sigma)$ of Σ is defined as follows. Let R_i be a position occurring in the head of some TGD $r \in \Sigma$, then*

- *if an existentially quantified variable appears in R_i , then $R_i \in aff(\Sigma)$;*
- *if the same universally quantified variable x appears both in position R_i and only in affected positions in the body of r , then $R_i \in aff(\Sigma)$. \square*

Definition 3.14 (Safe set of TGDs [MSL09a]). *Let Σ be a set of TGDs, then $prop(\Sigma) = (aff(\Sigma), E)$ denotes the propagation graph of Σ defined as follows. For every TGD $\phi(\mathbf{x}, \mathbf{z}) \rightarrow \exists \mathbf{y} \psi(\mathbf{x}, \mathbf{y})$ and for every x in \mathbf{x} occurring in ϕ in position R_i then*

- *if x occurs only in affected positions in ϕ then for every occurrence of x in ψ in position S_j there is an edge $R_i \rightarrow S_j$ in E ;*
- *if x occurs only in affected positions in ϕ then, for every y in \mathbf{y} and for every occurrence of y in ψ in position S_j there is a special edge $R_i \xrightarrow{*} S_j$ in E .*

A set of constraints Σ is said to be safe if the corresponding propagation graph $prop(\Sigma)$ has no cycles going through a special edge. \square

Consider again the set $\Sigma_{1.1}$ from Example $\Sigma_{1.1}$. It is safe since the unique affected position is E_2 and the propagation graph $prop(\Sigma_{1.1})$ does not have any edge. Remember that $\Sigma_{1.1}$ is stratified but not weakly acyclic.

On the other hand, the stratified set $\Sigma_{3.10}$ is not safe as all positions are affected and the associated propagation graph contains cycles with special edges. The following example presents a set of safe constraints which is not stratified.

Example 3.15. Let $\Sigma_{3.15}$ be the set of below constraints:

$$\begin{aligned} r_1 &: S(x), E(x, y), E(y, z) \rightarrow \exists w E(w, x) \\ r_2 &: E(x, y), E(y, z) \rightarrow S(z) \end{aligned}$$

stating that each special node having an outgoing path of length 2 has an incoming edge (r_1) and that each path of length 2 ends in a special node (r_2). Since $aff(\Sigma_{3.15}) = \{E_1\}$ the propagation graph does not contain any edge and, therefore, $\Sigma_{3.15}$ is safe. Observe that $\Sigma_{3.15}$ is not stratified since $r_1 \prec r_2$, $r_2 \prec r_1$ and the dependency graph of $\{r_1, r_2\}$ contains cycles with special edges. \square

Clearly, safety criterion strictly generalizes weak acyclicity criterion, is not comparable with (c-)stratification (see examples 3.10 and 3.15), and the problem of checking whether a set of TGDs is safe is polynomial in the size of $|\Sigma|$. Moreover, for every Σ being the union of a safe set of TGDs with a set of EGDs, and for every database instance D , there exists a polynomial in the size of D that bounds the length of every chase sequence of D with Σ .

Safe Restriction

A more refined extension of c-stratification and safety criteria has been proposed in [MSL09a, MSL09b] under the name of *safe restriction* (*SR*) criterion. Basically, safe restriction refines stratification by considering constraints firing and possible propagation of null values together.

In order to introduce this concept we need some further definitions. For any set of positions P and a TGD r , $aff(r, P)$ denotes the set of positions π from the head of r such that i) for every universally quantified variable x in π , x occurs in the body of r only in positions from P or ii) π contains an existentially quantified variable.

For any $r_1, r_2 \in \Sigma$ and $P \subseteq pos(\Sigma)$, $r_1 \prec_P r_2$ if 1) $r_1 \prec_c r_2$ (i.e. there exists a database instance K and two homomorphisms h_1 and h_2 such that i) $K \xrightarrow{\tau_1, h_1} J$, ii) $J \not\models h_2(r_2)$ and iii) $K \models h_2(r_2)$) and 2) there is null value propagated from the body to the head of $h_2(r_2)$ s.t. it occurs in K only in positions from P .

Definition 3.16 (Safe restriction [MSL09a, MSL09b]). A 2-restriction system is a pair $(G'(\Sigma), P)$, where $G'(\Sigma) = (\Sigma, E)$ is a directed graph and $P \subseteq pos(\Sigma)$ such that:

- for all $(r_1, r_2) \in E$: if r_1 is TGD, then $aff(r_1, P) \cap pos(\Sigma) \subseteq P$, whereas if r_2 is TGD, then $aff(r_2, P) \cap pos(\Sigma) \subseteq P$, and
- $r_1 \prec_P r_2 \Rightarrow (r_1, r_2) \in E$.

Σ is called *safely restricted* if and only if there is a restriction system $(G'(\Sigma), P)$ for Σ such that every strongly connected component in $G'(\Sigma)$ is safe. \square

A 2-restriction system is *minimal* if it is obtained from $((\Sigma, \emptyset), \emptyset)$ by a repeated application of the constraints from bullets one and two (until all constraints hold) s.t., in case of the first bullet, P is extended only by those positions that are required to satisfy the condition. In [Mei10] it has been shown that Σ is safely restricted if and only if every strongly connected component in $G'(\Sigma)$ is safe, where $(G'(\Sigma), P)$ is the minimal 2-restriction system for Σ .

Example 3.17. [MSL09A] Consider the below set of constraints $\Sigma_{3.17}$:

$$\begin{aligned} r_1 &: S(x) \wedge E(x, y) \rightarrow E(y, x) \\ r_2 &: S(x) \wedge E(x, y) \rightarrow \exists z E(y, z) \wedge E(z, x) \end{aligned}$$

asserting that each special node with an outgoing edge has cycles of length 2 and 3, respectively. As position S_1 is not affected the insertion of nulls in position E_1 does not contribute to introduce further tuples with null values. Assuming $P = \{E_1, E_2\}$ we have that $r_1 \not\prec_P r_1$, $r_1 \not\prec_P r_2$, $r_2 \prec_P r_1$ and $r_2 \not\prec_P r_2$. Thus, $G'(\Sigma_{3.17}) = (\{r_1, r_2\}, \{(r_2, r_1)\})$ and, consequently, $\Sigma_{3.17}$ is safely restricted. \square

It is worth noting that $\Sigma_{3.17}$ is neither safe (since the propagation graph $prop(\Sigma_{3.17})$ has a cycle $E_2 \xrightarrow{*} E_2$ nor c-stratified (since the chase graph $G_c(\Sigma_{3.17})$ has a cycle $r_1 \prec_c r_2 \prec_c r_1$).

Inductive Restriction

Safely restriction has been extended into a criterion called *inductive restriction* (IR), whose main idea is to decompose a given constraint set into smaller subsets (in a more refined way than safe restriction). In particular, IR first computes the system $(G'(\Sigma), P)$ and partition Σ into $\Sigma_1, \dots, \Sigma_n$, where each Σ_i is a set of constraints defining a strongly connected components in $G'(\Sigma)$, next, if $n = 1$ the safety criterion is applied to Σ , otherwise the IR criterion is applied inductively to each Σ_i .

Example 3.18. Assume to add to $\Sigma_{3.17}$ the constraint $r_3: \rightarrow \exists x \exists y S(x) \wedge E(x, y)$. The new set of constraints, denoted as $\Sigma_{3.18}$, is not safely restricted, but is inductively restricted since by partitioning $\Sigma_{3.18}$ into strongly connected components we obtain the two components $\{r_3\}$ and $\{r_1, r_2\}$ which are both safely restricted. \square

The problem of checking whether a set of constraints is inductively restricted is in $co\mathcal{NP}$. As well as c-stratification and safety, inductive restriction guarantees that for every database D there exists a polynomial in the size of D that bounds the length of every chase sequence of D with Σ [MSL09b]. In that work it has been also shown that $\mathcal{CStr} \subsetneq \mathcal{SR} \subsetneq \mathcal{IR}$ and $\mathcal{SC} \subsetneq \mathcal{SR}$.

Inductive restriction has been further extended by considering not only the relationships among pairs of constraints, but general sequences of m constraints, with $m \geq 2$ [MSL09a]. The use of sequences of $m \geq 2$ constraints allows a hierarchy of classes where each class is characterized by m and denoted by $\mathcal{T}[m]$, with $\mathcal{T}[2] = \mathcal{IR}$ and $\mathcal{T}[m] \subsetneq \mathcal{T}[m+1]$.

Example 3.19. The set of constraints $\Sigma_{3.19}^n$ consisting of the TGD

$$S(x) \wedge E(x, y_1, \dots, y_n) \rightarrow E(x, y_1, \dots, y_n, z)$$

belongs to $\mathcal{T}[n+1] - \mathcal{T}[n]$. \square

In the following we do not further discuss the \mathcal{T} -hierarchy as i) the complexity of checking whether a set of constraints is in $\mathcal{T}[m]$ increases, with respect to \mathcal{IR} , with a factor $|\Sigma|^m$; ii) we do not have any criterion to fix m , and iii) the same extension can be also defined for other stratification-based criteria such as the ones proposed in this thesis.

Super-weak Acyclicity

Super-weak acyclicity (*SwA*) [Mar09] builds a *trigger graph* $\Upsilon(\Sigma) = (\Sigma, E)$ where edges define relations among constraints. An edge $r_i \rightsquigarrow r_j$ means that a null value introduced by a constraint r_i is propagated (directly or indirectly) into the head of r_j .

Let Σ be a set of TGDs and let $sk(\Sigma)$ be the logic program obtained by skolemizing Σ , i.e. by replacing each existentially quantified variable y appearing in the head of a TGD r by the skolem function $f_y^r(\mathbf{x})$, where \mathbf{x} is the set of variables appearing both in the body and in the head of r . A *place* is a pair (a, i) where a is an atom of $sk(\Sigma)$ and $0 \leq i \leq ar(a)$. Given a TGD r and an existential variable y in the head of r , $Out(r, y)$ denotes the set of places (called *output places*) in the head of $sk(r)$ where a term of the form $f_y^r(\mathbf{x})$ occurs. Let r be a TGD r and let x be a universal variable of r , $In(r, x)$ denotes the set of places (called *input places*) in the body of r where x occurs.

Example 3.20. Consider the below set of TGDs $\Sigma_{3.20}$

$$\begin{aligned} r_1 &: N(x) \rightarrow \exists y \exists z E(x, y, z) \\ r_2 &: E(x, y, y) \rightarrow N(y) \end{aligned}$$

The logic program obtained by skolemizing $\Sigma_{3.20}$ is:

$$P(\Sigma_{3.20}) = \begin{cases} r'_1 : S(x) \rightarrow E(x, f_y^{r_1}(x), f_z^{r_1}(x)) \\ r'_2 : E(x, y, y) \rightarrow S(y) \end{cases}$$

and $Out(r_1, y) = \{p_3\}$, $Out(r_1, z) = \{p_4\}$, $In(r_2, y) = \{p_6, p_7\}$. \square

Given a set of variables V , a substitution θ of V is a function mapping each $v \in V$ to a finite term $\theta(v)$ built upon constants and function symbols. Two places (a, i) and (a', i) are *unifiable*, denoted as $(a, i) \sim (a', i)$, iff there exist two substitutions θ and θ' of (respectively) the variables a and a' such that $\theta(a) = \theta'(a')$. Given two sets of places Q and Q' we write $Q \sqsubseteq Q'$ iff for all $q \in Q$ there exists some $q' \in Q'$ such that $q \sim q'$.

For any set Q of places, $Move(\Sigma, Q)$ denotes the smallest set of places Q' such that $Q \subseteq Q'$, and for every constraint $r = B_r \rightarrow H_r$ in $sk(\Sigma)$ and every variable x , if $\Pi_x(B_r) \sqsubseteq Q'$ then $\Pi_x(H_r) \subseteq Q'$, where $\Pi_x(B_r)$ and $\Pi_x(H_r)$ denote the sets of places in B_r and H_r where x occurs.

Definition 3.21 (Trigger graph and Super-weak Acyclicity [Mar09]). Given a set Σ of TGDs and two TGDs $r, r' \in \Sigma$, we say that r triggers r' in Σ and we write $r \rightsquigarrow r'$ iff there exists an existential variable y in the head of r , and a universal variable x' occurring both in the body and head of r' such that $In(r'x') \sqsubseteq Move(\Sigma, Out(r, y))$. A set of constraints Σ is super-weakly acyclic iff the trigger graph $\Upsilon(\Sigma) = (\Sigma, \{(r_1, r_2) | r_1 \rightsquigarrow r_2\})$ is acyclic. \square

Example 3.20 (cont.) Since $Move(\Sigma_{3.20}, Out(r_1, y)) = \{p_3\}$, $Move(\Sigma_{3.20}, Out(r_1, z)) = \{p_4\}$ and $In(r_2, y) = \{p_6, p_7\}$, we have that $In(r_2, y) \not\sqsubseteq Move(\Sigma_{3.20}, Out(r_1, y))$ and $In(r_2, y) \not\sqsubseteq Move(\Sigma_{3.20}, Out(r_1, z))$. Consequently, r_2 is not triggered by r_1 (as well as $r_1 \not\rightsquigarrow r_1$) and, therefore, $\Sigma_{3.20}$ is super-weakly acyclic. \square

Observe that the set of constraints $\Sigma_{3.20}$ is also c-stratified (the activation of r_1 cannot fire r_2 since two different nulls are introduced in positions E_2 and E_3), but it is not safe as all positions are affected and the propagation graph contains cycles with special edges.

With respect to other criteria, *SwA* also takes into account the fact that a variable may occur more than once in the same atom. *SwA* extends *WA* and guarantees the termination of all chase sequences in polynomial time in the size of the input database. Moreover, it has been proved in [Mar09] that the problem of deciding whether a set of constraints is super-weakly acyclic is in PTIME.

3.4 Relationship among Chase Termination Conditions

We now analyze more deeply the relationship among the criteria proposed in the literature. The relationship among *WA*, *Str*, *CStr*, *SC*, *SR* and *IR* have already been investigated in [DNR08] and [MSL09b]. In particular, it has been shown that

- $WA \subsetneq SC$, $WA \subsetneq CStr$ and $CStr \not\parallel SC^1$, i.e. *CStr* and *SC* criteria both generalize *WA* criterion, but they are not comparable,
- $CStr \subsetneq SR$, $SC \subsetneq SR$ and $SR \subsetneq IR$, i.e. *CStr* and *SC* criteria are generalized by *SR* criterion. Obviously $CStr \subsetneq Str$.

Let us start by an observation about the relation $CStr \subsetneq SR$ by means of an example.

Example 3.22. Consider the below set of TGDs $\Sigma_{3.22}$ derived from $\Sigma_{3.10}$ by adding constraint r_2 :

$$\begin{aligned} r_1 : E(x, y) \wedge E(y, x) \rightarrow \exists z \exists w E(y, z) \wedge E(z, w) \wedge E(w, x) \\ r_2 : E(x, y) \rightarrow T(x, y) \end{aligned}$$

¹ The notation $A \not\parallel B$ is a shorthand for $A \not\subseteq B$ and $A \not\supseteq B$.

$\Sigma_{3.22}$ is c-stratified since $G_c(\Sigma_{3.10})$ is acyclic. On the other hand, since $r_1 <_{\emptyset} r_2$, the minimal 2-restriction system is $(G'(\Sigma_{3.22}), P)$, where $P = \{E_1, E_2, T_1, T_2\}$, graph $G'(\Sigma_{3.22})$ contains the unique edge $\{(r_2, r_1)\}$ and its strongly connected components are $\{r_1\}$ and $\{r_2\}$. Since $\{r_1\}$ is not safe, $\Sigma_{3.22}$ is not safely restricted. \square

The problem is that we should consider just *nontrivial* components (components containing at least one edge, i.e. cyclic components), as acyclic ones cannot induce infinite chase sequences. Although the formal definition of safe restriction refers to components, probably the authors referred to cyclic components. Therefore, from now on we assume that Σ is safely restricted if and only if every nontrivial strongly connected component in $G'(\Sigma)$ is safe, where $(G'(\Sigma), P)$ is the minimal 2-restriction system for Σ .

We now analyze the relationship between the above discussed classes and $\mathcal{S}wA$. Firstly, consider again the set of constraints $\Sigma_{3.20}$ of Example 3.20. The set $\Sigma_{3.20}$ is super-weakly acyclic, but not safe, consequently $\mathcal{S}wA \not\subseteq \mathcal{S}C$, that is $\mathcal{S}C$ criterion is not more general than $\mathcal{S}wA$ criterion. Therefore, the question is: “does $\mathcal{S}C \subseteq \mathcal{S}wA$ ”? In order to present the following results we need to introduce some notation.

For any place $p_j = (a, i)$, where a is an atom of the form $A(x_1, \dots, x_n)$ occurring in a constraint r and $1 \leq i \leq n$, we denote by $\Pi(p_j)$ the corresponding position A_i . Analogously, for a given set of places Q , $\Pi(Q) = \{\Pi(p_j) | p_j \in Q\}$ denotes the set of positions associated with the places in Q .

Lemma 3.23. *Let Σ be a set of TGDs, then*

1. *For every existentially quantified variable y appearing in a constraint $r \in \Sigma$, $\Pi(\text{Move}(\Sigma, \text{Out}(r, y))) \subseteq \text{aff}(\Sigma)$ holds;*
2. *For any two sets of places Q and Q' occurring in Σ , $Q \sqsubseteq Q'$ implies that $\Pi(Q) \subseteq \Pi(Q')$.* \square

Proof.

1. Let us denote by Q the set of positions in $\text{Move}(\Sigma, \text{Out}(r, y))$. We show that in the computation of Q , at each step $\Pi(Q) \subseteq \text{aff}(\Sigma)$.
 (Base case): Initially $Q = \Pi(\text{Out}(r, y))$; since y is existentially quantified we have that $Q \subseteq \text{aff}(\Sigma)$.
 (Inductive case): Assume to have a set of places Q such that $\Pi(Q) \subseteq \text{aff}(\Sigma)$ and consider a constraint $r = B_r \rightarrow H_r$ in Σ and a universally quantified variable x appearing in both B_r and H_r . If $\Pi_x(B_r) \sqsubseteq Q$ we have that $\Pi(\Pi_x(B_r)) \subseteq \text{aff}(\Sigma)$, i.e. all positions of B_r in which x appears are affected and, consequently, the positions of H_r in which x appears are affected as well, i.e. $\Pi(\Pi_x(H_r)) \subseteq \text{aff}(\Sigma)$. Therefore, assuming that the new value of Q is $Q' = Q \cup \Pi_x(H_r)$, we have that $\Pi(Q') \subseteq \text{aff}(\Sigma)$.
2. $Q \sqsubseteq Q'$ means that for all $q \in Q$ there is a $q' \in Q'$ such that $q \sim q'$. Moreover, since $q \sim q'$ implies that $\Pi(\{q\}) = \Pi(\{q'\})$ we have that $\Pi(Q) \subseteq \Pi(Q')$. \square

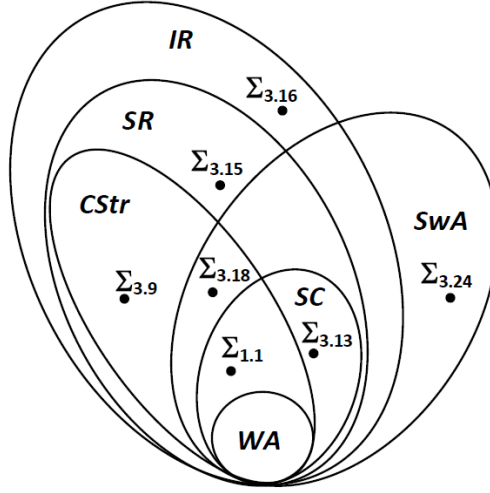


Fig. 3.1. Relationships among WA , SC , $CStr$, IR and SwA classes.

Theorem 3.24. $SC \not\subseteq SwA$. □

Proof. First of all observe that for any TGD r and for any existentially quantified variable y in r , we have that for all $R_i \in \Pi(Out(r, y))$ and for all $S_j \in \Pi(Move(\Sigma, Out(r, y)))$ there is a path $R_i \rightarrow \dots \rightarrow S_j$ in $prop(\Sigma)$, denoting that a (null) value could be propagated by means of universally quantified variables.

Assume now that there is a TGD r' (not necessarily distinct from r) such that $r' \rightsquigarrow r$. Then, there must be an existentially quantified variable z in r' and a universally quantified variable x in r such that $In(r, x) \sqsubseteq Move(\Sigma, Out(r', z))$. From Lemma 3.23 we have that $\Pi(In(r, x)) \subseteq \Pi(Move(\Sigma, Out(r', z)))$. Moreover, we also have that for every $R_i \in \Pi(In(r, x))$ and for every $S_j \in \Pi(Out(r, z))$ there is an edge $R_i \xrightarrow{*} S_j$ in $prop(\Sigma)$. Of course, we also have that for every $T_k \in \Pi(Out(r', z))$ there is a path $T_k \rightarrow \dots \rightarrow R_i$, as $\Pi(In(r, x)) \subseteq \Pi(Move(\Sigma, Out(r', z)))$.

Consequently, if there is a cycle in $\mathcal{T}(\Sigma)$, then there must be a cycle with a special edge in $prop(\Sigma)$. This implies that if $\Sigma \notin SwA$, then $\Sigma \notin SC$ as well, that is $SC \subseteq SwA$. The strict containment derives from the fact that there are sets of TGDs, such as $\Sigma_{3.20}$ in Example 3.20, which are SwA , but not SC . □

The below corollaries present two minor results which have been also independently achieved in [Mei10].

Corollary 3.25. $SwA \not\parallel CStr$. □

Proof. The set of constraints of Example 3.10 is (c-)stratified, but not super-weakly acyclic, thus $CStr \not\subseteq SwA$. Since $SC \not\subseteq CStr$ and $SC \subseteq SwA$, then $SwA \not\subseteq CStr$. Therefore, the two criteria are not comparable. □

Example 3.26. The following set of constraints $\Sigma_{3.26}$ is neither in \mathcal{IR} nor in \mathcal{SR} but it belongs to the class \mathcal{SuA} :

$$\begin{aligned} r_1 &: N(x) \rightarrow \exists y \exists z E(x, y, z) \\ r_2 &: E(x, y, z) \rightarrow T(x, y, z) \\ r_3 &: T(x, y, y) \rightarrow N(y) \end{aligned}$$

Indeed, $\Sigma'_{3.26}$ is not \mathcal{IR} (and, obviously, \mathcal{SR}) since $r_1 <_P r_2 <_P r_3 <_P r_1$, where $P = \{E_2, E_3, T_2, T_3, N_1, E_1, T_1\}$ and the unique component is not safe (i.e. $N_1 \xrightarrow{*} E_2, E_2 \rightarrow T_2, T_2 \rightarrow N_1$). Moreover, $\Sigma_{3.26}$ is in \mathcal{SuA} since the corresponding trigger graph is acyclic. \square

Corollary 3.27. $\mathcal{SuA} \not\parallel \mathcal{SR}$ and $\mathcal{SuA} \not\parallel \mathcal{IR}$. \square

Proof. Straightforward from Examples 3.26 and 3.10. \square

The previous results state that super-weak acyclicity generalizes safety and is not comparable with c-stratification and inductive restriction criteria. A complete characterization of the relationships among termination condition criteria is summed up in Figure 3.1. Notice that a set of constraints Σ_i in Figure 3.1 refers to the constraints used in the correspondent Example i .

Applications

This chapter describes several database applications in which some typical database problems arise and where the chase represents a fundamental algorithm for their solution. In every case, the termination of the chase algorithm guarantees the applicability of the proposed resolution methods.

4.1 Data Dependencies Implication Problem

The chase was first introduced in [BV84] for the *data dependencies implication problem*. Let Σ be a set of dependencies (TGDs and EGDs), and σ be a dependency. The implication problem is to decide whether $\Sigma \models \sigma$, that is to determine whether σ is true in every database in which each dependency of Σ is true (Σ *logically implies* σ). Let $SAT(\Sigma)$ be the set of all relations, only composed of constants, that satisfy all the dependencies in Σ , then the implication problem is equivalent to decide whether $SAT(\Sigma) \subseteq SAT(\sigma)$.

The implication problem can also be considered under a different view: that of *finite implication problem*. The finite implication problem is to decide whether σ is satisfied by every finite relation that satisfies all dependencies in Σ . However, as detailed below, this problem admits no proof procedure.

The chase procedure has been designed to solve the implication problem. Intuitively speaking, if the dependency to be proven is the TGD of the form $\sigma : \forall \mathbf{x} \phi_{\mathbf{r}}(\mathbf{x}, \mathbf{z}) \rightarrow \exists \mathbf{y} \psi_{\mathbf{r}}(\mathbf{x}, \mathbf{y})$, or the EGD $\sigma : \forall \mathbf{x} \phi_{\mathbf{r}}(\mathbf{x}) \rightarrow (x_1 = x_2)$, the chase procedure takes $\phi_{\mathbf{r}}(\mathbf{x}, \mathbf{z})$ (resp. $\phi_{\mathbf{r}}(\mathbf{x})$), and treats it as if it formed a set of tuples. Then it applies repeatedly the dependencies of Σ by using the chase procedure. We say that $\Sigma \models \sigma$ if there exists a homomorphism h s.t. $h(\psi_{\mathbf{r}}(\mathbf{x}, \mathbf{y})) \subseteq \text{chase}(\phi_{\mathbf{r}}(\mathbf{x}, \mathbf{z}), \Sigma)$ (if σ is a TGD), or $(x_1 = x_2)$ is true in $\text{chase}(\phi_{\mathbf{r}}(\mathbf{x}), \Sigma)$ (if σ is an EGD).

This proof procedure has been shown to be sound and complete in [Var84]. Moreover, it has been shown that the implication problem is decidable for inclusion dependencies (actually it is PSPACE-complete), but it results undecidable for functional and inclusion dependencies.

4.2 Database Design

The most important logical criteria in database design are reduction / elimination of redundancy and maintenance of database consistency. Therefore, the principle goal in relational design is to design relations that store each fact (tuple) only once in the database and remain consistent following database operations (updates, insertions and deletions). Such relations are said to be in *normal form*.

There are a number of different normal forms, e.g. for functional dependencies there are the *Third Normal Form* [Cod71, Zan82], the *Boyce-Codd Normal Form* [Cod74], the *Fourth Normal Form* [Fag77], etc. Each normal form satisfies specific criteria for that form, eliminating a different kind of redundancy. The goal is usually to get relations into the highest (or at least a high) normal form, without loss of information.

A relation is converted into a normal form by the process of *decomposition*: dividing the relation into smaller relations (i.e. relations that each have fewer attributes). This must be done in such a manner that the original relation can always be recovered (using a natural join). Such decomposition is called *lossless*. The general term for converting relations of a database into relations in a particular normal form is called *normalization*.

In the following definition, we use the symbol \bowtie to denote the natural join operator.

Definition 4.1 (Lossless join decomposition). *Let $R(Z)$ be a relation schema with attributes Z , FD a set of functional dependencies for R and $\langle R_1(X_1), \dots, R_n(X_n) \rangle$ a decomposition of R , where $X_i \subseteq Z$, for $i = 1, \dots, n$. We say that $\langle R_1(X_1), \dots, R_n(X_n) \rangle$ is a lossless join decomposition, w.r.t. FD , if for each instance r of R that satisfies FD we have $r = r[X_1] \bowtie r[X_2] \bowtie \dots \bowtie r[X_n]$. \square*

Lossless join property is necessary if the decomposed relation is to be recovered from its decomposition, and can be tested by using the chase as follows.

Let $R(Z)$ be a relation schema with attributes $Z = \{Z_1, \dots, Z_k\}$, FD a set of functional dependencies for R and $\langle R_1(X_1), \dots, R_n(X_n) \rangle$ a decomposition of R , where $X_i \subseteq Z$, for $i = 1, \dots, n$. Construct a tableau T with n rows and k columns. Set the element (i, j) of T equal to a_j if the attribute Z_j is present in the relation R_i , or equal to b_{ij} otherwise. Then, chase the table T with the constraints FD considering the a_j symbols as constants and the b_{ij} symbols as null values. At the end of the chase process, if a row of T is equal to (a_1, \dots, a_k) , then the decomposition $\langle R_1(X_1), \dots, R_n(X_n) \rangle$ is lossless.

The Third Normal Form and the Boyce-Codd Normal Form are both lossless join decompositions, while the Third Normal Form is also dependencies preserving.

4.3 Query Containment under Constraints

Query containment under constraints is the problem of checking whether for every database satisfying a given set of constraints, the result of one query is a subset of the result of another query. Containment of queries over relational databases has long been considered a fundamental problem in query optimization especially query containment under constraints such as TGDs and EGDs.

Definition 4.2 (Query containment). *Consider a relational schema \mathbf{R} and two queries Q_1 and Q_2 expressed over \mathbf{R} . We say that Q_1 is contained in Q_2 , denoted as $Q_1 \subseteq Q_2$, if for every database D we have that $Q_1(D)$ is contained in $Q_2(D)$. \square*

We say that the queries Q_1 and Q_2 are equivalent, $Q_1 \equiv Q_2$, if both $Q_1 \subseteq Q_2$ and $Q_2 \subseteq Q_1$ holds.

The following algorithm tests query containment in the case of conjunctive queries.

Let Q_1 and Q_2 be two conjunctive queries.

1. freeze $body(Q_1)$ and $head(Q_1)$ by turning each variable into a distinct (fresh) constant;
2. evaluate Q_2 over the frozen body of Q_1 ;
3. $Q_1 \subseteq Q_2$ iff the evaluation returns the frozen head of Q_1 .

However, checking query containment in the presence of constraints becomes more difficult, as it is needed to reason on the constraints imposed by the schema.

Definition 4.3 (Query containment under constraints). *Consider a relational schema \mathbf{R} , a set Σ of constraints on \mathbf{R} , and two queries Q_1 and Q_2 expressed over \mathbf{R} . We say that Q_1 is contained in Q_2 under Σ , denoted $Q_1 \subseteq_{\Sigma} Q_2$, if for every database instance K for \mathbf{R} such that $K \models \Sigma$ we have $Q_1(D) \subseteq Q_2(D)$. \square*

The following theorem shows that the chase is used to reduce the conjunctive query containment under tuple generating dependencies to classical query containment.

Theorem 4.4. *[JK84, DNR08] Consider a relational schema \mathbf{R} , a set Σ of TGDs on \mathbf{R} , and two conjunctive queries Q_1, Q_2 on \mathbf{R} . We have that $Q_1 \subseteq_{\Sigma} Q_2$ iff $h(head(Q_1)) \in Q_2(chase(h(body(Q_1)), \Sigma))$, where h is a homomorphism that maps every distinct variable in Q_1 into a distinct fresh constant. \square*

The idea is that, once we freeze Q_1 we are constructing a generic database that provides an answer to Q_1 ; but this database must satisfy the set of constraints Σ and we do that by constructing the chase of the frozen query.

Unfortunately, the previous result can be applied only when the chase is terminating for the set of embedded dependencies Σ . To test containment of conjunctive queries under IDs alone or *key-based dependencies* (a special class of FDs and IDs that is more general than the combination of key and foreign key dependencies), Johnson and Klug proved that it is sufficient to consider a *finite* portion of the chase; this leads to the decidability of the problem of containment, and it is also shown that the complexity of the problem of testing containment is PSPACE-complete. The problem is undecidable for general functional dependencies and inclusion dependencies. Other decidable classes of dependencies have been studied in the last ten years in the context of *query answering on incomplete databases*; in fact, query containment and answering under constraints have been shown to be tightly connected (see Section 4.5), as the implication problem can be reduced to both problems.

4.4 Query Optimization

The query optimization problem consists in finding a query Q_m equivalent to a given query Q such that Q_m has a minimal execution cost with respect to the original query Q .

The *query reformulation problem* is defined as follows: given a query Q_1 and a set of constraints Σ , decide whether there exists a query Q_2 such that $Q_1 \equiv_{\Sigma} Q_2$. Of course, there may be infinitely many queries Q_2 equivalent to Q_1 , and then, since we want to actually compute a Q_2 when it exists, we choose among all possible Q_2 by using cost criteria. The *query minimization problem* consists, instead, in searching for an equivalent query Q_2 that satisfy some syntactically determined minimality condition.

In the case in which we deal with conjunctive queries and the constraints are TGDs and EGDs, the *chase and backchase* algorithm has been proposed to solve the query minimization problem [DPT99, PDST00, DPT06]. In this case the *minimality criterium* adopted is the following: a conjunctive query Q is Σ -minimal if there are no queries P_1 and P_2 where P_1 is obtained from Q by replacing zero or more variables with other variables of Q , and P_2 by dropping at least one atom from P_1 such that $Q \equiv_{\Sigma} P_1 \equiv_{\Sigma} P_2$.

The chase and backchase algorithm proceeds in two phases:

1. in the *chase phase*, the original query Q is chased with the constraints in Σ , yielding the query U called a *universal plan*;
2. then the *backchase phase* enumerates all Σ -minimal subqueries SQ of U such that $SQ \equiv_{\Sigma} Q$.

The backchase phase is so called because it needs to test $SQ \equiv_{\Sigma} Q$ and then query containment by using again the chase procedure.

The applicability of the backchase algorithm requires that the chase with the set of constraints Σ is terminating.

4.5 Query Answering on Incomplete Data

Let $\mathcal{R} = \langle \mathbf{R}, \Sigma_{\mathbf{R}} \rangle$ be a relational schema where a set of constraints $\Sigma_{\mathbf{R}}$ is defined over \mathbf{R} , and let D be a database for \mathcal{R} , i.e. a database over the schema \mathbf{R} . Traditionally, the database theory essentially specifies a single database instance for such a schema \mathcal{R} . This means assuming that each relation R in \mathbf{R} has to be considered exact, i.e., given a database instance D consistent with \mathcal{R} , R is satisfied by exactly the tuples that satisfy R in D .

On the other hand, different assumptions can be adopted for interpreting the tuples that D assigns to relations in \mathbf{R} with respect to tuples that actually satisfy \mathcal{R} . In particular, tuples in D can be considered a subset or a superset of the tuples that satisfy \mathcal{R} , or exactly the set of tuples satisfying \mathcal{R} . These interpretations give raise to three different semantics, called *sound*, *complete*, and *exact*, respectively.

More specifically, the semantic of \mathcal{R} w.r.t. D , denoted $sem_x(\mathcal{R}, D)$, where $x \in \{s, c, e\}$ (for sound, complete and exact semantics, respectively) is the set of all databases B for \mathcal{R} such that:

1. B is a database consistent with \mathcal{R} , i.e. a database over \mathbf{R} that satisfies the constraints in $\Sigma_{\mathbf{R}}$;
2. D satisfies the assumptions specified on D , namely:
 - $B \supseteq D$ when $x = s$, i.e. D is a sound, but not complete database;
 - $B \subseteq D$ when $x = c$, i.e. D is a complete, but not sound database;
 - $B = D$ when $x = e$, i.e. D is a consistent database.

In this section we deal with the problem of conjunctive query answering on incomplete databases, where integrity constraints are TGDs (and EGDs). In addition, we also assume that the database is sound, i.e. it satisfies the EGDs specified by the schema (the chase does not fail). In this case, if the TGDs are not satisfied, we may add suitable facts to the database in order to satisfy them (according to the sound semantics we cannot delete facts from the database to solve such violations). Of course, the new tuple added to the database must be consistent w.r.t. the EGDs.

The (decisional) problem of conjunctive query answering on incomplete data is the following: given a relational schema $\mathcal{R} = \langle \mathbf{R}, \Sigma \rangle$ where Σ is a set of TGDs and EGDs on R , a database instance D , that may not satisfy the TGDs in Σ , a conjunctive query Q and a tuple of values t , is t an answer to Q in every database instance $B \in sem_s(\mathcal{R}, D)$?

The set of all ground atoms t such that for every $B \in sem_s(\mathcal{R}, D)$, $t \in Q(B)$ holds is denoted as $Q(D, \Sigma)$.

Theorem 4.5. [DNR08] Consider a relational schema $\mathcal{R} = \langle \mathbf{R}, \Sigma \rangle$ where Σ is a set of TGDs on \mathbf{R} , and an atom t ; we have that $t \in Q(D, \Sigma)$ iff $\text{chase}(D, \Sigma) \models t$. \square

This important result holds because the (possibly infinite) chase is a universal solution and then a representative of all the databases in $\text{sem}_s(\mathcal{R}, D)$.

The following well-known result shows the tight connection between the conjunctive query containment and the conjunctive query answering in incomplete data.

Theorem 4.6. Under TGDs, the (decisional) problem of conjunctive query answering on incomplete data, and the problem of conjunctive query containment are mutually PTIME-reducible. \square

In this scenario, conditions guaranteeing the termination of the chase algorithm over a set of TGDs for all database instances become a useful tool to allow query answering on incomplete databases. Recently, very interesting results have been exhibited for classes of TGDs for which the problem is decidable even if the chase does not halt, since it is sufficient to consider only a finite portion of the chase [CLR03, CGK08, CGL09, CGP10]:

- inclusion dependencies,
- *guarded TGDs*, that is TGDs characterized by the presence of a *guard*, namely an atom in the body that contains all the (universally quantified) variables in the TGD body;
- *weakly guarded TGDs*, that are characterized by the presence of a *weak guard* containing all the (universally quantified) variables in the TGD body appearing in affected positions;
- *linear TGDs*, that is TGDs having exactly one atom in the body and one body and one in the head; they are the class which is closest to inclusion dependencies, and they correspond to inclusion dependencies with repetition of columns.
- *sticky sets of TGDs*, which are sets of TGDs with a restriction on multiple occurrences of variables (including joins) in the TGD bodies.

Adding equality generating dependencies to the constraint may in general lead to undecidability of query answering, in fact the problem is undecidable already for inclusion dependencies and key dependencies [CLR03]. However, in order to deal with EGDs, the notions of *separability* has been proposed in [CLR03].

Definition 4.7 (Separability). Consider a set Σ_T of TGDs over a schema \mathbf{R} , and a set Σ_E of EGDs over \mathbf{R} . We say that the set $\Sigma = \Sigma_T \cup \Sigma_E$ is separable if, for every database D for \mathbf{R} , either $\text{chase}(D, \Sigma)$ fails, or, $\text{chase}(D, \Sigma) \models Q$ iff $\text{chase}(D, \Sigma_T) \models Q$, for every boolean conjunctive query Q over \mathbf{R} . \square

In other words, if the property of separability holds, EGDs do not play any role in query answering, and queries can be answered by considering the TGDs only (apart from an initial check whether the chase fails). Some examples of separable classes of TGDs and EGDs are *non-key-conflicting (NKC) IDs* [CLR03] and *non-key-conflicting TGDs* [CGL09]. Further classes have been defined in [CGP10, CP11].

4.6 Data Integration

The task of a *data integration system* is to combine data residing at different sources, providing the user with a unified view of them, called *global schema*. User queries are formulated over the global schema, and the system suitably queries the sources, providing an answer to the user, who is not obliged to have any information about the sources.

A central aspect of query processing is the specification of the relationship between the global schema and the sources; such a specification is given in the form of a so-called *mapping*.

More formally, an *information integration system* \mathcal{I} is a triple $\langle \mathcal{G}, S, \mathcal{M} \rangle$, where $\mathcal{G} = \langle G, \Sigma_G \rangle$ is the global schema specified by a set of relations G and a set of integrity constraints Σ_G on G , S is the source schema and \mathcal{M} is the mapping between S and G .

The global schema is a representation of the domain of interest of the data integration system: integrity constraints are expressed on such a schema to enhance its expressiveness, thus improving its capability of representing the real world. Regarding the mapping, there are basically two approaches for specifying it. The first approach, called *global-as-view (GAV)*, requires that a view over the sources is associated with every relation of the global schema. Conversely, the second approach, called *local-as-view (LAV)*, requires the sources to be defined as views over the global schema [Len02, DL97]. A GAV schema mapping is specified by a set of TGDs of the form

$$\forall \mathbf{x} \forall \mathbf{z} \phi_s(\mathbf{x}, \mathbf{z}) \rightarrow U(\mathbf{x})$$

where $\phi_s(\mathbf{x}, \mathbf{z})$ is a conjunction of atoms over the source schema S and U is a relation of the global schema \mathcal{G} , while a LAV schema mapping is specified by a set of TGDs of the form

$$\forall \mathbf{x} \forall \mathbf{z} R(\mathbf{x}, \mathbf{z}) \rightarrow \exists \mathbf{y} \psi_G(\mathbf{x}, \mathbf{y})$$

where $\psi_G(\mathbf{x}, \mathbf{y})$ is a conjunction of atoms over the global schema \mathcal{G} and R is a relation of the source schema S .

Sources are in general autonomous and the data that they provide are likely not to satisfy the constraints on the global schema. For this reason, the integrity constraints Σ_G have to be taken into account during query processing; otherwise, the system may return incorrect answers to the user

[FKMP05, CCGL04]. Another significant issue is that the sources may not provide exactly the data that satisfy the corresponding portion of the global schema; in particular, they may provide either a subset or a superset of the data satisfying the mentioned portion, and the mapping is to be considered *sound* or *complete* respectively. Mappings that are both sound and complete are called *exact*.

The semantics for a data integration system \mathcal{I} is the set of all databases that satisfy \mathcal{I} , i.e., the logical models of \mathcal{I} . Given a source database D on the source schema S (called a *source database* for \mathcal{I}), the set of all databases for G that satisfy \mathcal{I} relative to D is denoted as $sem_x(\mathcal{I}, D)$, where $\mathbf{x} \in \{s, c, e\}$ (for sound, complete and exact semantics, respectively), and consists of all the databases B that satisfy \mathcal{G} and that satisfy \mathcal{M} w.r.t. D .

Let \mathcal{I} be a data integration system and D a source database for \mathcal{I} , we call *retrieved global database*, denoted $ret(\mathcal{I}, D)$, the global database obtained by evaluating each query of the mapping \mathcal{M} on D , i.e. $ret(\mathcal{I}, D) = chase(D, \mathcal{M})$. Observe that $ret(\mathcal{I}, D)$ is not necessarily consistent with \mathcal{G} .

The semantics for queries is the following. Given a source database D for \mathcal{I} we call *answers* to a query Q of arity n w.r.t. \mathcal{I} and D , the set $Q(D, \mathcal{I})$ defined as follows: $Q(D, \mathcal{I}) = \{\langle c_1, \dots, c_n \rangle \mid \text{for each } B \in sem_x(\mathcal{I}, D), \langle c_1, \dots, c_n \rangle \in Q(B)\}$.

Dealing with Incomplete Data

The problem of query answering in the presence of incomplete data also arises in data integration scenarios. Query processing in LAV (without global constraints) has been traditionally considered a form of reasoning in the presence of incomplete information. In fact, sources in LAV data integration systems are generally assumed to be sound, but not necessarily complete (each view is assumed to store only a subset of the data that satisfy the corresponding view on the global schema).

Query processing in GAV (without global constraints), instead, is less involved as the form of the mapping straightforwardly allows for the computation of a global database instance, i.e. the retrieved global database, over which the query can be directly evaluated. It is worth noting that, proceeding in this way, is analogous to *unfold* queries with respect to the mapping, i.e. substitute each atom in the query with its correspondent view specified by the mapping, and evaluate the unfolded queries directly on the source database. In this case, the views in the mapping are exact. However, in GAV it may also happen that the sources are considered sound rather than exact, in particular when integrity constraints are specified on the global schema. In this case the query unfolding is no more sufficient, and reasoning on the constraint is needed in order to compute the certain answers to a query. Moreover, it is easy to see that reasoning on the constraints is also needed in the LAV approach.

4.7 Data Exchange

In data exchange [FKMP05], data structured under one schema (which we call a *source schema*) must be restructured and translated into an instance of a different schema (a *target schema*). Let \mathbf{S} be the source schema and \mathbf{T} be the target schema, we assume that \mathbf{S} and \mathbf{T} are disjoint. Moreover, since \mathbf{T} is an independently created schema, it has its own set of constraints, denoted as $\Sigma_{\mathbf{T}}$. The mapping of the data from the source schema to the target schema is expressed as a set of *source-to-target TGDs* Σ_{st} , that specifies how and what source data should appear in the target.

More formally, the *data exchange setting* is defined as follows.

Definition 4.8. A *schema mapping (or data exchange setting)* is a 4-tuple $M = (\mathbf{S}, \mathbf{T}, \Sigma_{\text{st}}, \Sigma_{\mathbf{T}})$ where:

- \mathbf{S} is the source schema,
- \mathbf{T} is the target schema,
- Σ_{st} is a set of source-to-target TGDs of the form $\phi_{\mathbf{S}}(\mathbf{x}, \mathbf{z}) \rightarrow \exists \mathbf{y} \psi_{\mathbf{T}}(\mathbf{x}, \mathbf{y})$ and,
- $\Sigma_{\mathbf{T}}$ is a set of target dependencies, whereas each target dependency in $\Sigma_{\mathbf{T}}$ is either a TGD $\phi_{\mathbf{T}}(\mathbf{x}, \mathbf{z}) \rightarrow \exists \mathbf{y} \psi_{\mathbf{T}}(\mathbf{x}, \mathbf{y})$ or an EGD $\phi_{\mathbf{T}}(\mathbf{x}) \rightarrow (x_1 = x_2)$. \square

Note that the input to a data exchange problem is a source instance only; the data exchange setting itself (i.e. source and target schemas and dependencies) is considered fixed.

The *data exchange problem* associated with a setting M is the following: given a finite source instance D over \mathbf{S} , find a finite target instance J over \mathbf{T} such that $\langle D, J \rangle$ satisfies Σ_{st} and J satisfies $\Sigma_{\mathbf{T}}$. J is called a solution for D or, simply a solution if D is understood. Observe that there may be many solutions (or none) for a given instance of the data exchange problem. The set of all the solutions is denoted by $Sol(D, \Sigma_{\text{st}} \cup \Sigma_{\mathbf{T}})$. However, a special class of solutions has been identified for the semantics of the data exchange problem, namely the class of *universal solutions* for D . This kind of solution is justified by the fact that a universal solution is representative of the space of all solutions, as it can be mapped into any other solutions. The set of all universal solutions for D is denoted by $USol(D, \Sigma_{\text{st}} \cup \Sigma_{\mathbf{T}})$. Given a source database D , it has been shown that a universal solution J for D and $\Sigma_{\text{st}} \cup \Sigma_{\mathbf{T}}$ can be computed by using the chase algorithm, i.e. $J = \text{chase}(D, \Sigma_{\text{st}} \cup \Sigma_{\mathbf{T}})$. In this case J is called a *canonical universal solution*. Moreover, it is proven that, if $\text{chase}(D, \Sigma_{\text{st}} \cup \Sigma_{\mathbf{T}})$ fails, no solution exists for the assigned data exchange problem. However, in general, for arbitrary sets of dependencies, there may not exist a finite chase. Thus, special classes of constraints for which the chase termination is guaranteed in polynomial time data complexity allows to (i) check the existence of a solution, and, (ii) eventually compute it, in polynomial time.

Moreover, among all universal solutions, the "best" universal solution was found to be the *core* [FKP05]. The core is the smallest universal solution and it has been shown that it is unique, i.e. all universal solutions have the same core (up to isomorphism). Thus, the core is the ideal solution to materialize, and, in particular, an algorithm has been proposed to compute it in polynomial time when target constraints are a weakly acyclic set of TGDs and EGDs [GN08].

The semantics of target query answering adopts the notion of *certain answers*, i.e. the answers to a query Q consist of the set of all tuples t of constants from the given source instance D such that for every solution J for the the data exchange problem, we have that $t \in Q(J)$. For the class of union of conjunctive queries it has been shown that computing certain answers to a query Q is equivalent to evaluate Q over a universal solution J and take all the tuples of constants in $Q(J)$. This result holds since conjunctive queries are preserved under homomorphisms.

New Chase Termination Conditions

In this chapter we present some improvements for termination conditions discussed in Chapter 3 and then introduce the class of *locally stratified dependencies*, that generalizes previously known classes, for which termination of the chase algorithm is guaranteed. The idea underlying stratification, also used in its variation and extensions (e.g. *CStr*, *SR*) and in the super-weak acyclicity, is to consider, in the propagation of nulls, how constraints may fire each other. However, there are simple cases where current criteria are not able to understand that all chase sequences are finite (see, for instance, examples 1.2 and 5.10). Thus, in this chapter we first introduce a new version of stratification, called *WA-stratification* (*WA-Str*) which generalizes *CStr* and guarantees, for all databases, termination of all chase sequences.

5.1 WA-Stratification

(C-)stratification does not specify what kind of cycles are checked (i.e. simple or general) [CLRS01]. Checking simple cycles is not correct as it may not consider all possible chase sequences, but checking general cycles, means that for each strongly connected component there is one cycle including all nodes in the component which subsumes all other cycles on the same component (in terms of constraints to be considered).

Example 5.1. Consider the following set of TGDs $\Sigma_{5.1}$:

$$\begin{aligned} r_1 : P(x) &\rightarrow \exists y Q(x, y) \\ r_2 : Q(x, y) &\rightarrow R(x, y) \\ r_3 : R(x, y) &\rightarrow P(x) \\ r_4 : R(x, y) &\rightarrow S(y, x) \\ r_5 : S(x, y) &\rightarrow Q(x, y) \end{aligned}$$

We have that $r_1 \prec_c r_2$, $r_2 \prec_c r_3$, $r_3 \prec_c r_1$, $r_2 \prec_c r_4$, $r_4 \prec_c r_5$ and $r_5 \prec_c r_2$. The c-chase graph contains two simple cycles, i.e. $\{r_1, r_2, r_3\}$ and $\{r_2, r_4, r_5\}$, that

are both weakly acyclic, and a general cycle involving all the TGDs in $\Sigma_{5.1}$ that is not weakly acyclic. \square

Although considering the constraints involved in every cycle is not wrong, this is equivalent to just considering the subsets of constraints involved in every strongly connected component, since if the weak acyclicity property is satisfied by a set of constraints it is satisfied by all its subsets as well. Moreover, the number of cycles in a graph could be exponential, whereas the number of strongly connected components is polynomial. Thus, a first observation on (c-)stratification (in terms of correctness, if simple cycles are considered, or in terms of efficiency, if all cycles are considered) is that it refers to cycles instead of strongly connected components. A further observation is that it uses oblivious chase for checking termination of standard chase and, as previously said, its applicability is limited.

Definition 5.2 (WA-Stratification). Given a set of dependencies Σ and $r_1, r_2 \in \Sigma$, we say that $r_1 < r_2$ iff there exist a relational database instance K , homomorphisms h_1, h_2 and a set S of atoms, such that

- (i) $K \not\models h_1(r_1)$,
- (ii) $K \xrightarrow{r_1, h_1} J$,
- (iii) $K \cup S \models h_2(r_2)$,
- (iv) $J \cup S \not\models h_2(r_2)$ and
- (v) $Null(S) \cap (Null(J) - Null(K)) = \emptyset$ (i.e. S does not contain new null values introduced in J).

We say that Σ is *WA-stratified* (*WA-Str*) iff the constraints in every nontrivial strongly connected component of the *firing graph* $\Gamma(\Sigma) = (\Sigma, \{(r_1, r_2) | r_1 < r_2\})$ are weakly acyclic. \square

With respect to stratification, *WA-Str* also considers in the satisfaction of constraint r_2 , in addition to the database K , a set of atoms S (cond. (iii)) and atoms in S cannot contain null values introduced in the application of the constraint r_1 (cond. (v)).

Moreover, since we are considering strongly connected components (instead of cycles) these components must not be trivial, that is they must have at least one edge, otherwise the constraint cannot be fired cyclically. As a further important observation, in the above definition we consider standard chase for both constructing the graph $\Gamma(\Sigma)$ and checking weak acyclicity.

Example 5.3. Consider again the set of constraints $\Sigma_{1.2}$ of Example 1.2. It is easy to see that, by considering standard chase, does not exist an initial database instance such that the constraint can fire itself, while, by considering the oblivious chase, the constraint fires itself ad infinitum. Thus, the set of constraints $\Sigma_{1.2}$ is WA-stratified, but not c-stratified. \square

The following proposition states that *WA-Str* criterion is more general than *CStr* and is not comparable with *SC*. Consequently it is not comparable even with *SwA* as *SC* is strictly contained in *SwA* and *CStr* is not comparable with *SwA*.

Proposition 5.4. $CStr \subsetneq WA-Str$ and $SC \not\parallel WA-Str$. \square

Proof. (sketch) $CStr \subseteq WA-Str$ derives from the fact that the firing graph $\Gamma(\Sigma)$ is contained in the chase graph $G_c(\Sigma)$ used by c-stratification. The containment is strict as it is possible to define constraints which are WA-stratified and do not satisfy the c-stratification criterion (see, for instance, the set of constraints $\Sigma_{1.2}$ of Example 1.2).

The relationship $SC \not\parallel WA-Str$ derives from the fact that there are examples belonging to one of these classes, but not to the other. For instance, the following set Σ is safe, but not WA-stratified:

$$\begin{aligned} \alpha &: S(x_2, x_3) \wedge R(x_1, x_2, x_3) \rightarrow \exists y R(x_2, y, x_1) \\ \beta &: R(x_1, x_2, x_3) \rightarrow S(x_1, x_3) \end{aligned}$$

On the other hand, $\Sigma_{1.2}$ is WA-stratified but not safe. \square

It is important to observe that *WA-Str* criterion could be improved by testing safety instead of weak acyclicity over the firing graph. Further improvements could be obtained by considering super-weak acyclicity instead of safety.

Definition 5.5 (SC-Stratification and SwA-Stratification). Given a set of TDGs Σ , we say that

- Σ is *SC-stratified* (*SC-Str*) if the constraints in every strongly connected component of the firing graph $\Gamma(\Sigma)$ are safe, and
- Σ is *SwA-stratified* (*SwA-Str*) if the constraints in every strongly connected component of the firing graph $\Gamma(\Sigma)$ are super-weak acyclic. \square

We now analyze the complexity of the above criteria starting by defining a bound on the complexity of the firing problem, i.e. the complexity of checking whether $r_1 < r_2$.

Lemma 5.6. Let $r_1 : \phi_1 \rightarrow A_1 \wedge \dots \wedge A_k$ and $r_2 : B_1 \wedge \dots \wedge B_n \rightarrow \psi_2$ be two TDGs. The problem of checking whether $r_1 < r_2$ is bounded by $O((k+1)^n)$. \square

Proof. (sketch) We can define a database instance D consisting of body atoms $B_1 \dots B_n$ (assuming that variables denote constants), and define a set of atoms satisfying firing conditions S and homomorphisms h_1, h_2 so that atoms in B_1, \dots, B_n unify with some of head atoms $A_1 \dots A_k$. Let us denote with $k_i = |\psi_1(B_i)| + 1$ the number of atoms in the head of r_1 unifying with the atom B_i appearing in the body of r_2 plus one (representing that B_i may not unify with any head atom). Then, we have k_1 choices for unification of B_1 , $k_1 \times k_2$

choices for the sequence B_1, B_2 and so on. Finally, for the sequence B_1, \dots, B_n we have $\prod_{i=1}^n k_i$ choices. Thus, the number of choice arcs in the exploration tree is bounded by $O(\sum_{j=1}^n (\prod_{i=0}^{j-1} k_i) \times k_j)$, where $k_0 = 1$. The complexity of defining the sets $\psi_1(B_i)$, for $i \in [1..n]$, is bounded by $n \times k$. Since each k_i is bounded by $O(k+1)$, the global complexity is $O(k \times n + (k+1)^n) = O((k+1)^n)$. \square

Although the theoretical complexity of the "firing" problem is exponential, in most cases it is very low (e.g. inclusion dependencies, multivalued dependencies [AHV95]), as usually the number n of body atoms in the fired constraint r_2 is small and the number of atoms in the head of constraint r_1 which could be used to fire r_2 through their unification with B_i (i.e. $k_i > 1$) is even smaller. Indeed, if the number of atoms in the body of r_2 is bounded by a constant, the firing problem is in PTIME. Significant subclasses of constraints for which the firing problem becomes polynomial could be identified, but this is outside the aim of this thesis.

In the following, for a given set of constraints Σ , we shall denote with C_{ij} the complexity of the problem of checking whether $r_i < r_j$, for $r_i, r_j \in \Sigma$, and with $C_m = \max\{C_{ij} | r_i, r_j \in \Sigma\}$.

Proposition 5.7. Let Σ be a set of TGDs, D be a database Then:

- the problem of checking whether Σ is WA-stratified (resp. SC-stratified, SwA-stratified) is bounded by $O(C_m \times |\Sigma|^2)$;
- if Σ is WA-stratified (resp. SC-stratified, SwA-stratified), the length of every chase sequence of Σ over D is polynomial in the size of D . \square

Proof. (sketch)

1. (i) The cost of checking whether $r_i < r_j$ is denoted by C_{ij} .
(ii) The cost of constructing the firing graph is bounded by $O(\sum_{r_i, r_j \in \Sigma} C_{ij}) = O(|\Sigma|^2 \times C_m)$ where $C_m = \max\{C_{ij} | r_i < r_j\}$.
(iii) The detection of strongly connected components in the firing graph is bounded by $O(|\Sigma|^2)$. Consequently, checking whether Σ is WA-stratified (resp. SC-stratified, SwA-stratified) is bounded by $O(C_m \times |\Sigma|^2)$.
2. Σ can be partitioned into $\Sigma_1, \dots, \Sigma_k$ with $k \leq |\Sigma|$, where each Σ_i consists of the constraints belonging to a strongly connected component of $\Gamma(\Sigma)$. For any database D , the universal solutions of (D, Σ) can be computed by taking one Σ_i at time, following the topological order of the firing graph $\Gamma(\Sigma)$. Since each Σ_i is weakly acyclic (resp. safe, super-weakly acyclic), the length of all chase sequences of Σ_i over D is polynomial in the size of D and, therefore, the length of every chase sequence of Σ over D is polynomial in the size of D . \square

The class of constraints satisfying criterion C -Str, for $C \in \{WA, SC, SwA\}$, will be denoted by \mathcal{C} -Str. The next theorem states the relationships among the above mentioned criteria and other previously defined conditions.

Theorem 5.8.

1. $WA\text{-}Str \subsetneq SC\text{-}Str \subsetneq SwA\text{-}Str$,
2. for $C \in \{WA, SC, SwA\}$, $C \subsetneq \mathcal{C}\text{-}Str$ and
3. $SR \not\parallel SwA\text{-}Str$ and $IR \not\parallel SwA\text{-}Str$. \square

Proof.

1. It follows from the fact that $WA \subsetneq SC \subsetneq SwA$.
2. It follows from the fact that the $C\text{-}Str$ criterion first divides the set of constraints Σ into subsets (strongly connected components of the firing graph) and then checks the specific criterion C on each subset.
3. It is possible to find examples of sets of constraints that are in SR (and IR) but not in $SwA\text{-}Str$ and viceversa (see Examples 5.9 and 5.10). \square

The following two examples show that both SR and IR criteria are not comparable with $SwA\text{-}Str$.

Example 5.9. Consider the below set of constraints $\Sigma_{5.9}$:

$$\begin{aligned} r_1 &: N(x) \rightarrow \exists y \exists z E(x, y) \wedge S(z, y) \\ r_2 &: E(x, y) \wedge S(x, y) \rightarrow N(y) \\ r_3 &: E(x, y) \rightarrow E(x, x) \end{aligned}$$

The set $\Sigma_{5.9}$ is SR (and, obviously IR), but not $SwA\text{-}Str$. Indeed, $r_1 < r_3 < r_2 < r_1$ and $\Sigma_{5.9}$ is not SwA . \square

Example 5.10. Consider the following set of constraints $\Sigma'_{5.10}$:

$$\begin{aligned} r_1 &: N(x) \rightarrow \exists y \exists z E(x, y, z) \\ r_2 &: E(x, y, z) \rightarrow T(x, y, z) \\ r_3 &: T(x, y, y) \rightarrow N(y) \end{aligned}$$

The set $\Sigma'_{5.10}$ is not IR (and, obviously, SR) since $r_1 <_P r_2 <_P r_3 <_P r_1$, where $P = \{E_2, E_3, T_2, T_3, N_1, E_1, T_1\}$ and the unique component is not safe (i.e. $N_1 \xrightarrow{*} E_2$, $E_2 \rightarrow T_2$, $T_2 \rightarrow N_1$). On the other hand, the below set of constraints $\Sigma''_{5.10}$

$$r_4 : R(x, y) \wedge R(y, x) \rightarrow \exists u \exists v R(x, u), R(u, v), R(v, x)$$

is not SwA , but it is stratified and, therefore, safely restricted. The set of constraints $\Sigma_{5.10} = \Sigma'_{5.10} \cup \Sigma''_{5.10}$, is neither IR , nor SwA , but it belongs to the class of $SwA\text{-}Str$ constraints. \square

5.2 Local Stratification

It is trivial that more powerful criteria could be defined by composing criteria which are not comparable. We next present a different generalization of super-weak acyclicity which also generalizes the class IR .

We start by introducing a notion of *fireable place*. We say that a place q appearing in the body of constraint r could be fired by a place q' appearing in the head of constraint r' , denoted by $q' < q$, if $q \sim q'$ and $r' < r$. Given two sets of places Q and Q' we say that Q could be fired by Q' , denoted by $Q' < Q$ iff for all $q \in Q$ there exists some $q' \in Q'$ such that $q' < q$.

Given a set Q of places, we define $MOVE(\Sigma, Q)$ as the smallest set of places Q' such that $Q \subseteq Q'$, and for every constraint $r = B_r \rightarrow H_r$ in $sk(\Sigma)$ and every variable x , if $Q' < \Pi_x(B_r)$ then $\Pi_x(H_r) \subseteq Q'$, where $\Pi_x(B_r)$ and $\Pi_x(H_r)$ denote the sets of places in B_r and H_r where x occurs.

With respect to the function *Move*, the new function *MOVE* here considered takes into account the firing of places and not only the unification of places.

Definition 5.11 (Local Stratification). Given a set Σ of TGDs and two TGDs $r_1, r_2 \in \Sigma$, we say that r_1 triggers r_2 in Σ and write $r_1 \hookrightarrow r_2$ iff there exists an existential variable y in the head of r_1 , and a universal variable x occurring both in the body and head of r_2 such that $MOVE(\Sigma, Out(r_1, y)) < In(r_2, x)$. A set of constraints Σ is *locally stratified (LS)* iff the trigger graph $\Delta(\Sigma) = \{(r_1, r_2) | r_1 \hookrightarrow r_2\}$ is acyclic. \square

Proposition 5.12. For every set of TGDs Σ and for every database D

- the problem of checking whether Σ is locally stratified is bounded by $O(C_m \times |\Sigma|^2)$;
- if Σ is locally stratified, the length of every chase sequence of Σ over D is polynomial in the size of D . \square

Proof. (sketch)

1. From the polynomial complexity of the function *Move* [Mar09] and the complexity of building the firing graph, bounded by $O(|\Sigma|^2 \times C_m)$, it follows that the complexity of building the trigger graph $\Delta(\Sigma)$ is bounded by $O(|\Sigma|^2 \times C_m)$. Since the complexity of checking whether the trigger graph $\Delta(\Sigma)$ is acyclic has a cost bounded by $O(|\Sigma|^2)$, the global complexity is bounded by $O(|\Sigma|^2 \times C_m)$.
2. The proof follows the one of Proposition 5.7. \square

The below theorem states that the class of locally stratified constraints (denoted by \mathcal{LS}) is more general than *SwA-Str* and \mathcal{IR} .

Theorem 5.13. *SwA-Str* \subsetneq \mathcal{LS} and $\mathcal{IR} \subsetneq \mathcal{LS}$. \square

Proof. (sketch) From the definitions of *SwA* and *LS* criteria it follows that i) $MOVE(\Sigma, Out(r, y)) \subseteq Move(\Sigma, Out(r, y))$ and ii) if $Q' < Q$ then $Q \sqsubseteq Q'$. As a consequence, *SwA-Str* $\subseteq \mathcal{LS}$.

Assume that Σ is *SR* and that $\langle G'(\Sigma), P \rangle$ is the associated 2-restricted system. If $r_i \hookrightarrow r_j$ (i.e. $(r_i, r_j) \in \Delta(\Sigma)$), then there exists a path from r_i

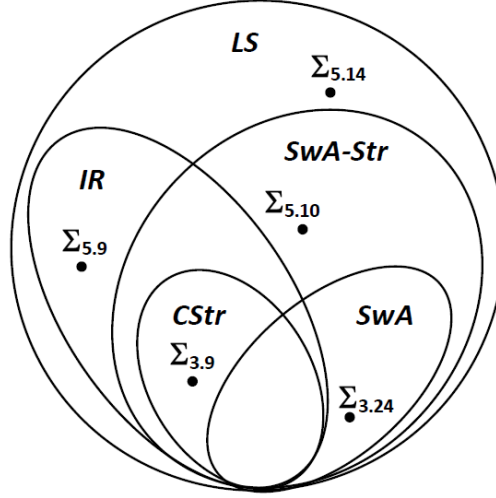


Fig. 5.1. Criteria Relationships.

to r_j in $G'(\Sigma)$ because, when we construct the set $MOVE(\Sigma, Out(r_i, y))$, we take into account both i) the firing relation $<$ between places, and ii) the propagation of null values introduced in the position associated with the existentially quantified variable y .

If the relation \leftrightarrow is cyclic, then there exists a cycle in $G'(\Sigma)$ containing the constraints involved in the cycle in $\Delta(\Sigma)$ and, consequently, if the constraints are not locally stratified, then they are neither *SwA* and nor safe. Moreover, the constraints appearing in the cycle also belong to the same partition defined by *IR*. As a consequence, $IR \subseteq LS$. To show that the containments are strict it is sufficient to consider the set of constraints $\Sigma_{5.14}$ which is in LS , but neither in *IR* nor in *SwA-Str*. \square

The next example shows that the containment of *SwA-Str* and *IR* in LS is strict, whereas Figure 5.1 resumes the relationships among the above discussed criteria.

Example 5.14. The following set of constraints $\Sigma_{5.14}$ is locally stratified, but it is neither super-weakly acyclic nor inductively restricted:

$$\begin{aligned} r_1 &: N(x) \rightarrow \exists y \exists z E(x, y) \wedge S(z, y) \\ r_2 &: E(x, y) \wedge S(x, y) \rightarrow N(y) \\ r_3 &: E(x, y) \rightarrow E(y, x) \end{aligned}$$

Considering *SwA*, we have that $Move(\Sigma, Out(r_1, y)) = \{p_3, p_5, p_{10}, p_{13}, p_2, p_{14}\}$ and $In(r_1, x) = \{p_1\} \sqsubseteq Move(\Sigma, Out(r_1, y))$. The trigger graph is cyclic as $r_1 \rightsquigarrow r_1$ and, therefore, $\Sigma_{5.14}$ is not super-weakly acyclic. As $r_1 < r_3 < r_2 < r_1$ we have that $\Sigma_{5.14}$ is not *SwA-Str* as well. $\Sigma_{5.14}$ is not *IR* as $r_1 \prec_c r_3 \prec_c$

$r_2 \prec_c r_1$ and for each pair of constraints r_i, r_j such that $r_i \prec_c r_j$, it is possible to construct a database containing null values in positions E_1, E_2, N_1 and S_2 such that whenever r_i fires r_j a null value is propagated from the head of r_i to the head of r_j .

Moreover, as $r_1 \not\prec r_1$ ($MOVE(\Sigma, (r_1, y)) = \{p_3, p_5, p_{13}\}$ and $In(r_1, x) = \{p_1\} \sqsubseteq MOVE(\Sigma, Out(r_1, y))$), $\Delta(\Sigma_{5.14})$ is acyclic and, thus, $\Sigma_{5.14}$ is locally stratified. \square

Checking Chase Termination by Constraints Rewriting

In this chapter we present a technique for checking chase termination based on rewriting the original set of TGDs Σ into an ‘equivalent’ set Σ^α and verifying the structural properties for chase termination on Σ^α . The technique performs a deep analysis of constraints by considering pattern analysis through the introduction of adornments associated with predicates. The adornments here considered are similar to those used in binding propagation in deductive databases (e.g. magic-set) for the optimization of bound queries [BR91].

6.1 Constraints Rewriting

Before presenting our rewriting technique we introduce some definitions concerning constraints equivalence. In particular, the equivalence between two sets of constraints Σ_1 and Σ_2 defined, respectively, over two schemas \mathbf{R}_1 and \mathbf{R}_2 , is given with respect to two sets of relations $R, S \subseteq \mathbf{R}_1 \cap \mathbf{R}_2$ called, respectively, input and output relations.

Definition 6.1 (Sets of constraints equivalence). *Given two sets of constraints Σ_1 and Σ_2 over the two database schemas \mathbf{R}_1 and \mathbf{R}_2 , respectively and two sets of relations $R, S \subseteq \mathbf{R}_1 \cap \mathbf{R}_2$, we say that $\langle \mathbf{R}_1, \Sigma_1 \rangle \sqsubseteq_{R/S} \langle \mathbf{R}_2, \Sigma_2 \rangle$ if for all database D over R , $USol(D, \Sigma_1)[S] \subseteq USol(D, \Sigma_2)[S]$. Moreover, we say that $\langle \mathbf{R}_1, \Sigma_1 \rangle$ and $\langle \mathbf{R}_2, \Sigma_2 \rangle$ are equivalent with respect to R/S and write $\langle \mathbf{R}_1, \Sigma_1 \rangle \equiv_{R/S} \langle \mathbf{R}_2, \Sigma_2 \rangle$ if both $\langle \mathbf{R}_1, \Sigma_1 \rangle \sqsubseteq_{R/S} \langle \mathbf{R}_2, \Sigma_2 \rangle$ and $\langle \mathbf{R}_2, \Sigma_2 \rangle \sqsubseteq_{R/S} \langle \mathbf{R}_1, \Sigma_1 \rangle$. \square*

When $R = S = \mathbf{R}_1 \cap \mathbf{R}_2$ we simply write $\langle \mathbf{R}_1, \Sigma_1 \rangle \sqsubseteq \langle \mathbf{R}_2, \Sigma_2 \rangle$ and $\langle \mathbf{R}_1, \Sigma_1 \rangle \equiv \langle \mathbf{R}_2, \Sigma_2 \rangle$.

Example 6.2. Consider the database schema $\mathbf{R}_1 = \{E(A, B)\}$ consisting of the binary relation E and the database schema $\mathbf{R}_2 = \{E(A, B), Q(C)\}$ con-

sisting of the binary relation E and the unary relation Q . Assume to have the following sets of TGDs

$$\begin{aligned}\Sigma_1 &= \{E(x, y) \rightarrow E(y, x)\} \text{ and} \\ \Sigma_2 &= \{E(x, y) \rightarrow Q(x), \quad Q(x) \wedge E(x, y) \rightarrow E(y, x)\}\end{aligned}$$

defined over \mathbf{R}_1 and \mathbf{R}_2 , respectively.

Clearly, $USol(D, \Sigma_1)[E] = USol(D, \Sigma_2)[E]$ for all databases D over $\mathbf{R}_1 \cap \mathbf{R}_2 = \{E\}$ and, therefore, $\langle \mathbf{R}_1, \Sigma_1 \rangle \equiv \langle \mathbf{R}_2, \Sigma_2 \rangle$. \square

Adornments

An adornment α of a predicate p with arity m is a string of length m over the alphabet $\{b, f\}$. A predicate symbol p^α is said to be adorned, whereas an adorned atom is of the form $p^{\alpha_1 \dots \alpha_m}(x_1, \dots, x_m)$; if $\alpha_i = b$ we say that the variable x_i is bounded, otherwise ($\alpha_i = f$) we say that x_i is free. Intuitively, bounded terms can take values from finite domains; consequently, constant terms are always adorned with the symbol b . If each body variable of a TGD is associated with a unique adornment we say that the adornment of the body is coherent. Before introducing how constraints are adorned, let us introduce some further definitions and notations.

We assume that TGDs are in standard form, that is existentially quantified variables appear within the scope of universally quantified ones; variables appearing in constraints with empty body are replaced by Skolem constants.

Given a TGD $r : \phi(\mathbf{x}) \rightarrow \exists \mathbf{y} \psi(\mathbf{z}, \mathbf{y})$ with $\mathbf{z} \subseteq \mathbf{x}$ and let α be a coherent adornment for the body atoms, then $HeadAdn(r, \phi^\alpha(\mathbf{x}))$ denotes the *adornment of the head* of r (with respect to the adorned body $\phi^\alpha(\mathbf{x})$) obtained by adorning head atoms as follows: i) every universally quantified variable has the same adornment of the body occurrences, ii) constants are adorned as b ; iii) existentially quantified variables are adorned as f .

Rewriting Algorithm

Given a set of TGDs Σ over a schema \mathbf{R} the corresponding rewriting set $Adn(\Sigma)$ consists of the union of four sets of TGDs: the base set $Base(\Sigma)$, the derived set $Derived(\Sigma)$, the input set $In(\Sigma)$ and the output set $Out(\Sigma)$.

The rewriting is performed by means of the function Adn reported in Figure 6.1. It starts by adorning, for each TGD, body predicates with strings of b symbols and adorning heads according to the body adornments by using the function $HeadAdn$ (base set); then, each new adorned predicate symbol is used to generate new adorned constraints until all adorned predicate symbols are used (derived set); at the end, TGDs mapping source relations into relations adorned with strings of b symbols (input set) and TGDs mapping relations having the same predicate and different adornments into a unique relation (output set) are added.

Function $Adn(\Sigma)$;
Input: Set of TGDs Σ over a schema \mathbf{R} ;
Output: The set of (adorned) TGDs $Base \cup Derived \cup In \cup Out$;
begin
 $Derived = \emptyset$;
// Let $Body^b(r)$ be the conjunction obtained by adorning atoms in $Body(r)$
// with strings of b symbols
 $Base = \{Body^b(r) \rightarrow HeadAdn(r, Body^b(r)) \mid \exists r \in \Sigma\}$;
 $Used_Pred = \{p_r^b \mid \exists r \in \Sigma \text{ and } \exists p^b \text{ in } Body^b(r)\}$;
 $New_Pred = \{p_r^\alpha \mid \exists r \in \Sigma \text{ and } p^\alpha(t) \in SkHeadAdn(r, Body^b(r))\} - Used_Pred$;
while ($New_Pred \neq \emptyset$) **do begin**
Select nondeterministically $p^{\alpha_1 \dots \alpha_n} \in New_Pred$;
 $New_Pred = New_Pred - \{p^{\alpha_1 \dots \alpha_n}\}$;
 $Used_Pred = Used_Pred \cup \{p^{\alpha_1 \dots \alpha_n}\}$;
for each $r \in (Base \cup Derived)$ **do**
for each $p^\beta(x_1, \dots, x_n) \in Body(r)$ **do begin**
 $B' = Body(r) - \{p^\beta(x_1, \dots, x_n)\} \cup \{p^{\gamma_1 \dots \gamma_n}(x_1, \dots, x_n)\}$;
 $\gamma_i = b$ if $x_i \in Consts$; $\gamma_i = \alpha_i$ if $x_i \in Vars$; ($i \in [..n]$);
if B' is coherent **then**
 $Derived = Derived \cup \{B' \rightarrow HeadAdn(Adn^{-1}(r), B')\}$;
 $New_Pred = New_Pred \cup \{p^\omega \mid p^\omega \text{ appears in } HeadAdn(Adn^{-1}(r), B') \wedge p^\omega \notin Used_Pred\}$;
else
 $Derived = Derived \cup \{B' \rightarrow Adn^{-1}(r)\}$;
end for;
end while;
Delete from $Derived$ constraints with unadorned heads;
 $In = \{p(x_1, \dots, x_n) \rightarrow p^{b \dots b}(x_1, \dots, x_n) \mid p(A_1, \dots, A_n) \in \mathbf{R}\}$;
 $Out = \{p^\alpha(x_1, \dots, x_n) \rightarrow \hat{p}(x_1, \dots, x_n) \mid p^\alpha(z_1, \dots, z_n) \text{ appears in } (Base \cup Derived)\}$;
return $Base \cup Derived \cup In \cup Out$;
end.

Fig. 6.1. Constraint Rewriting Function Adn

In the definition of the function Adn we have also used the function $Adn^{-1}(\cdot)$ which takes in input an adorned first order formula consisting of a conjunction of atoms or a constraint or a set of constraints and gives in output the same formula without adornments. Clearly, for any set of constraints Σ , $Adn^{-1}(Adn(\Sigma)) = \Sigma$.

For any input database schema \mathbf{R} and set of constraints Σ over \mathbf{R} , we shall denote with (i) $\hat{\mathbf{R}} = \{\hat{p}(A_1, \dots, A_n) \mid p(A_1, \dots, A_n) \in \mathbf{R}\}$ the output schema derived from \mathbf{R} , (ii) $Map(\mathbf{R}) = \mathbf{R} \cup \hat{\mathbf{R}}$ the union of the input and output schemas, and (iii) $Adn(\mathbf{R}, \Sigma) = \mathbf{R} \cup \{p^\alpha(A_1, \dots, A_n) \mid p(A_1, \dots, A_n) \in \mathbf{R} \wedge p^\alpha \text{ appears in } Adn(\Sigma)\} \cup \hat{\mathbf{R}}$ the schema obtained by adding to \mathbf{R} the schemas of the relations introduced in the rewriting of constraints. Moreover, we shall also denote with $Map(\Sigma) = \Sigma \cup \{p(x_1, \dots, x_n) \rightarrow \hat{p}(x_1, \dots, x_n) \mid p(A_1, \dots, A_n) \in \mathbf{R}\}$ the set of constraints containing, in addition to Σ , a set of TGDs mapping tuples over the input schema to tuples over the output schema.

Example 6.3. Consider the constraints $\Sigma_{1.3}$ of Example 1.3. Initially, $Adn(\Sigma_{1.3})$ contains two constraints derived by adorning the body variables as bound ($Base(\Sigma_{1.3})$)

$$\begin{aligned} r_1 &: N^b(x) \rightarrow \exists y E^{bf}(x, y) \\ r_2 &: S^b(x) \wedge E^{bb}(x, y) \rightarrow N^b(y) \end{aligned}$$

In the second step two new constraints are generated ($Derived(\Sigma_{1.3})$). Due to the new predicate E^{bf} , the following constraint, derived from constraint r_2 , has been introduced:

$$r_3 : S^b(x) \wedge E^{bf}(x, y) \rightarrow N^f(y)$$

At this point the new predicate symbol N^f has been generated and, thus, a new constraint derived from r_1 is added:

$$r_4 : N^f(x) \rightarrow \exists y E^{ff}(x, y)$$

From the new predicate E^{ff} no new constraint is generated since the variable x in the body of the second constraint is bounded as it also appears in the predicate S^b .

Moreover, $Adn(\Sigma)$ also contains TGDs mapping input tuples into “bounded predicates” ($In(\Sigma_{1.3})$):

$$\begin{aligned} r_5 &: N(x) \rightarrow N^b(x) \\ r_6 &: S(x) \rightarrow S^b(x) \\ r_7 &: E(x, y) \rightarrow E^{bb}(x, y) \end{aligned}$$

and TGDs mapping tuples of adorned relations into “output” relations ($Out(\Sigma_{1.3})$):

$$\begin{aligned} r_8 &: N^b(x) \rightarrow \hat{N}(x) \\ r_9 &: N^f(x) \rightarrow \hat{N}(x) \\ r_{10} &: S^b(x) \rightarrow \hat{S}(x) \\ r_{11} &: E^{bb}(x, y) \rightarrow \hat{E}(x, y) \\ r_{12} &: E^{bf}(x, y) \rightarrow \hat{E}(x, y) \\ r_{13} &: E^{ff}(x, y) \rightarrow \hat{E}(x, y) \quad \square \end{aligned}$$

It is important to observe that the set of constraints $\Sigma_{1.3}$ is neither stratified nor super-weak acyclic, while $Adn(\Sigma_{1.3})$ is weakly acyclic. In fact, $dep(Adn(\Sigma_{1.3}))$, without considering edges in $In(\Sigma_{1.3})$ and $Out(\Sigma_{1.3})$, which do not affect chase termination, contains only the following edges: $N_1^b \rightarrow E_1^{bf}$, $N_1^b \xrightarrow{*} E_2^{bf}$, $E_2^{bb} \rightarrow N_1^b$, $E_2^{bf} \rightarrow N_1^f$, $N_1^f \rightarrow E_1^{ff}$, $N_1^f \xrightarrow{*} E_2^{ff}$.

Fact 1 *Let Σ be a set of standard TGDs. A position R_i belongs to $aff(\Sigma)$ iff there is some predicate $R_i^{\alpha_1 \dots \alpha_m}$ in $Adn(\Sigma)$ such that $\alpha_i = f$. \square*

From now on we shall denote with \mathbf{Cr} the whole set of criteria so far considered for checking termination of all chase sequences, i.e. $\mathbf{Cr} = \{WA, CStr, SC, SwA, SR, IR, WA-Str, SC-Str, SwA-Str, LS\}$.

Theorem 6.4. *For every set of TGDs Σ over a database schema \mathbf{R}*

$$\langle \text{Map}(\mathbf{R}), \text{Map}(\Sigma) \rangle \equiv_{\mathbf{R}/\hat{\mathbf{R}}} \langle \text{Adn}(\mathbf{R}, \Sigma), \text{Adn}(\Sigma) \rangle$$

□

Proof. We have to prove that for every database D over \mathbf{R} , $USol(D, \text{Map}(\Sigma))[\hat{\mathbf{R}}] = USol(D, \text{Adn}(\Sigma))[\hat{\mathbf{R}}]$, that is for every database $J \in USol(D, \text{Map}(\Sigma))$ there is a database $K \in USol(D, \text{Adn}(\Sigma))$ such that $J[\hat{\mathbf{R}}] = K[\hat{\mathbf{R}}]$ and vice versa. To simplify the notation we shall use Σ^α denoting $\text{Adn}(\Sigma)$ and $\bar{\Sigma}$ to denote $\text{Map}(\Sigma)$.

- (Base case - Step 0) Let K_0 be the database obtained by applying a single step of the core chase, i.e. $D \xrightarrow{\Sigma^\alpha \downarrow} K_0$, we have that $\text{Adn}^{-1}(K_0) = J_0 = D$ and $J_0[\hat{\mathbf{R}}] = K_0[\hat{\mathbf{R}}] = \emptyset$. Observe that K_0 could be obtained by just applying TGDs in $\text{In}(\Sigma)$ as at the first step adorned relations are empty.
- (Inductive case - Step i) Assume that J_{i-1} and K_{i-1} are s.t. $J_{i-1} \subseteq \text{Adn}^{-1}(K_{i-1})$ and $J_{i-1}[\hat{\mathbf{R}}] = K_{i-1}[\hat{\mathbf{R}}]$ (up to nulls renaming). Let $J_{i-1} \xrightarrow{\bar{\Sigma}} J'_i$, $K_{i-1} \xrightarrow{\Sigma^\alpha} K'_i$ and $J_i = \text{core}(J'_i)$, $K_i = \text{core}(K'_i)$. We have that $J_i \subseteq \text{Adn}^{-1}(K_i)$ and $J_i[\hat{\mathbf{R}}] = K_i[\hat{\mathbf{R}}]$ since the set of retracted tuples derived by means of Σ^α is contained (up to variable renaming) in the set of retracted tuples derived by means of $\bar{\Sigma}$. Indeed, two tuples $p(t_1)$ and $p(t_2)$ in J'_i may 'correspond' to tuples $p^{\alpha_1}(t_1)$ and $p^{\alpha_2}(t_2)$ in K'_i and therefore, $p(t_1)$ and $p(t_2)$ may be isomorphic, while $p^{\alpha_1}(t_1)$ and $p^{\alpha_2}(t_2)$ are not isomorphic because of the different adornments. On the other side if there are two isomorphic tuples in K'_i , the 'corresponding' tuples in J'_i are isomorphic as well. However, since the relations in $J'_i[\hat{\mathbf{R}}]$ and $K'_i[\hat{\mathbf{R}}]$ 'eliminate' adornments for any two isomorphic tuples in $J'_i[\hat{\mathbf{R}}]$ the 'corresponding' tuples in $K'_i[\hat{\mathbf{R}}]$ are isomorphic too, and vice versa. □

The previous theorem states that for every database D over a schema \mathbf{R} and for each universal solution J derived by applying the source TGDs Σ to D there is a universal solution K derived by applying the rewritten constraints $\text{Adn}(\Sigma)$ to D such that $J[\hat{\mathbf{R}}] = K[\hat{\mathbf{R}}]$ and vice versa. In particular, since $p^\alpha(t) \in K - D$ implies that there is a $p(t) \in J$ and $p(t) \in J$ implies that there is a $p^\alpha(t) \in K$, we have that each relation in J is partitioned into relations of $K - D$. The tuples in D appear twice in K since they are also copied into 'bounded' relations. Note that if the set of constraints $\text{Adn}(\Sigma)$ satisfies some chase termination criterion, the chase terminates considering both the source set Σ and the rewritten set $\text{Adn}(\Sigma)$. Clearly, the set $\text{Adn}(\Sigma)$ is only used to check chase termination conditions (at compile-time), whereas we use the source set Σ to compute (at run-time) universal solutions.

Example 6.3. (cont.) Consider again the constraints $\Sigma_{1.3}$ of Example 1.3 and the source database $D = \{S(a), N(a)\}$. The set of constraints $\text{Map}(\Sigma_{1.3})$ is equal to

$$\Sigma_{1.3} \cup \left\{ \begin{array}{l} S(x) \rightarrow \hat{S}(x) \\ N(x) \rightarrow \hat{N}(x) \\ E(x, y) \rightarrow \hat{E}(x, y) \end{array} \right\}$$

The application of the chase to $\langle D, \text{Map}(\Sigma_{1.3}) \rangle$ produces the database $J = \{S(a), N(a), E(a, n_1), N(n_1), E(n_1, n_2), \hat{S}(a), \hat{N}(a), \hat{E}(a, n_1), \hat{N}(n_1), \hat{E}(n_1, n_2)\}$. The application of the chase to $\langle D, \text{Adn}(\Sigma) \rangle$ produces the database $K = \{S(a), N(a), S^b(a), N^b(a), N^f(n_1), E^{bf}(a, n_1), E^{ff}(n_1, n_2), \hat{S}(a), \hat{N}(a), \hat{E}(a, n_1), \hat{N}(n_1), \hat{E}(n_1, n_2)\}$. Clearly, the two solutions are equivalent (with respect to $\{S, N, E\}/\{\hat{S}, \hat{N}, \hat{E}\}$). \square

Let \mathcal{C} denote the class of TGDs satisfying criterion C , $\text{Adn-}\mathcal{C}$ denotes the class of TGDs Σ such that $\text{Adn}(\Sigma)$ satisfies criterion C . From Theorem 6.4 we have that if a set of constraints $\Sigma \in \text{Adn-}\mathcal{C}$ and $\mathcal{C} \in \mathbf{Cr}$ all chase sequences terminate, whereas if $\Sigma \in \text{Adn-Str}$ there is at least one terminating chase sequence. Notice that the rewriting functions is applied to sets of TGDs, while several of the criteria so far analyzed also consider EGDs. It is also worth noting that although the size of the adorned program increases and in the worst case it is exponential in number of affected variables in a constraints ($|\text{Adn}(\Sigma)| = O(\sum_{r \in \Sigma} 2^{|\text{AffVar}(r)|}) = O(|\Sigma| \times 2^{\text{maxAffVar}})$, where $\text{AffVar}(r)$ denotes the set of distinct variables occurring only in affected positions in r and maxAffVar is the maximum number of distinct variables occurring only in affected positions in a constraint, i.e. $\text{maxAffVar} = \text{max}\{|\text{AffVar}(r)| \mid r \in \Sigma\}$), chase termination checking is a compile time operation.

The below theorem states that the rewriting technique allows to recognize (by using classical criteria) larger classes of constraints for which chase termination is guaranteed.

Theorem 6.5. $\mathcal{C} \subsetneq \text{Adn-}\mathcal{C}$ for $C \in \mathbf{Cr} \cup \{\text{Str}\}$. \square

Proof. Assume that Σ is in \mathcal{C} and is not in $\text{Adn-}\mathcal{C}$, for $C \in \mathbf{Cr} \cup \{\text{Str}\}$: this means that $\text{Base}(\Sigma) \cup \text{Derived}(\Sigma)$ is not in \mathcal{C} , since $\text{In}(\Sigma)$ and $\text{Out}(\Sigma)$ are acyclic and do not affect termination conditions. However, as $\text{Adn}^{-1}(\text{Base}(\Sigma) \cup \text{Derived}(\Sigma)) = \Sigma$, we have that Σ is not in \mathcal{C} as well (which contradicts the hypothesis). In addition, $\mathcal{C} \subsetneq \text{Adn-}\mathcal{C}$ since there are sets of TGDs Σ such that Σ is in $\text{Adn-}\mathcal{C}$, but not in \mathcal{C} (see, for instance, Example 6.3). \square

6.2 Cyclicity Detection during Rewriting Process

Theorem 6.5 in the previous section evidenced that the rewriting function Adn can still be useful to improve termination criteria and enlarge the class of constraints for which chase termination is guaranteed. Moreover, there are still simple sets of constraints which are not recognized by any technique so far considered, even if constraints are rewritten. The below example shows such a case.

Example 6.6. Consider the set of constraints $\Sigma_{6.6}$:

$$\begin{aligned} r_1 &: R(x_1^{p_1}) \rightarrow S(x_2^{p_2}, x_3^{p_3}) \\ r_2 &: S(x_1^{p_4}, x_2^{p_5}) \rightarrow \exists z T(x_2^{p_6}, z^{p_7}) \wedge Q(x_2^{p_8}) \\ r_3 &: T(x_1^{p_9}, x_2^{p_{10}}) \wedge T(x_1^{p_{11}}, x_3^{p_{12}}) \wedge T(x_3^{p_{13}}, x_1^{p_{14}}) \rightarrow R(x_2^{p_{15}}) \end{aligned}$$

The set $\Sigma_{6.6}$ is not *LS* since $r_2 \hookrightarrow r_3$. Indeed, $r_2 < r_3 < r_1 < r_2$ and $MOVE(\Sigma_{6.6}, Out(r_2, z)) = \{p_7, p_{15}, \mathbf{p}_2, p_3, p_6, p_8\} < In(r_2, x_1) = \{\mathbf{p}_4\}$. $\Sigma_{6.6}$ is not in *Adn-LS* as well. In fact we have that the set $Adn(\Sigma_{6.6})$ is as follows:

$$\begin{aligned} s_1 &: R^b(x) \rightarrow S^{bb}(x, x) \\ s_2 &: S^{bb}(x_1, x_2) \rightarrow \exists z T^{bf}(x_2, z) \wedge Q^b(x_2) \\ s_3 &: T^{bb}(x_1, x_2) \wedge T^{bb}(x_1, x_3) \wedge T^{bb}(x_3, x_1) \rightarrow R^b(x_2) \\ s_4 &: T^{bf}(x_1, x_2) \wedge T^{bb}(x_1, x_3) \wedge T^{bb}(x_3, x_1) \rightarrow R^f(x_2) \\ s_5 &: R^f(x) \rightarrow S^{ff}(x, x) \\ s_6 &: S^{ff}(x_1, x_2) \rightarrow \exists z T^{ff}(x_2, z) \wedge Q^f(x_2) \\ s_7 &: T^{bf}(x_1, x_2) \wedge T^{ff}(x_1, x_3) \wedge T^{ff}(x_3, x_1) \rightarrow R^f(x_2) \\ s_8 &: T^{ff}(x_1, x_2) \wedge T^{ff}(x_1, x_3) \wedge T^{ff}(x_3, x_1) \rightarrow R^f(x_2) \end{aligned}$$

$Adn(\Sigma_{6.6})$ is not in *LS* as constraints s_5, s_6, s_8 reproduce the same structure of the source constraints which, as discussed in Example 6.6, are not in *LS*.

However it is easy to check that the chase terminates for all database instances. \square

Thus, in this section we present a new rewriting technique which allow us to detect larger classes of constraints for which chase termination is guaranteed. Our rewriting algorithm is inspired by the one presented in [GS10] and similar to the algorithm Adn^+ there presented, but uses different adornments for free variables. Before presenting our rewriting algorithm let us recall some further notations. Let $Adn(\cdot)$ be the function rewriting a set of constraints Σ , defined over a database schema \mathbf{R} , into an adorned set Σ^α (as proposed in [GS10]) we shall denote with $Adn^{-1}(\cdot)$ the function taking in input an adorned first order formula consisting of a conjunction of atoms or a constraint or a set of constraints and gives in output the same formula without adornments. Since the new rewriting function here introduced adorns constraints using different free adornments of the form f_i , we shall denote with $src(r^\alpha)$ (or simply r) the constraint in the source set Σ from which an adorned one r^α has been derived.¹

Head adornment. Given a TGD $r: \phi(\mathbf{x}, \mathbf{z}) \rightarrow \exists \mathbf{y} \psi(\mathbf{x}, \mathbf{y})$ and let $\phi^\alpha(\mathbf{x}, \mathbf{z})$ be a coherent adornment for the body atoms, then $SkHeadAdn(r, \phi^\alpha(\mathbf{x}, \mathbf{z}))$ denotes the adorned head of r obtained as follows: i) every universally quantified variable has the same adornment of the body

¹ We shall use $src(r^\alpha)$, instead of $Adn^{-1}(r^\alpha)$, when we are interested in the constraint identifier rather than in the constraint definition.

occurrences, ii) existentially quantified variables appearing in constraints with empty body and constants are adorned as b ; iii) every existentially quantified variable y is adorned with an adornment f_i where the subscript is an integer value associated to the skolem function $f_y^r(\alpha[\mathbf{x}])$, (here $\alpha[\mathbf{x}]$ denotes the substring of α corresponding to \mathbf{x}). For instance, for $r : R(x, z) \rightarrow \exists y R(x, y)$ we have that $SkHeadAdn(r, R^{bb}(x, z)) = \exists y R^{bf_1}(x, y)$, where $f_1 = f_y^r(b)$, and $SkHeadAdn(r, R^{bf_1}(x, z)) = \exists y R^{bf_1}(x, y)$.

Adornment substitution. In order to have terminating sequences we will also use substitutions for adornments sequences. In particular, a substitution θ is a set of pairs f_i/f_j such that $i \neq j$; obviously, the same symbol cannot be used in both left and right sides of substitutions, i.e. a symbol f_j used to replace a symbol f_i cannot be substituted in θ by a symbol f_k .

The algorithm builds a graph E storing dependencies among adorned predicates. In particular, an edge (p_r^α, q_s^β) states that the predicate p^α , appearing in the head of an adorned constraint derived from r , has caused the creation of an adorned predicate q^β appearing in the head of a constraint derived from s . The graph is built for checking cyclic dependencies among adorned predicates denoting possible non-terminating chase sequences.

Adornment algorithm. The rewriting of constraints is performed by the function $Adn++$ reported in Figure 6.2 which differs from the Adn function defined in [GS10] in several aspects: (i) in the generation of adorned predicates it also considers how constraints may fire each other; (ii) the adornment of the head is done by applying the function $SkHeadAdn$ which introduces adornments with subscripts and these different free adornments appear in the output set; (iii) when a new adorned constraint r^α is generated, if there is an adorned constraint r^β and a substitution θ such that $r^\alpha\theta = r^\beta$, r^α is replaced by $Body(r^\alpha) \rightarrow Head(r^\beta)$, to avoid the creation of an infinite set of adorned constraints; (iv) in addition to the adorned set of TGDs, the algorithm also returns a boolean taking into account the fact that a form of cyclicity has been detected and a substitution has been used to unify two adorned constraints (to avoid an infinite set of adorned TGDs).

It is worth noting that if there are constraints with constants it is possible to generate adorned constraints with (adorned) body predicates not depending on the source predicates; these constraints are useless and, therefore, could be dropped.

Analogously to the function Adn , the function $Adn++$ receives in input a set of TGDs Σ , but differently from Adn , it returns a pair consisting of an adorned set of TGDs (with different free symbols) and a boolean value giving information about the detection of a form of cyclicity. The two elements returned by the algorithm (set of constraints and boolean value) are denoted by $Adn++(\Sigma)[1]$ and $Adn++(\Sigma)[2]$, respectively. $Adn++(\Sigma)[1]$ consists of four different subsets: $Base(\Sigma)$, $Derived(\Sigma)$, $In(\Sigma)$ and $Out(\Sigma)$, denoting, respectively, base, derived, input and output constraints.

```

Function  $Adn++(\Sigma)$ ;
Input Set of TGDs  $\Sigma$  over schema  $\mathbf{R}$ ;
Output Set of (adorned) TGDs  $Base \cup Derived \cup In \cup Out$ 
        Boolean value  $Cyc$ ;
begin
   $Base = Derived = In = Out = New\_Pred = E = \emptyset$ ;
   $Cyc = false$ ;
  // Let  $Body^b(r)$  be the conjunction obtained by adorning atoms
  // in  $Body(r)$  with strings of  $b$  symbols
   $Used\_Pred = \{p_r^b \mid \exists r \in \Sigma \text{ and } \exists p^b \text{ in } Body^b(r)\}$ ;
  for each applicable  $r \in \Sigma$  do begin
     $Base = Base \cup \{Body^b(r) \rightarrow SkHeadAdn(r, Body^b(r))\}$ ;
     $New\_Pred = New\_Pred \cup \{p_r^\alpha | p^\alpha(t) \in SkHeadAdn(r, Body^b(r))\}$ 
       $- Used\_Pred$ ;
  end for;
  while ( $New\_Pred \neq \emptyset$ ) do begin
    Select nondeterministically  $p_s^{\alpha_1 \dots \alpha_n} \in New\_Pred$ ;
     $New\_Pred = New\_Pred - \{p_s^{\alpha_1 \dots \alpha_n}\}$ ;
     $Used\_Pred = Used\_Pred \cup \{p_s^{\alpha_1 \dots \alpha_n}\}$ ;
    for each  $r \in (Base \cup Derived)$  s.t.  $s < src(r)$  do
      for each  $p^\beta(x_1, \dots, x_n) \in Body(r)$  do begin
         $B' = Body(r) - \{p^\beta(x_1, \dots, x_n)\} \cup \{p^{\gamma_1 \dots \gamma_n}(x_1, \dots, x_n)\}$ ;
        //  $\gamma_i = b$  if  $x_i \in Consts$ ;
        //  $\gamma_i = \alpha_i$  if  $x_i \in Vars$ ; ( $i \in [1..n]$ )
        if  $B'$  is coherent then begin
          Let  $H' = SkHeadAdn(Adn^{-1}(r), B')$ ;
          if ( $\exists r^\beta \in Derived$  and  $\exists subst. \theta$  s.t.  $(B' \rightarrow H')\theta = r^\beta$ )
            then begin
               $Derived = Derived \cup \{B' \rightarrow Head(r^\beta)\}$ ;
               $E = E \cup \{(p_s^{\alpha_1 \dots \alpha_n}, p_{src(r)}^\omega) \mid p^\omega(t) \in Head(r^\beta)\}$ ;
              if ( $r$  is exist. quantified and  $E$  is cyclic) then
                 $Cyc = true$ ;
            end;
          else begin
             $Derived = Derived \cup \{B' \rightarrow H'\}$ ;
             $E = E \cup \{(p_s^{\alpha_1 \dots \alpha_n}, p_{src(r)}^\omega) \mid p^\omega(t) \in H'\}$ ;
             $New\_Pred = New\_Pred \cup \{p_{src(r)}^\omega \mid p^\omega(t) \in H' \wedge p_{src(r)}^\omega \notin Used\_Pred\}$ ;
          end;
        end;
      else  $Derived = Derived \cup \{B' \rightarrow Head(Adn^{-1}(r))\}$ ;
    end for;
  end while;
  Delete from  $Derived$  constraints with unadorned heads;
  for each  $p(A_1, \dots, A_n) \in \mathbf{R}$  do
     $In = In \cup \{p(x_1, \dots, x_n) \rightarrow p^{b \dots b}(x_1, \dots, x_n)\}$ ;
  for each  $p(A_1, \dots, A_n) \in \mathbf{R}$  do
    for each  $p^\alpha(z_1, \dots, z_n)$  appearing in  $Base \cup Derived$  do
       $Out = Out \cup \{p^\alpha(x_1, \dots, x_n) \rightarrow \hat{p}(x_1, \dots, x_n)\}$ ;
  return  $\langle Base \cup Derived \cup In \cup Out, Cyc \rangle$ ;
end.

```

Fig. 6.2. Constraint Rewriting Function $Adn++$.

Example 6.7. Consider the set of constraints $\Sigma_{6,7}$

$$\begin{aligned} r_1 : N(x) &\rightarrow \exists y E(x, y) \\ r_2 : E(x, y) &\rightarrow N(y) \end{aligned}$$

The following set of base adorned constraints are first introduced:

$$\begin{aligned} s_1 : N^b(x) &\rightarrow \exists y E^{bf_1}(x, y) \\ s_2 : E^{bb}(x, y) &\rightarrow N^b(y) \end{aligned}$$

Next, the below adorned constraints are introduced in the set *Derived*:

$$\begin{aligned} s_3 : E^{bf_1}(x, y) &\rightarrow N^{f_1}(y) \\ s_4 : N^{f_1}(x) &\rightarrow \exists y E^{f_1f_2}(x, y) \\ s_5 : E^{f_1f_2}(x, y) &\rightarrow N^{f_2}(y) \\ s_6 : N^{f_2}(x) &\rightarrow \exists y E^{f_2f_3}(x, y) \\ s_7 : E^{f_2f_3}(x, y) &\rightarrow N^{f_3}(y) \end{aligned}$$

At this point the constraint

$$s' : N^{f_3}(x) \rightarrow \exists y E^{f_3f_4}(x, y)$$

should be added, but since there is a substitution $\theta = \{f_3/f_1, f_4/f_2\}$, the TGD

$$s_8 : N^{f_3}(x) \rightarrow \exists y E^{f_1f_2}(x, y)$$

is added and the generation of derived constraints terminates. Moreover, the graph E contains a cycle $E^{f_1f_2} \rightarrow N^{f_2} \rightarrow E^{f_2f_3} \rightarrow N^{f_3} \rightarrow E^{f_1f_2}$ and the adornment function returns the boolean value $Adn_{++}(\Sigma_{6,7})[2] = true$. \square

The next results state that the rewriting function Adn_{++} terminates and gives in output a set of TGDs equivalent to the input set.

Lemma 6.8. *For every set of TGDs Σ the function Adn_{++} always terminates.* \square

Proof. (Sketch). Termination of the rewriting algorithm is guaranteed by the use of substitutions which collapse adorned dependencies deriving from the same source constraint. \square

Theorem 6.9. *For every set of TGDs Σ over a database schema \mathbf{R} ,*

$$\langle Map(\mathbf{R}), Map(\Sigma) \rangle \equiv_{\mathbf{R}/\hat{\mathbf{R}}} \langle Adn_{++}(\mathbf{R}, \Sigma), Adn_{++}(\Sigma)[1] \rangle$$

where $Adn_{++}(\mathbf{R}, \Sigma) = \mathbf{R} \cup \{p^\alpha(A_1, \dots, A_n) \mid p(A_1, \dots, A_n) \in \mathbf{R} \wedge p^\alpha \text{ appears in } Adn_{++}(\Sigma)[1]\} \cup \hat{\mathbf{R}}$. \square

Proof. (Sketch). The proof follows the one of Theorem 6.4; it is sufficient to set Σ^α equal to $Adn++(\Sigma)[1]$ and $\bar{\Sigma}$ equal $Map(\Sigma)$. \square

Let \mathcal{C} denote a class of TGDs for which chase termination is guaranteed by checking a given criterion (e.g. $C \in \mathbf{Cr}$), we shall denote with $Adn++\mathcal{C}$ the class of TGDs Σ such that $Adn++(\Sigma)[1]$ is in \mathcal{C} . The following theorem shows that the rewriting technique here introduced is useful to enlarge the class of constraints which are recognized as terminating by a given criterion C .

Theorem 6.10. $\mathcal{C} \subsetneq Adn\text{-}\mathcal{C} \subsetneq Adn++\mathcal{C}$, for $C \in \mathbf{Cr} \cup \{\mathbf{Str}\}$. \square

Proof. $\mathcal{C} \subsetneq Adn\text{-}\mathcal{C}$ has been stated in Theorem 6.5. To show that $Adn\text{-}\mathcal{C} \subsetneq Adn++\mathcal{C}$, we first show $Adn\text{-}\mathcal{C} \subseteq Adn++\mathcal{C}$ by contradiction. We shall use the function Adn^{-s} which, similarly to the function Adn^{-1} , takes in input an adorned first order formula consisting of a conjunction of atoms or a constraint or a set of constraints and gives in output the same formula without subscripts, i.e. it eliminates subscripts from adornments. Suppose that there is a Σ and a criterion C such that $Adn(\Sigma) \in \mathcal{C}$ and $Adn++(\Sigma)[1] \notin \mathcal{C}$. Moreover, since $Adn^{-s}(Adn++(\Sigma)[1]) \subseteq Adn(\Sigma)$, every criterion for chase termination C verified by $Adn++(\Sigma)[1]$ is verified by $Adn(\Sigma)$ as well. The strict containment, for all criteria considered, is proved by the fact that the set of constraints $\Sigma_{6.6}$ is in $Adn++\mathcal{W}\mathcal{A}$, but not in $Adn\text{-}\mathcal{L}\mathcal{S}$. \square

It is important to recall that a rewriting technique using adornments with subscripts has been first defined in [GS10]. Moreover, differently from the previous rewriting technique, where new subscripts are introduced any time a new adorned constraint with existentially quantified variables is generated, the current technique controls the introduction of subscript (i.e. different free symbols) using skolem functions applied to adornments of head universally quantified variables. In addition, different free symbols appear in the output set of the rewriting function here presented, whereas previous techniques returned in output a set of constraints with just two adornment symbols (b and f). To better understand the advantages provided by the current rewriting technique we refer to the following example.

Example 6.11. Considering the below set of constraints:

$$\begin{aligned} c_1 &: R(x, z) \rightarrow \exists y T(x, y) \\ c_2 &: T(x, y) \rightarrow R(x, y) \end{aligned}$$

the current rewriting technique $Adn++$ generates the dependencies:

$$\begin{aligned} r_1 &: R^{bb}(x, z) \rightarrow \exists y T^{bf_1}(x, y) \\ r_2 &: T^{bb}(x, y) \rightarrow R^{bb}(x, y) \\ r_3 &: T^{bf_1}(x, y) \rightarrow R^{bf_1}(x, y) \\ r_4 &: R^{bf_1}(x, z) \rightarrow \exists y T^{bf_1}(x, y) \end{aligned}$$

which are acyclic and, therefore, terminating, whereas the technique $Adn+$ proposed in [GS10] generates the below constraints:

$$\begin{aligned}
s_1 : R^{bb}(x, z) &\rightarrow \exists y T^{bf_1}(x, y) \\
s_2 : T^{bb}(x, y) &\rightarrow R^{bb}(x, y) \\
s_3 : T^{bf_1}(x, y) &\rightarrow R^{bf_1}(x, y) \\
s_4 : R^{bf_1}(x, z) &\rightarrow \exists y T^{f_1f_2}(x, y) \\
s_5 : T^{f_1f_2}(x, y) &\rightarrow R^{f_1f_2}(x, y) \\
s_6 : R^{f_1f_2}(x, z) &\rightarrow \exists y T^{f_2f_3}(x, y) \\
s_7 : T^{f_2f_3}(x, y) &\rightarrow R^{f_2f_3}(x, y) \\
s_8 : R^{f_2f_3}(x, z) &\rightarrow \exists y T^{f_3f_4}(x, y) \\
s_9 : T^{f_3f_4}(x, y) &\rightarrow R^{f_3f_4}(x, y) \\
s_{10} : R^{f_3f_4}(x, z) &\rightarrow \exists y T^{f_4f_5}(x, y)
\end{aligned}$$

At this point the rewriting procedure stops because was not able to detect the termination and returns a set of adorned constraints without subscripts. The output set does not give any advantage in terms of analysis of its structural properties. Therefore, the rewriting performed by the function $Adn++$ is more effective of the one presented in [GS10]. \square

We conclude by introducing our final class of constraints for which chase termination is guaranteed.

Definition 6.12 (Acyclicity). *A set of TGDs Σ is said to be Acyclic (AC) if $Adn++(\Sigma)[2]$ is false.* \square

The class of acyclic constraint is denoted by \mathcal{AC} . The next theorem shows that the hierarchy criteria here introduced collapse and coincide with the class \mathcal{AC} when constraints are rewritten using the adornment function $Adn++$.

Theorem 6.13. $\mathcal{AC} = Adn++\mathcal{WA} = Adn++\mathcal{LS}$. \square

Proof. (sketch) To show $Adn++\mathcal{WA} = Adn++\mathcal{LS}$ it is sufficient to demonstrate that $Adn++\mathcal{WA} \supseteq Adn++\mathcal{LS}$, as $Adn++\mathcal{WA} \subseteq Adn++\mathcal{LS}$ is trivial, that is if there is a Σ such that $Adn++(\Sigma)[1] \notin \mathcal{WA}$, then $Adn++(\Sigma)[1] \notin \mathcal{LS}$ also holds. Indeed, for any $\Sigma^\alpha = Adn++(\Sigma)[1]$, $\Sigma^\alpha \notin \mathcal{WA}$ means that there is a cycle in $dep(\Sigma^\alpha)$ with a special edge. It can be easily shown that in such a case there is a cycle in the firing graph $\Gamma(\Sigma^\alpha)$ and a cycle in the trigger graph $\Delta(\Sigma)$ as well.

We show now that $Adn++\mathcal{WA} = \mathcal{AC}$. We first consider the case of $\mathcal{AC} \subseteq Adn++\mathcal{WA}$ showing that for any Σ in \mathcal{AC} the set of constraints $Adn++(\Sigma)[1]$ is weakly acyclic. In fact, as said above, a cycle in $dep(\Sigma^\alpha)$ means that there is a cycle C in $\Gamma(\Sigma^\alpha)$ where nulls are created and propagated. This means that in the rewriting there is a constraint belonging to C which is used more than once and, therefore, a substitution is employed to avoid an infinite number of adorned constraints. Moreover, each predicate symbol in the (head of a TGD contained in the) path C depends on the other

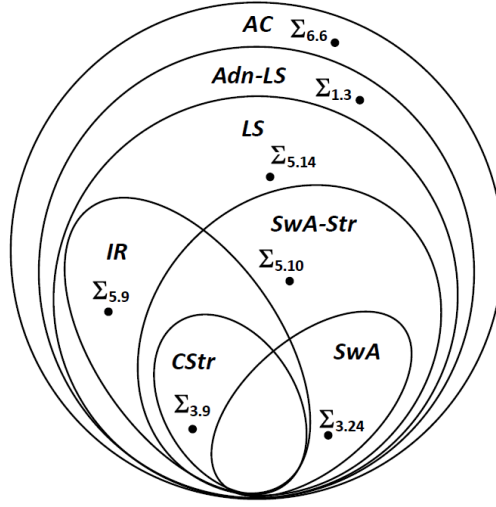


Fig. 6.3. Criteria Relationships.

predicate symbols. Consequently, in such a case $Adn++(\Sigma)[2] = true$. Viceversa, $Adn++WA \subseteq AC$ holds as for any $\Sigma \notin AC$ we have that $\Sigma \notin Adn++WA$. $Adn++(\Sigma)[2] = true$ means that a cycle where nulls may be propagated has been detected; this implies that $\Gamma(\Sigma^\alpha)$ is cyclic and that $dep(\Sigma^\alpha)$ contains a cycle with a special arc. \square

The below corollaries derive straightforwardly from the previous theorem and state that AC is the most general chase termination criterion and, for acyclic constraints, the length of all chase sequences is polynomial in the size of the input database.

Corollary 6.14. *Let Σ be a set of acyclic TGDs. Then for every database D all chase sequences of Σ over D terminate in polynomial time in the size of D .* \square

Corollary 6.15. $LS \subsetneq AC$. \square

The complete relationships among the previously discussed criteria is reported in Figure 6.3.

The *ChaseTEQ* System Prototype

This chapter presents *ChaseTEQ*, a system prototype for repairing and querying incomplete databases. The system consists of three different modules allowing users to i) design data dependencies and apply different criteria for checking chase termination, ii) execute the chase fixpoint to produce a (finite) repaired database with labelled nulls, and iii) execute (restricted) SQL queries over the repaired database.

7.1 System Description

ChaseTEQ is a system prototype for querying and repairing (using chase fixpoint algorithms) inconsistent databases. The main features devoted to support *constraint design*, *database repairing* and *database querying* are implemented by means of three different modules which will be next discussed. Thus, the user first defines a set of data dependencies Σ associated with a given database schema \mathbf{R} . After have checked that Σ guarantees chase termination, independently of the input database instance, the input database D (defined over \mathbf{R}) can be uploaded and the chase algorithm is applied to fix possible inconsistencies. The resulting database \hat{D} is then used to answer queries.

Constraints Design

The environment supporting constraints design is implemented by a module called *ChaseT*. The user, after having defined data dependencies over a given database, could check whether a specific criterium guarantees that the chase algorithm terminates. Alternatively the user could ask whether there exists a criterium (among those implemented in the system) which guarantees termination. To this aim, *ChaseT* provides a library of methods able to check the termination of the chase procedure for a given set of integrity constraints.

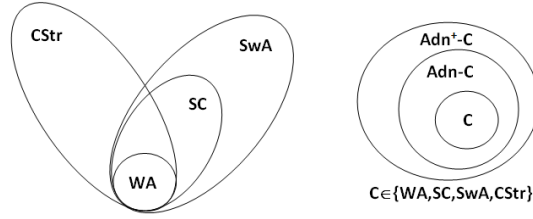


Fig. 7.1. Termination Criterion Fields

In particular, *ChaseT* implements weak acyclicity, safety, super-weak acyclicity and c-stratification criteria and two rewriting techniques: *Adn* and *Adn*⁺ (presented in [GS10]). The capacity of implemented criteria to recognize chase termination is shown in Figure 7.1.

The rewriting techniques, implemented by means of the two functions *Adn* and *Adn*⁺, produce, for a given set of constraints Σ , equivalent sets of constraints *Adn*(Σ) and *Adn*⁺(Σ), allowing to extend the application fields of known termination criteria. From a computational point of view, the test of WA, SC and SwA criteria is polynomial (in the size of dependencies), whereas the test of CStr criterion is in *coNP*. The rewriting functions could need exponential time in the size of the input constraints.

For a better understanding of possibly non-terminating conditions, *ChaseT* allows users to take vision of the graphical simulation of nulls propagation by showing the *constraints graph*, representing how constraints may activate each other, and the *position graph*, showing how values and nulls may be propagated through positions. In more detail, the constraints graph denotes i) the c-chase graph if the underlying criterium is CStr, and ii) trigger graph if the underlying criterium is SwA. Concerning the position graph, it denotes the dependency graph or the propagation graph according to the selected WA or SC criterium.

Database Repairing

Databases are repaired by applying the chase fixpoint algorithm which is implemented by the *ChaseE* module.

To model the presence of multiple null values, introduced by the chase procedure, *ChaseE* duplicates *affected* attributes (i.e attributes containing labelled nulls). More specifically, the input databases is first translated into a textual database D' . Next the chase algorithm is applied to D' and produces a new database D'' which could contain labelled nulls (represented by special strings). Finally, the database D'' is rewritten into a database \hat{D} so that each column has the original format and affected attributes are duplicated, that is for each affected attribute A there is a “dummy” attribute $A.d$ which is used to store the value of attribute A in database D'' , i.e. it is used to store

labelled nulls or the textual version of values appearing in affected attributes. Therefore, if the textual database D'' has a tuple t'' containing a labelled null n_j in attribute A , the corresponding tuple \hat{t} in the final database \hat{D} has a (standard) null value in attribute A and a labelled null value n_j in attribute A_d .

Example 7.1. Consider a database containing information concerning courses, teachers and students stored in four relations whose schemas are $Course(Name)$, $Teaches(Teacher, Course)$, $Follows(Student, Course)$ and $ST(Student, Teacher)$. The below set of constraints $\Sigma_{7.1}$

$$\begin{aligned} Course(x) &\rightarrow \exists y Teaches(y, x) \\ Follows(x, y) \wedge Teaches(z, y) &\rightarrow ST(x, z) \end{aligned}$$

states that each course must have a teacher and the teachers of a student are those teaching a course followed by the student.

Assuming to have the database instance $D_{7.1}$ consisting of the facts $Course(Db)$, $Follows(Tom, Db)$ and $Follows(Ann, Db)$. The application of the chase algorithm gives in output a (textual) database $D_{7.1}''$ containing the tuples $Course(Db)$, $Follows(Tom, Db)$, $Follows(Ann, Db)$, $Teaches(n_1, Db)$, $ST(Tom, n_1)$ and $ST(Ann, n_1)$. The corresponding database $\hat{D}_{7.1}$ consists of the tuples $Course(Db)$, $Follows(Tom, Db)$, $Follows(Ann, Db)$, $Teaches(\perp, Db, n_1)$, $ST(Tom, \perp, n_1)$ and $ST(Ann, \perp, n_1)$, where \perp denotes the standard null value. \square

Query Answering

Users could submit restricted SQL queries corresponding to relational expressions defining unions of conjunctive queries. The evaluation of queries is performed by the module *ChaseQ* which allows users to see a database having the same schema of the source one, that is dummy attributes are hidden and they are just used to check whether two nulls denote the same element. In particular, each query having equality condition $A = B$ involving affected attributes A and B , is rewritten by replacing this condition with $A_d = B_d$, where A_d and B_d are the dummy attributes corresponding to A and B , respectively. For instance, the following query

```
SELECT ST1.Student, ST2.Student
FROM   ST ST1, ST ST2
WHERE  ST1.Teacher = ST2.Teacher
AND    ST1.Student < ST2.Student
```

asking for pairs of students having the same teacher, defined over the database $D_{7.1}$ of Example 7.1 is rewritten as

```
SELECT ST1.Student, ST2.Student
FROM   ST ST1, ST ST2
WHERE  ST1.Teacher_d = ST2.Teacher_d
AND    ST1.Student < ST2.Student
```

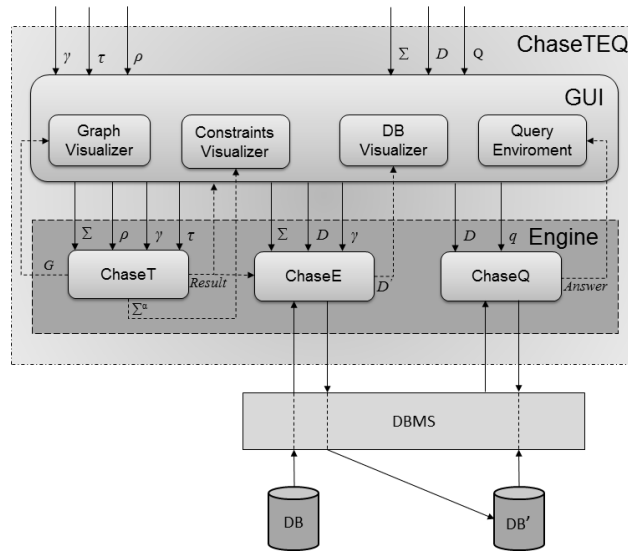


Fig. 7.2. *ChaseTEQ* architecture

and applied to the rewritten database $\hat{D}_{7.1}$.

7.2 Implementation

The top-level architecture of *ChaseTEQ* is depicted in Figure 7.2.

The *Graphical User Interface* (GUI) allows the user to provide the set Σ of data dependencies and three parameters: i) γ , denoting the type of chase she/he is interested in (standard, skolem oblivious, naive oblivious), ii) τ , denoting the selected termination criterium (WA, SC, SwA, CStr), and iii) ρ , denoting the possible rewriting technique she/he would use (Adn, Adn+ or none). Through the GUI it is also possible to upload a database D (specifying the database location and the credentials) and fix its possible inconsistencies by running the desired chase algorithm.

If the user wants to check the termination of the selected chase algorithm by applying a rewriting technique, the input set of dependencies Σ is rewritten (by the auxiliary *Rewriter* module) into a set of adorned dependencies Σ^α . Since the rewriting output also depends on the particular chase procedure (standard or oblivious) the user would use, the *Rewriter* module receives in input Σ and the parameters ρ and γ and gives in output the rewritten set of constraints Σ^α .

The system allows users to select a specific termination condition (WA, SC, etc.), but it is also possible to check termination without indicating any specific criterium (default option). In such a case, all techniques are applied and the system returns the properties of the input dependencies (see the

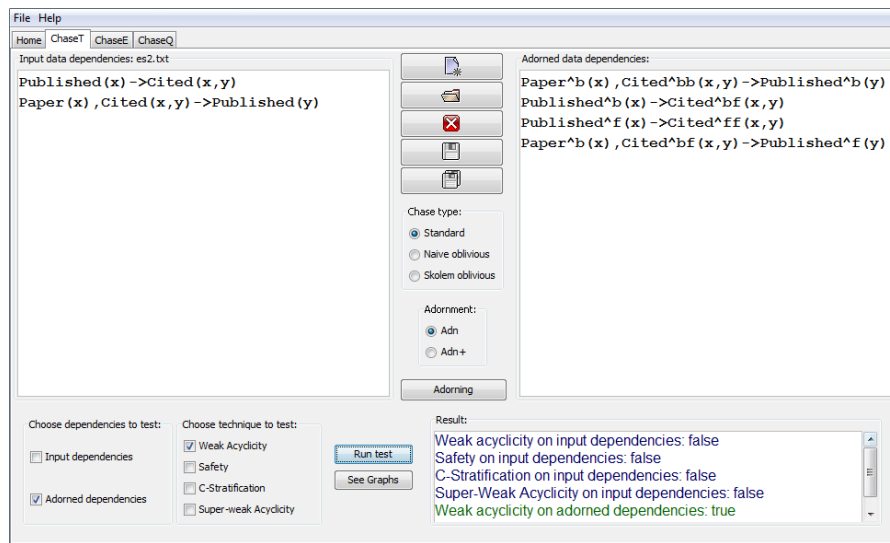


Fig. 7.3. *ChaseT* User Interface

bottom right window in Figure 7.3). The indication of a specific criterium is useful to identify the origin of possible non-terminating conditions.

Figure 7.3 shows how the user interacts with the system through the GUI. The left window shows the input set of dependencies, while the rewritten set of dependencies is showed in the right window; parameters are introduced through check boxes.

It is worth noting that the data dependencies defined by the user are first parsed to check syntactic errors and inconsistencies (e.g. the use of predicates having the same name and different arity).

For the analysis of the structural properties of a set of dependencies Σ , *ChaseT* builds two specific graphs for the selected criterium: the constraints graph and the position graph. The graphs can be visualized using a graph visualization tool for a better understanding of data dependency properties (see Figure 7.4).

The module *ChaseE* applies the desired chase algorithm (specified by the parameter γ) to the input database D and set of dependencies Σ ; it returns the rewritten repaired database \hat{D} where multiple null values are stored into additional dummy attributes. As said above, the presence of dummy attributes is hidden to the user, and labelled nulls appear as standard null values.

To execute queries *ChaseTEQ* uses the standard SQL engine. The *ChaseQ* module takes in input the query Q , performs its rewriting in \hat{Q} (as discussed in the previous section) and submits \hat{Q} on the rewritten database \hat{D} through the DBMS.

The system has been developed in Java using *Eclipse IDE* and is downloadable from www.info.deis.unical.it/chaseteq. The interactions among the

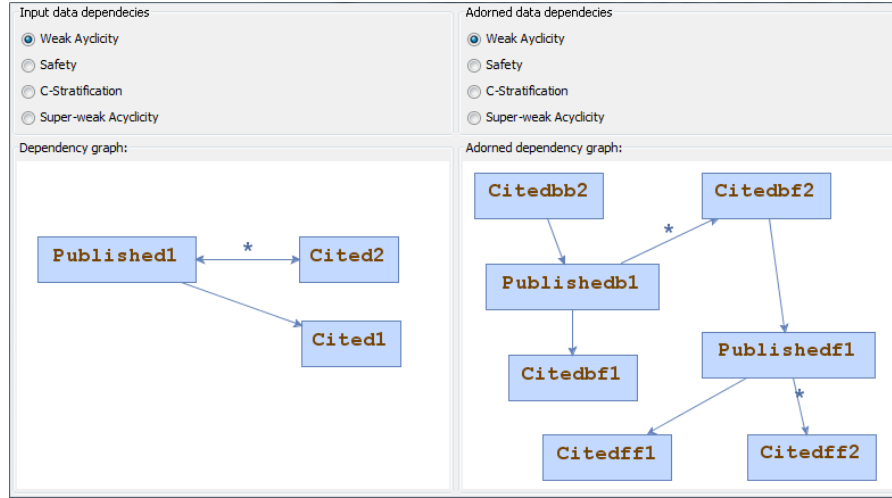


Fig. 7.4. Graph visualizer

different modules are carried out through interfaces so that each module can be easily modified without any inference on the other modules. The GUI has been written using the *Swing Java* libraries and the open source library *JGraphX* for the visualization of graphs. The current version of the system uses *Connector/J* driver and supports MySQL DBMS, however it is able to manage different DBMS drivers that can be easily added.

7.3 Application Scenario

In the following, a typical use-case scenario of *ChaseTEQ* is shown.

Suppose, for instance, that the user wants to check the termination of the standard chase procedure for the following set of constraints $\Sigma_{7.3}$:

$$\begin{aligned}\sigma_1 &: \text{Published}(x) \rightarrow \exists y \text{Cites}(x, y) \\ \sigma_2 &: \text{Paper}(x) \wedge \text{Cites}(x, y) \rightarrow \text{Published}(y)\end{aligned}$$

defined over the schema $\text{Paper}(\text{PaperId})$, $\text{Published}(\text{PaperId})$ and $\text{Cites}(\text{Paper}, \text{CitedPaper})$ representing published papers and their citations.

Figure 7.3 shows the use of the *constraint design* features, provided by the *ChaseT* tool. In order to test the termination of the standard chase procedure for the given set of constraints $\Sigma_{7.3}$, the user introduces it in the “*Input data dependency*” window, select “*standard*” as chase type and apply the known termination conditions (*Run test* button), by taking into account that more general techniques require greater computational effort and the explanation is more complex.

For the example shown in Figure 7.3 the application of all termination conditions to the original set of dependencies produces a negative result. However,

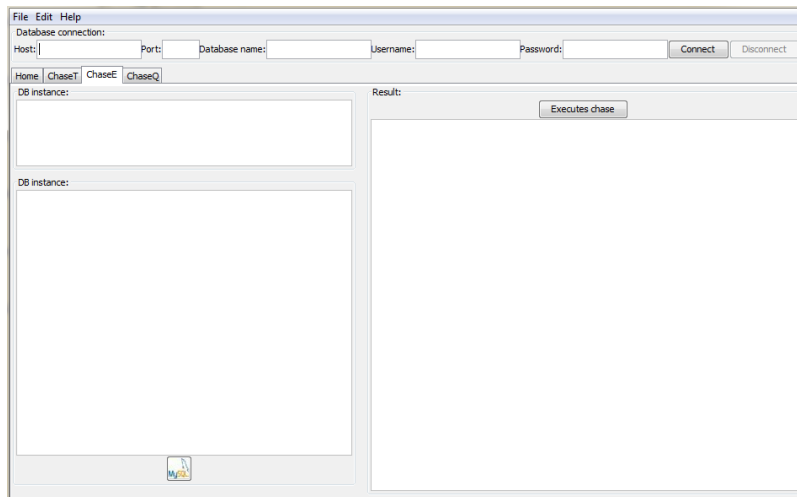


Fig. 7.5. *ChaseE* User Interface

the application of the simplest rewriting generates the equivalent set $Adn(\Sigma)$, which is weakly acyclic. The rewritten set (generated using the *Adorning* button), is visualized in the “*Adorned data dependency*” window (in the right side of Figure 7.3).

The position graphs for the source and rewritten constraints can be visualized (using the *See Graphs* button) to understand the behaviors of the different techniques. As shown in Figure 7.4, the dependency graph of Σ contains a cycle going through a special edge (“*Dependency graph*” window), whereas the dependency graph of $Adn(\Sigma)$ is acyclic (see “*Adorned dependency graph*” window).

In order to execute the chase algorithm the user select the “execution frame” (see Figure 7.5), loads a database instance stored in a relational database and executes the previous selected chase algorithm by using the *Execute Chase* button. The result can be viewed in the Result window and stored in the source relational database.

Queries are submitted and evaluated on the repaired data-base using the “query frame” shown in Figure 7.6. SQL queries are written or uploaded through the “*Query window*”, whereas the input database can be visualized in the “*DB instance window*”. The result of the execution of the input query on the (repaired) database is showed in the “*Result window*”. The rewritings of database and query are hidden to users which see a database having the same schema of the source database. Concerning the execution of the chase fixpoint algorithm, the user interacts with a “chase execution frame” which, from a graphical point of view, is similar to the query frame. This frame allows users to upload the set of constraints Σ and the input databases D (visualized

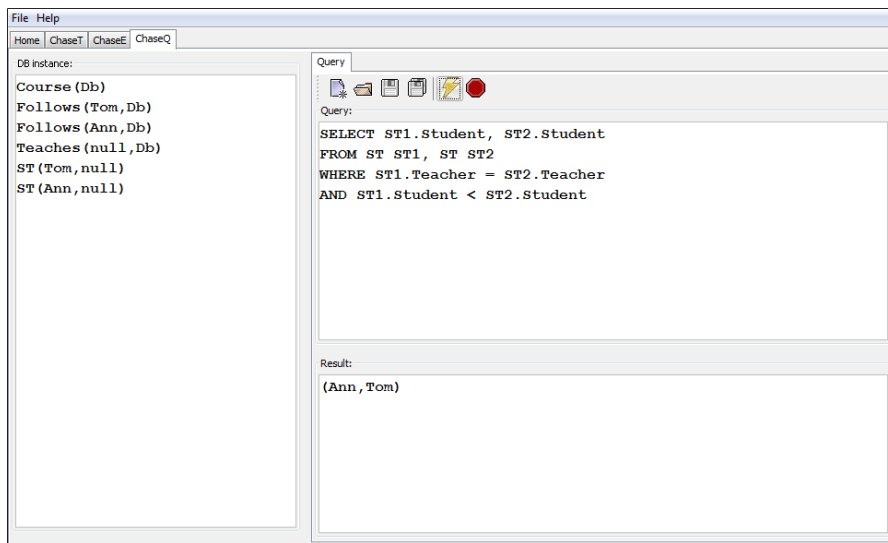


Fig. 7.6. *ChaseQ* User Interface

in two different windows) and to execute the chase algorithm on Σ and D ; the resulting repaired database is stored and visualized in the “*Result window*”.

Conclusions

This thesis has presented new criteria and rewriting techniques that allow to recognize larger classes of constraints for which the chase fixpoint algorithm always terminates, independently from the database instance.

We have first introduced extensions of the well-known stratification criterion and have defined further criteria which generalize both super-weak acyclicity and stratification-based criteria (including the class of constraints which are inductively restricted).

Next we have introduced a rewriting algorithm transforming the original set of constraints into an equivalent set so that structural properties for chase termination can be checked on the rewritten set. We have shown that significant classes of constraints for which the chase execution terminates can be captured by classical criteria after that constraints have been adorned. The rewriting technique is orthogonal to termination conditions and improves current chase termination criteria.

A more general rewriting algorithm producing as output an equivalent set of dependencies and a boolean value stating whether a sort of cyclicity has been detected, has also been proposed. The new rewriting technique and the checking of acyclicity allowed us to introduce the class of acyclic constraints which guarantees that all chase sequences are finite with a length polynomial in the size of the input database. Acyclicity represent the most general criterion so far defined.

Regarding the \mathcal{T} -hierarchy, it is important to observe that for any fixed finite m , $\mathcal{AC} \not\subseteq \mathcal{T}[m]$ and even $Adn\text{-}\mathcal{C} \not\subseteq \mathcal{T}[m]$, for all criteria C , since there are examples of constraints which are in \mathcal{AC} and also in $Adn\text{-}\mathcal{C}$, which are not in $\mathcal{T}[m]$. Moreover it is also important to observe that although the problem of checking whether $\prec_P (r_1, \dots, r_m)$ is in \mathcal{NP} , the complexity could be bounded by $O(k^{m \times n})$, where n (resp. k) is the maximum number of atoms in the body (resp. head) of a constraint (see Lemma 5.6). Even worse, the number of checks which need to be done is equal to $|\Sigma|^m$. Therefore, considering sequences of firing constraints greater than two make these classes not very relevant from a complexity point of view.

Finally, we presented *ChaseTEQ*, a tool for testing chase termination, and, repairing and querying incomplete databases. We described the architecture of the system and the use-case scenarios. As future work, we plan to extend the set of chase termination criteria with new criteria proposed in this thesis.

Dealing with EGDs

In this thesis we have not considered equality generating dependencies. Current work is addressing this important aspect. A possible solution to this problem could be based on simple transformations replacing EGDs with TGDs, such as the ones made in [Mar09]. Moreover, as discussed in [PLC⁺08, CGL09, GS10], there are simple cases of EGDs that could be easily treated such as those defining, in the framework of ontology languages, functional restriction on roles [PLC⁺08] and keys which are not conflicting with TGDs [CGL09].

In general, when we consider both TGDs and EGDs, different situations may happen. We have already seen in Example 1.4, that in some cases the presence of EGDs allows to have terminating chase sequences in the case where the set of only TGDs dependencies is non-terminating, but we also have the opposite case (see next example) where the presence of EGDs allows to have non-terminating chase sequences in the case where the set of only TGDs dependencies is terminating.

Consider for instance the following set of constraints Σ :

$$\begin{aligned} r_1 &: N(x) \rightarrow \exists y \exists z E(x, y, z) \\ r_2 &: E(x, y, y) \rightarrow N(y) \\ r_3 &: E(x, y, z) \rightarrow y = z \end{aligned}$$

The set of TGDs $\Sigma' = \{r_1, r_2\}$ is terminating for all database instances as recognized by several criteria (e.g. super-weak acyclicity). Moreover, the chase fixpoint applied to Σ and the database $D = \{N(a)\}$ is non-terminating as it introduces an infinite number of tuples $E(a, \eta_1, \eta_1), N(\eta_1), E(\eta_1, \eta_2, \eta_2), N(\eta_2), \dots$

Thus, we are working on extending our framework by also taking into account EGDs. In particular, we are developing more general rewriting techniques for TGDs and EGDs guaranteeing the termination of all chase sequences, or of at least one, for all database instances D in polynomial time in the size of D .

References

- [ABU79] Alfred V. Aho, Catriel Beeri, and Jeffrey D. Ullman. The theory of joins in relational databases. *ACM Trans. Database Syst.*, 4(3):297–314, 1979.
- [AHV95] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
- [Ber06] Leopoldo E. Bertossi. Consistent query answering in databases. *SIGMOD Record*, 35(2):68–76, 2006.
- [BKL11] Leopoldo E. Bertossi, Solmaz Kolahi, and Laks V. S. Lakshmanan. Data cleaning and query answering with matching dependencies and matching functions. In *ICDT*, pages 268–279, 2011.
- [BR91] Catriel Beeri and Raghu Ramakrishnan. On the power of magic. *J. Log. Program.*, 10(1/2/3&4):255–299, 1991.
- [BV84] Catriel Beeri and Moshe Y. Vardi. A proof procedure for data dependencies. *J. ACM*, 31(4):718–741, 1984.
- [CCGL04] Andrea Cali, Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Data integration under integrity constraints. *Inf. Syst.*, 29(2):147–163, 2004.
- [CGK08] Andrea Cali, Georg Gottlob, and Michael Kifer. Taming the infinite chase: Query answering under expressive relational constraints. In *Description Logics*, 2008.
- [CGL09] Andrea Cali, Georg Gottlob, and Thomas Lukasiewicz. Datalog[±]: a unified approach to ontologies and integrity constraints. In *ICDT*, pages 14–30, 2009.
- [CGP10] Andrea Cali, Georg Gottlob, and Andreas Pieris. Advanced processing for ontological queries. *PVLDB*, 3(1):554–565, 2010.
- [Cho07] Jan Chomicki. Consistent query answering: Five easy pieces. In *ICDT*, pages 1–17, 2007.
- [CLR03] Andrea Cali, Domenico Lembo, and Riccardo Rosati. On the decidability and complexity of query answering over inconsistent and incomplete databases. In *PODS*, pages 260–271, 2003.
- [CLRS01] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 2001.
- [Cod71] E. F. Codd. Normalized data structure: A brief tutorial. In E. F. Codd and A. L. Dean, editors, *Proceedings of 1971 ACM-SIGFIDET Workshop on Data Description, Access and Control, San Diego, California, November 11-12, 1971*, pages 1–17. ACM, 1971.

- [Cod72] E. F. Codd. Relational completeness of data base sublanguages. *In: R. Rustin (ed.): Database Systems: 65-98, Prentice Hall and IBM Research Report RJ 987, San Jose, California, 1972.*
- [Cod74] E. F. Codd. Recent investigations in relational data base systems. *In IFIP Congress*, pages 1017–1021, 1974.
- [CP11] Andrea Cali and Andreas Pieris. On equality-generating dependencies in ontology querying - preliminary report. *In AMW*, 2011.
- [DGST11] Andrea De Francesco, Sergio Greco, Francesca Spezzano, and Irina Trubitsyna. Chaset: A tool for checking chase termination. *In SUM*, pages 520–524, 2011.
- [DL97] Oliver M. Duschka and Alon Y. Levy. Recursive plans for information gathering. *In IJCAI (1)*, pages 778–784, 1997.
- [DLLR07] Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. On reconciling data exchange, data integration, and peer data management. *In PODS*, pages 133–142, 2007.
- [DNR08] Alin Deutsch, Alan Nash, and Jeffrey B. Remmel. The chase revisited. *In PODS*, pages 149–158, 2008.
- [DPT99] Alin Deutsch, Lucian Popa, and Val Tannen. Physical data independence, constraints, and optimization with universal plans. *In VLDB*, pages 459–470, 1999.
- [DPT06] Alin Deutsch, Lucian Popa, and Val Tannen. Query reformulation with constraints. *SIGMOD Record*, 35(1):65–73, 2006.
- [DST11] Andrea De Francesco, Francesca Spezzano, and Irina Trubitsyna. Chaset: A tool for checking chase termination. *In SEBD*, pages 163–174, 2011.
- [DT03] Alin Deutsch and Val Tannen. Reformulation of xml queries and constraints. *In ICDT*, pages 225–241, 2003.
- [Fag77] Ronald Fagin. Multivalued dependencies and a new normal form for relational databases. *ACM Trans. Database Syst.*, 2(3):262–278, 1977.
- [Fag81] Ronald Fagin. A normal form for relational databases that is based on domains and keys. *ACM Trans. Database Syst.*, 6(3):387–415, 1981.
- [FKMP05] Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa. Data exchange: semantics and query answering. *Th. Comp. Sc.*, 336(1):89–124, 2005.
- [FKP05] Ronald Fagin, Phokion G. Kolaitis, and Lucian Popa. Data exchange: getting to the core. *ACM Trans. Database Syst.*, 30(1):174–210, 2005.
- [GN08] Georg Gottlob and Alan Nash. Efficient core computation in data exchange. *J. ACM*, 55(2), 2008.
- [GS10] Sergio Greco and Francesca Spezzano. Chase termination: A constraints rewriting approach. *PVLDB*, 3(1):93–104, 2010.
- [GST11] Sergio Greco, Francesca Spezzano, and Irina Trubitsyna. Stratification criteria and rewriting techniques for checking chase termination. *PVLDB*, 4(11):1158–1168, 2011.
- [HY90] Richard Hull and Masatoshi Yoshikawa. Ilog: Declarative creation and manipulation of object identifiers. *In VLDB*, pages 455–468, 1990.
- [JK84] 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.
- [Len02] Maurizio Lenzerini. Data integration: A theoretical perspective. *In PODS*, pages 233–246, 2002.

- [Mar09] Bruno Marnette. Generalized schema-mappings: from termination to tractability. In *PODS*, pages 13–22, 2009.
- [Mei10] Michael Meier. *On the Termination of the Chase Algorithm*. Albert-Ludwigs-Universität Freiburg (Germany), 2010.
- [MMS79] David Maier, Alberto O. Mendelzon, and Yehoshua Sagiv. Testing implications of data dependencies. *ACM Trans. Database Syst.*, 4(4):455–469, 1979.
- [MPR09] Giansalvatore Mecca, Paolo Papotti, and Salvatore Raunich. Core schema mappings. In *SIGMOD Conference*, pages 655–668, 2009.
- [MSL09a] Michael Meier, Michael Schmidt, and Georg Lausen. On chase termination beyond stratification. *PVLDB*, 2(1):970–981, 2009.
- [MSL09b] Michael Meier, Michael Schmidt, and Georg Lausen. On chase termination beyond stratification. *CoRR*, abs/0906.4228, 2009.
- [PDST00] Lucian Popa, Alin Deutsch, Arnaud Sahuguet, and Val Tannen. A chase too far? In *SIGMOD Conference*, pages 273–284, 2000.
- [PLC⁺08] Antonella Poggi, Domenico Lembo, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati. Linking data to ontologies. *J. Data Semantics*, 10:133–173, 2008.
- [PS11] Reinhard Pichler and Sebastian Skritek. The complexity of evaluating tuple generating dependencies. In *ICDT*, pages 244–255, 2011.
- [tCCKT09] Balder ten Cate, Laura Chiticariu, Phokion G. Kolaitis, and Wang Chiew Tan. Laconic schema mappings: Computing the core with sql queries. *PVLDB*, 2(1):1006–1017, 2009.
- [Var84] Moshe Y. Vardi. The implication and finite implication problems for typed template dependencies. *J. Comput. Syst. Sci.*, 28(1):3–28, 1984.
- [Zan82] Carlo Zaniolo. A new normal form for the design of relational database schemata. *ACM Trans. Database Syst.*, 7(3):489–499, 1982.