Closed World Chasing

Gösta Grahne
Concordia University
Montreal, Canada, H3G 1M8
grahne@cs.concordia.ca

Adrian Onet *
Concordia University
Montreal, Canada, H3G 1M8
a_onet@cs.concordia.ca

ABSTRACT

We give a new closed world semantics for data exchange called constructible solutions semantics, and argue that this semantics is well suited for answering non-monotonic queries in data exchange. We show that the space of constructible solutions can be represented by conditional tables obtainable trough a novel conditional chase procedure which generalizes the classical chase.

1. INTRODUCTION

The chase is an algebraic proof procedure originally developed for testing logical implication between (sets of) embedded dependencies. Recently the chase has experienced a revival due to its applications in data exchange. Data exchange consists of transforming a source database into a target database, according to a set of embedded dependencies that describe the mapping between the source and the target database. A problem arises from the fact that there usually are infinitely many valid target databases, also called solutions, that correctly reflect the mapping. Nevertheless, the target database has to be materialized in some way, and chasing the source instance with the dependencies has been shown to yield a result, the so called universal solution, that correctly reflects all positive facts implied by the dependencies to be in the target database. Consequently the certain answer to any monotone query can be correctly evaluated on the universal solution. This is however no longer true for non-monotonic queries, e.g. queries containing negation. Some proposals to handle non-monotonic queries in data exchange have appeared in the literature, but their semantic justifications have been somewhat ad hoc, and as we show, they do not always yield intuitively correct answers. In this paper we formulate a closed world assumption based on possible worlds and constructively obtainable solutions. A fact will be certainly true (false) if it is true (respectively false) in all constructible solutions. We show that the space of constructible solutions can be represented as a conditional table, and that this conditional table can be obtained by a novel conditional chase on the source instance. Materializing the conditional table means that we now can correctly evaluate queries containing negation, or more generally, any generic query.

Summary of results

This paper introduces *constructible solutions* as a "good" interpretation of the set of solution for a data exchange setting and a source instance. We show that the space of the constructible solutions can be represented by a conditional table. We also prove that such a conditional table can be computed directly from the source instance and the data exchange setting. This computation is done trough a novel chase method called the *conditional chase*. The conditional table resulting from the conditional chase process can be used to compute certain answers for first order queries. The computation of such answers is shown to be, in case of richly acyclic set of dependencies, coNP-complete and it remains polynomial for the certain answer over unions of conjunctive queries.

⋆

The rest of this paper is organized as follows. Basic definitions together with the introduction of the constructible solution space are given in the next section. Section 3 contains a detailed description on how to chase conditional tables with tuple generating dependencies. Section 4 makes the link between conditional chasing, incomplete information and "classical" chasing. Comparisons with some closed world assumptions previously proposed in the data exchange literature are made in Section 5. Conclusions and further work is covered in Section 6. Due to space restrictions, proofs are omitted, as is the incorporation of equality generating dependencies.

2. PRELIMINARIES

For required background knowledge the reader should consult [1]. Throughout the paper, the symbol ω will denote the set $\{0, 1, 2, \ldots\}$.

Relational and Herbrand instances. Let Cons be a countably infinite set of *constants*, usually denoted $a, b, c, \ldots$, possibly subscripted. Let k be a positive integer. A *k-ary relation* is a (finite) subset of the set $\mathsf{Cons} \times \mathsf{Cons} \times \cdots \times \mathsf{Cons}$, k-times, also denoted Cons^k. In order to have names for

*Contact author.

LID 2011 March 25, 2011, Uppsala, Sweden

relations, such as R, R_1, R_2, etc, we assume a function I that, when applied to a "k-ary" relation name R, gives the *instance* $I(R)$ of the actual tuples from Cons^k. For a relational "schema" $\mathbf{R} = \{R_1, R_2, \ldots, R_n\}$, the database instance consists of an instance $I(R_i)$ for each relation name R_i, $i \in \{1, \ldots, n\}$. When the schema is understood a database instance is simply written as I.

In this paper we adopt a variation of modeling of relational instances, influenced by the logic programming approach to databases. From the domain Cons and $\mathbf{R}$, we build up a Herbrand structure consisting of all expressions of the form $R(a_1, a_2, \ldots, a_k)$, where R is a "k-ary" relation name in $\mathbf{R}$, and the a_i's are values in Cons. Such expressions are called *ground atoms*, or *ground tuples*. A database instance I is then simply a finite set of ground tuples, e.g. $\{R_1(a_1, a_3), R_2(a_2, a_3, a_4)\}$. We denote the set of constants occurring in an instance I with $dom(I)$.

Tuple generating dependencies. A *tuple generating dependency (tgd)* is a first order formula of the form

$$\forall \mathbf{x} : \alpha(\mathbf{x}, \mathbf{y}) \rightarrow \exists \mathbf{z}\ \beta(\mathbf{x}, \mathbf{z})$$

where α is a conjunction of atoms, β is an atom[1], and $\mathbf{x}$,$\mathbf{y}$ and $\mathbf{z}$ are vectors of variables. When α is the antecedent of a tgd, we shall sometimes conveniently regard it as a *set* of atoms, and refer to it as the *body* of the tgd. Similarly we refer to β as the *head* of the tgd. Frequently, we omit the universal quantifiers in tgd's formulae. Also, when the variables are not relevant in the context, we denote a tgd $\alpha(\mathbf{x}, \mathbf{y}) \rightarrow \exists \mathbf{z}\ \beta(\mathbf{x}, \mathbf{z})$ simply as $\alpha \rightarrow \beta$.

Let $\xi = \alpha \rightarrow \beta$ be a tuple generating dependency, and I an instance. Then we say that I *satisfies* ξ if $I \models \xi$, that is, if I is a model of ξ in the standard model theoretic sense. Equivalently, we could require that for every homomorphism h, such that $h(\alpha) \subseteq I$, there is an extension h' of h, such that $h'(\beta) \in I$. The set of all instances satisfying ξ is denoted $Sat(\xi)$, and for a set Σ of tgd's, we have $Sat(\Sigma) = \bigcap_{\xi \in \Sigma} Sat(\xi)$.

Constructible solutions. Let Σ be a finite set of tgd's. Then the set of variables occurring in Σ is finite. Consequently there is an enumeration, say $v_0, v_1, \ldots, v_n, \ldots$, of all mappings (also called valuations) from this set of variables to Cons. We define

$$ground\big(\forall \mathbf{x}, \mathbf{y}\ \alpha(\mathbf{x}, \mathbf{y}) \rightarrow \exists \mathbf{z}\ \beta(\mathbf{x}, \mathbf{z})\big) = \bigcup_{i \in \omega} \bigcup_{j \in \omega} \bigcup_{k \in \omega} \big\{\alpha(v_i(\mathbf{x}), v_j(\mathbf{y})) \rightarrow \beta(v_i(\mathbf{x}), v_k(\mathbf{z}))\big\}.$$

We assume an enumeration $\xi_1, \xi_2, \ldots$, of the dependencies under consideration. If $\xi_p = \alpha(\mathbf{x}, \mathbf{y}) \rightarrow \exists \mathbf{z}\ \beta(\mathbf{x}, \mathbf{z})$, a dependency $\alpha(v_i(\mathbf{x}), v_j(\mathbf{y})) \rightarrow \beta(v_i(\mathbf{x}), v_k(\mathbf{z}))$ in $ground(\xi_p)$ can uniquely be denoted $\xi_{p,i,j,k}$. For a set Σ of tuple generating dependencies we have $ground(\Sigma) = \bigcup_{\xi_p \in \Sigma} ground(\xi_p)$. The set $ground(\Sigma)$ now consists of all ground instances of the dependencies in Σ. The elements of $ground(\Sigma)$ are called *ground dependencies*, or *ground tgd's*. A set G of ground tgd's is said to be *finitely generated*, if $G \subseteq ground(\Sigma)$, for some finite set Σ of tgd's.

Let I be an instance, G a set of finitely generated ground dependencies, and $\xi_{p,i,j,k} = \alpha \rightarrow \beta$ a ground dependency in G. Then $\alpha \rightarrow \beta$ *applies to* (I, G) if $\alpha \subseteq I$. Furthermore, we say that $\xi_{p,i,j,k} = \alpha \rightarrow \beta$ *derives* (I', G') from (I, G), if

$$(I', G') = \big(I \cup \{\beta\}, G \setminus \{\xi_{p,i,j,\ell} : \ell \in \omega\}\big).$$

This relation is denoted $(I, G) \Rightarrow_{\alpha \rightarrow \beta} (I', G')$. The intuition behind deleting the dependencies $\xi_{p,i,j,\ell}$ from G is that since ξ_p was fired using valuations v_i and v_j for the body, and with $v_k(\mathbf{z})$ as a "witness" for the existential variables in the head, we do not want to fire this dependency again, using v_i and v_j.

Consider now a sequence

$$(I_0, G_0), (I_1, G_1), \ldots, (I_n, G_n), \ldots$$

where $(I_0, G_0) = (I, G)$ and $(I_i, G_i) \Rightarrow_{\alpha \rightarrow \beta} (I_{i+1}, G_{i+1})$, for some ground dependency $\alpha \rightarrow \beta \in G_i$ that applies to I_i, or where $(I_i, G_i) = (I_{i+1}, G_{i+1})$, if no $\alpha \rightarrow \beta \in G_i$ applies to I_i. We call such a sequence a *chase sequence originating from* (I, G).

A chase sequence originating from (I, G) is said to be *fair* if for all ground dependencies $\xi_{p,i,j,k} \in G$, for which there exists an $n \in \omega$ such that $\xi_{p,i,j,k}$ applies to (I_n, G_n), there also exists $m, \ell \in \omega$ such that $(I_m, G_m) \Rightarrow_{\alpha \rightarrow \beta} (I_{m+1}, G_{m+1})$, where $\alpha \rightarrow \beta = \xi_{p,i,j,\ell}$. Fairness means that all applicable dependencies are eventually fired for some instantiation of the existential variables in the head. From now one, unless stated otherwise, we consider all chase sequences to be fair.

It is easy to see that $I_i \subseteq I_{i+1}$ and that $G_i \supseteq G_{i+1}$, for each $i \in \omega$. We can thus define the limit of a chase sequence as

$$\Big(\bigcup_{i \in \omega} I_i, \bigcap_{i \in \omega} G_i\Big).$$

The notation $\mathcal{C}(I, G)$ will stand for the set of limits of *all* chase sequences originating from (I, G).

We are now ready to define $\Sigma(I)$, the set of all *constructible models* of I and Σ, as

$$\Sigma(I) = \{J\ :\ (J, G) \in \mathcal{C}\big(I, ground(\Sigma)\big), G \subseteq ground(\Sigma)\}.$$

It is easy to see that the definition of $\Sigma(I)$ does not depend on the particular enumeration of the valuations of the variables in the dependencies in Σ. Note also that $\Sigma(I)$ may contain finite as well as infinite ground instances.

The following lemma follows directly from the definition of $\Sigma(I)$.

LEMMA 1. $\Sigma(I) \subseteq Sat(\Sigma)$.

The definition of function $\Sigma(I)$ is extended to a set of instances $\mathcal{I}$ point-wise as $\Sigma(\mathcal{I}) = \bigcup_{I \in \mathcal{I}} \Sigma(I)$.

Tableaux and their unification. Let Vars be a countably infinite set, disjoint from the set of constants. Elements in Vars are called *variables*, and are denoted $X, Y, Z, \ldots$, possibly subscripted. We can then also allow *non-ground atoms*, i.e. expressions of the form $R(a, X, b, \ldots, X, Y)$. A *tableau* T is then a finite set of atoms (ground, or non-ground). The set of variables and constants in a tableau is denoted $dom(T)$. The number of atoms, also called *tuples*, in a tableau T is denoted $|T|$.

Let T and U be tableaux. A mapping h from $dom(T)$ to $dom(U)$, that is the identity on Cons, extended componentwise, is called a *homomorphism* from T to U if $h(T) \subseteq U$. If $h(T) = U$, the mapping h is an *epimorphism*. If the range of h is included in Cons, the mapping h is called a *ground homomorphism*, or *valuation*. The homomorphism

[1]Note that there is no loss of generality in assuming that the head consists of a single atom, cf. [2].

h from $dom(T)$ to $dom(U)$ is *full* if for any k-ary relation R and all $a_i \in \mathsf{Cons} \cup \mathsf{Vars}$, $R(a_1, a_2, \ldots, a_k) \in T$ iff $R(h(a_1), h(a_2), \ldots, h(a_k)) \in U$. An injective full homomorphism is also called an *embedding*. If there is an injective full epimorphism from T to U, the two tableaux are said to be *isomorphic*. Furthermore, if h is a mapping on $dom(U)$, such that $h(U) \subseteq U$, it is called an *endomorphism* or *retraction*. A retraction h on U is called a *proper retraction* if $h(U) \subset U$. We denote with $core(U)$ a tableau such that there exists an endomorphism h with $h(U) = core(U)$, and there does not exist a proper retraction g of $core(U)$ with $g(core(U)) \subset core(U)$. As shown in [5], the core of a tableau is unique up to isomorphism. If for a tableau U, we have that $U = core(U)$, then U is called a *core* . We shall also need the concept of an h-core of a tableau, for an endomorphism h.

DEFINITION 1. ***Endomorphic core.*** Let T be a tableau, and h an endomorphism on $dom(T)$. Then we say that T is an *h-endomorphic core*, or simply *h-core*, if there is no $T' \subset T$, such that $h(T') = h(T)$.

Clearly a tableau T is a core if and only if T is an h-core for all endomorphisms h on T. Next we introduce unifiers of tableaux.

Let T and U be tableaux. A *unifier* for T and U, if it exists, is a pair (θ_1, θ_2), where θ_1 is a homomorphism from $dom(T)$ to $dom(U)$, and θ_2 is an endomorphism on $dom(U)$, such that $\theta_1(T) = \theta_2(U)$. Note the asymmetrical role of T and U: a unifier for (T, U) is not necessarily a unifier for (U, T).

A unifier (θ_1, θ_2) is *more general* than a unifier (γ_1, γ_2), if there is a mapping f on $dom(U)$, identity on constants, such that $\gamma_1 = f \circ \theta_1$ and $\gamma_2 = f \circ \theta_2$. Clearly this defines a partial pre-order on unifiers, and a partial order on equivalence classes of isomorphic unifiers. Unifiers (θ_1, θ_2) and (γ_1, γ_2) of (T, U) are considered isomorphic if $(\theta_1(T), \theta_2(U))$ is isomorphic with $(\gamma_1(T), \gamma_2(U))$. A unifier (θ_1, θ_2) is a *most general unifier (mgu)* for tableaux T and U, if all unifiers (γ_1, γ_2) of T and U that are more general than (θ_1, θ_2) actually are isomorphic with (θ_1, θ_2). We denote by $mgu(T, U)$ the set of (representatives of the) equivalence classes of all mgu's of T and U.

For example, consider $T = \{R(X, Y), R(Y, Z)\}$ and $U = \{R(a, V), R(W, b)\}$, two tableaux. In this case, the only mgu of T and U is given by the equivalence class containing the following two obviously isomorphic instances:

$$\big\{\{R(a, V), R(V, b)\}, \{R(a, W), R(W, b)\}\big\}.$$

As is well known, if a set of atoms (a tableau) is unifiable, then it has a most general unifier that is unique up to isomorphism. On the other hand, when unifying two tableaux, there might be several non-isomorphic mgu's. For example, let $T = \{R(X, Y), R(Y, Z)\}$ and $U = \{R(a, V), R(b, W)\}$. Then (θ_1, θ_2), with $\theta_1 = \{X/a, Y/b, Z/W\}$ and $\theta_2 = \{V/b\}$ is an mgu for T and U. On the other hand, the unifier (γ_1, γ_2), with $\gamma_1 = \{X/b, Y/a, Z/V\}$ and $\gamma_2 = \{W/a\}$ is also an mgu for T and U. It is easy to see that (γ_1, γ_2) and (θ_1, θ_2) are not isomorphic.

We want an upper bound for the complexity of the set $mgu(T, U)$. The next lemma follows directly from Lemma 1.

LEMMA 2. *Let T and U be tableaux, and (θ_1, θ_2) a most general unifier of T and U. If U is θ_2-core, then $|U| \leq |T|$.*

We now have

PROPOSITION 1. *Let $|U| \leq |T| = c$, and let k be the arity of the widest atom in $T \cup U$. Then the set $mgu(T, U)$ can be computed in time $O(ck^{ck})$.*

Conditional tables. A conditional table (c-table) [9] is a pair (T, φ), where T is a tableau, and φ is a mapping that associates a *local condition* $\varphi(t)$ with each tuple $t \in T$. A (local) condition is a Boolean formula built up from atoms of the form $x = y$, $x = a$, and $a = b$ for $x, y \in \mathsf{Vars}$, and $a, b \in \mathsf{Cons}$. An atomic equality of the form $a = a$, for $a \in \mathsf{Cons}$ represents the logical constant **true**, and for two distinct constants a and b, the equality $a = b$ represents **false**. If $\varphi(t) = \mathbf{true}$, for all $t \in T$, the conditional table (T, φ) is sometimes denoted simply with T. A conditional table (T, φ) *represents* a set of possible worlds (ground instances, complete databases). For this, let v be a *valuation*, that is, a ground homomorphism. Then

$$v(T, \varphi) = \{v(t) : t \in T, \text{ and } v\big(\varphi(t)\big) = \mathbf{true}\}.$$

The set of possible worlds represented by (T, φ) is

$$Rep(T, \varphi) = \{v(T, \varphi) : v \text{ is a valuation}\}.$$

When it comes to dependencies, a conditional table (T, φ) is said to *satisfy* a set Σ of dependencies, if all possible worlds it represents satisfy Σ, that is, if $Rep(T, \varphi) \subseteq Sat(\Sigma)$.

For formulae φ and ϕ, we denote the logical implication between φ and ϕ with $\varphi \models \phi$, and their logical equivalence with $\varphi \equiv \phi$. We shall also need the following partial order on conditional tables:

$(T, \varphi) \subseteq (U, \phi)$, means that $T \subseteq U$, and $\forall t \in T, \varphi(t) \models \phi(t)$.

Let (T, φ) and (U, ψ) be c-tables. Consider the following binary union operation: $(T, \varphi) \cup (U, \phi) =_{\mathsf{def.}} (T \cup U, \varphi \vee \phi)$, where

$$\varphi \vee \phi\,(t) = \begin{cases} \varphi(t) \vee \phi(t) & \text{if } t \in T \cap U, \\ \varphi(t) & \text{if } t \in T \setminus U, \\ \phi(t) & \text{if } t \in U \setminus T. \end{cases}$$

Clearly, this operation is a least upper bound of the partial order on conditional tables defined above.

Two conditional tables are *equivalent* if the represent the same set of possible worlds. The following definition states a stronger relationship between c-tables.

DEFINITION 2. Let (T, φ) and (U, ϕ) be conditional tables. We say that they are *isomorphic* if there is an injective epimorphism h, such that $h(T) = U$, and $\varphi(h(t)) \equiv \phi(t)$, for all $t \in T$. This is denoted $(T, \varphi) \cong (U, \phi)$.

PROPOSITION 2. *Let (T, φ) and (U, ϕ) be conditional tables. If $(T, \varphi) \cong (U, \phi)$, then $Rep(T, \varphi) = Rep(U, \phi)$.*

3. THE CONDITIONAL CHASE

In this section we generalize the classical chase procedure to work on conditional tables. For this we need the following central concept of a trigger.

DEFINITION 3. Let Σ be a set of tgd's, and (T, φ) a conditional table. A *trigger for Σ on (T, φ)* is a tuple $\tau = (\xi, \theta, T')$, where $\xi \in \Sigma$, and $\theta = (\theta_1, \theta_2)$ is an mgu that unifies the body of dependency ξ with T', where $T' \subseteq T$, and T' is a θ_2-core. The set $trigg_\Sigma(T, \varphi)$ contains the set of all triggers for Σ on (T, φ).

Figure 1: Conditional tables from Example 2

(U_1, ψ_1)

t	$\psi_1(t)$
$R(X,b)$	**true**
$R(b,c)$	**true**
$R(Y,d)$	**true**
$T(X,c)$	**true**

(U_2, ψ_2)

t	$\psi_2(t)$
$R(X,b)$	**true**
$R(b,c)$	**true**
$R(Y,d)$	**true**
$T(X,c)$	**true**
$T(X,d)$	$Y = b$

(U_3, ψ_3)

t	$\psi_3(t)$
$R(X,b)$	**true**
$R(b,c)$	**true**
$R(Y,d)$	**true**
$T(X,c)$	**true**
$T(X,d)$	$Y = b$
$T(b,d)$	$Y = c$

(U_4, ψ_4)

t	$\psi_4(t)$
$R(X,b)$	**true**
$R(b,c)$	**true**
$R(Y,d)$	**true**
$T(X,c)$	**true**
$T(X,d)$	$Y = b$
$T(b,d)$	$Y = c$
$T(Y,b)$	$X = d$

(U_5, ψ_5)

t	$\psi_5(t)$
$R(X,b)$	**true**
$R(b,c)$	**true**
$R(Y,d)$	**true**
$T(X,c)$	**true**
$T(X,d)$	$Y = b$
$T(b,d)$	$Y = c$
$T(Y,b)$	$X = d$
$T(b,b)$	$X = c$

EXAMPLE 1. Consider the conditional table (T, φ) with the following tabular representation:

t	$\varphi(t)$
$R(X,b)$	**true**
$R(b,c)$	**true**
$R(Y,d)$	**true**

Let $\Sigma = \{\xi_1, \xi_2\}$, where tuple generating dependency ξ_1 is $R(x,y), R(y,z) \rightarrow T(x,z)$ and ξ_2 is $T(x,x) \rightarrow R(x,x)$. Then $trigg_\Sigma(T,\varphi) = \{\tau_1, \tau_2, \tau_3, , \tau_4, \tau_5\}$, where

$$\begin{aligned}
\tau_1 &= (\xi_1, (\{x/X, y/b, z/c\}, \{\}), \{R(X,b), R(b,c)\}),\\
\tau_2 &= (\xi_1, (\{x/X, y/b, z/d\}, \{Y/b\}), \{R(X,b), R(Y,d)\}),\\
\tau_3 &= (\xi_1, (\{x/b, y/c, z/d\}, \{Y/c\}), \{R(b,c), R(Y,d)\}),\\
\tau_4 &= (\xi_1, (\{x/Y, y/d, z/b\}, \{X/d\}), \{R(Y,d), R(X,b)\}),\\
\tau_5 &= (\xi_1, (\{x/c, y/c, z/b\}, \{X/c\}), \{R(b,c), R(X,b)\}).
\end{aligned}$$

Note that ξ_2, does not generate any triggers, as there are no tuples over relation name T. ◂

As applying a trigger to a c-table will generate new variables (unless the tgd is total), we need a mechanism for controlling this generation in such a way that the new variables are true representatives of the implicit Skolemization taking place. Let $\tau = (\xi, (\theta_1, \theta_2), T')$ be a trigger, where $\xi = \alpha(\mathbf{x}, \mathbf{y}) \rightarrow \exists \mathbf{z}\ \beta(\mathbf{x}, \mathbf{z})$. For each z in $\mathbf{z}$, consider a function $z \mapsto z_\tau$, such that if $\rho = (\xi, (\theta_1, \theta_3), T'')$, then $z_\tau = z_\rho$. Clearly such a function exists (assuming `Vars` can be well ordered).

The following abbreviation will be useful. Let (T, φ) be a conditional tableau. Then we define

$$\curlywedge(T, \varphi) \quad =_{\mathsf{def}} \quad \bigwedge_{t \in T} \varphi(t).$$

Also, when θ is finite partial mapping from Vars to the set $\mathsf{Vars} \cup \mathsf{Const}$ we shall use the abbreviation

$$\curlywedge(\theta) \quad =_{\mathsf{def}} \quad \bigwedge_i X_i = \theta(X_i),$$

where the X_i's are all the variables in the domain of θ.

DEFINITION 4. Let $\tau = (\alpha \rightarrow \beta, (\theta_1, \theta_2), T') \in trigg_\Sigma(T, \varphi)$. We denote with θ_1' the extension of θ_1 that maps each existentially quantified variable z from the tgd to the variable z_τ. We say that the conditional table (U, ϕ) *is obtained from* (T, φ) *by applying the trigger* τ if (U, ϕ) contains all the c-tuples from T together with their possibly modified local conditions, and possibly a new c-tuple, as follows:

If T contains a tuple t that is syntactically equal to $\theta_1'(\beta)$, ***then*** for (U, ϕ) the local condition of t is changed to

$$\phi(t) \quad =_{\mathsf{def}} \quad \varphi(t) \vee \curlywedge(\tau),$$

where $\curlywedge(\tau)$ denotes [2] the formula $\curlywedge(T', \varphi) \wedge \curlywedge(\theta_2)$, ***else*** add the c-tuple $\theta_1'(\beta)$ with local condition

$$\phi(\theta_1'(\beta)) \quad =_{\mathsf{def}} \quad \curlywedge(\tau).$$

If (U, ϕ) is obtained from (T, φ) in this way we write

$$(T, \varphi) \rightarrow_\tau (U, \phi),$$

and call the transformation a *conditional chase micro-step.*◂

Having a table (T, φ) and a finite set Σ of tgd's, the set of triggers $trigg_\Sigma(T, \varphi)$ is obviously finite. Consider a sequence

$$(T_0, \varphi_0) \rightarrow_{\tau_1} (T_1, \varphi_1) \rightarrow_{\tau_2} \cdots \rightarrow_{\tau_n} (T_n, \varphi_n),$$

where $\tau_1, \tau_2, \ldots, \tau_n$ is an ordering of all the triggers from the set $trigg_\Sigma(T_0, \varphi_0)$. We call it a *conditional chase micro-sequence for* (T, φ) *using* $trigg_\Sigma(T, \varphi)$.

EXAMPLE 2. Let us continue with the c-table from example 1. The c-table (U_1, ψ_1) in Figure 1 is obtained from (T, φ) in one chase micro-step by applying the trigger τ_1. That is $(T, \varphi) \rightarrow_{\tau_1} (U_1, \psi_1)$. Next, by applying trigger τ_2 to (U_1, ψ_1), we obtain (U_2, ψ_2). Note that the last tuple has a local condition generated by the substitution $\{Y/b\}$ from τ_2. In the same way we apply τ_3 to (U_2, ψ_2) obtaining (U_3, ψ_3). Then by applying τ_4 c-table (U_4, ψ_4) is obtained, to which we apply τ_5 obtaining c-table (U_5, ψ_5). This gives the following sequence of chase micro-steps:

$$\begin{aligned}
(T, \varphi) &\rightarrow_{\tau_1} (U_1, \psi_1) \rightarrow_{\tau_2} (U_2, \psi_2) \rightarrow_{\tau_3}\\
&(U_3, \psi_3) \rightarrow_{\tau_4} (U_4, \psi_4) \rightarrow_{\tau_5} (U_5, \psi_5),
\end{aligned}$$

where the c-tables in question are displayed in Figure 1.◂

We note that the order in which the triggers are applied in a conditional micro-sequence does affect the outcome, but we shall see that in the end the order will not matter.

After a micro-sequence, it is not guaranteed that the result will satisfy the dependencies. Then additional triggers will be generated. We therefore abstract a micro-sequence into a macro-step, as follows.

[2] Note that the formula $\curlywedge(\tau)$ is uniquely determined by τ. Intuitively it represents the ("abducted") conditions induced by τ and necessary for the body α to apply in the standard sense.

DEFINITION 5. Let (T, φ) and (U, ϕ) be conditional tables, and Σ be a set of tgd's. We write $(T, \varphi) \Rightarrow_\Sigma (U, \phi)$, if (U, ϕ) is obtained from (T, φ) by applying all micro-steps generated from $trigg_\Sigma(T, \varphi)$. The transformation from (T, φ) to (U, ϕ) is called a *conditional chase macro-step using* Σ.

The macro-step is monotone, that is, if $(T, \varphi) \Rightarrow_\Sigma (U, \phi)$, then $(T, \varphi) \subseteq (U, \phi)$.

We are now ready to introduce the concept of a conditional chase-sequence.

DEFINITION 6. Let (T, φ) be a c-table and Σ a set of tgd's. A sequence $(T_0, \varphi_0), (T_1, \varphi_1), \ldots, (T_n, \varphi_n), \ldots$ of c-tables, inductively constructed as

$$\begin{aligned} (T_0, \varphi_0) &= (T, \varphi), \\ (T_{i+1}, \varphi_{i+1}) &= (U, \phi), \text{ where } (T_i, \varphi_i) \Rightarrow_\Sigma (U, \phi), \\ (T_\omega, \varphi_\omega) &= \bigcup_{i \in \omega} (T_i, \varphi_i), \end{aligned}$$

is called a *conditional chase sequence* associated with (T, φ) and Σ.

EXAMPLE 3. Continuing Example 2, let $(T_0, \varphi_0) = (T, \varphi)$ and $(T_1, \varphi_1) = (U_5, \psi_5)$. Then $(T_0, \varphi_0) \Rightarrow_\Sigma (T_1, \varphi_1)$. Next, $trigg_\Sigma(T_1, \varphi_1) = \{\tau_1, \tau_2, \tau_3, \tau_4, \tau_5, \tau_6, \tau_7, \tau_8\}$, where $\tau_1, \ldots, \tau_5$ are as before, and

$$\begin{aligned} \tau_6 &= (\xi_2, (\{x/d, y/d\}, \{X/d\}), \{T(X, d)\}), \\ \tau_7 &= (\xi_2, (\{x/b, y/b\}, \{Y/b\}), \{T(Y, b)\}), \\ \tau_8 &= (\xi_2, (\{x/b, y/b\}, \{\}), \{T(b, b)\}), \end{aligned}$$

By applying the triggers in the order $\tau_1, \tau_2, \ldots, \tau_8$ we obtain another micro-sequence ending in (T_2, φ_2), shown below. In other words, $(T_1, \varphi_1) \Rightarrow_\Sigma (T_2, \varphi_2)$.

(T_2, φ_2)

t	$\varphi_2(t)$
$R(X, b)$	**true**
$R(b, c)$	**true**
$R(Y, d)$	**true**
$T(X, c)$	**true**
$T(X, d)$	$Y = b \vee Y = b$
$T(b, d)$	$Y = c \vee Y = c$
$T(Y, b)$	$X = d \vee X = d$
$R(d, d)$	$(Y = b \vee Y = b) \wedge (X = d)$
$R(b, b)$	$((X = d \vee X = d) \wedge (Y = b)) \vee (X = c)$

Note that the conditional chase process is not finished yet, as for table (T_2, φ_2), beside the previous triggers the following new triggers need to be considered as well:

$$\begin{aligned} \tau_9 &= (\xi_1, (\{x/X, y/b, z/b\}, \{\}), \{R(X, b), R(b, b)\}), \\ \tau_{10} &= (\xi_1, (\{x/b, y/b, z/b\}, \{X/b\}), \{R(b, b), R(X, b)\}), \\ \tau_{11} &= (\xi_1, (\{x/Y, y/d, z/d\}, \{\}), \{R(Y, d), R(d, d)\}), \\ \tau_{12} &= (\xi_1, (\{x/d, y/d, z/d\}, \{Y/d\}), \{R(d, d), R(Y, d)\}). \end{aligned}$$

By applying $\tau_1, \ldots, \tau_{12}$ the micro-sequence will end up in the table (T_3, φ_3) below. Thus $(T_0, \varphi_0), (T_1, \varphi_1), (T_2, \varphi_2), (T_3, \varphi_3)$ is a conditional chase sequence associated with (T, φ) and Σ. For readability, (T_3, φ_3) is shown below in a "cleaner" isomorphically equivalent form.◂

(T_3, φ_3)

t	$\varphi_3(t)$
$R(X, b)$	**true**
$R(b, c)$	**true**
$R(Y, d)$	**true**
$T(X, c)$	**true**
$T(X, d)$	$Y = b$
$T(b, d)$	$Y = c$
$T(Y, b)$	$X = d$
$R(d, d)$	$Y = b \wedge X = d$
$R(b, b)$	$(X = d \wedge Y = b) \vee (X = c)$
$T(X, b)$	$(X = d \wedge Y = b) \vee (X = c)$
$T(Y, d)$	$Y = b \wedge X = d$

The following result shows that no matter what order we choose to apply the triggers in the micro-sequences, the results of the corresponding conditional chase sequences are isomorphically equivalent.

THEOREM 1. *Let (T, φ) be a conditional table and Σ a set of tgd's. Let $(T_0, \varphi_0), (T_1, \varphi_1), \ldots, (T_{n-1}, \varphi_{n-1}), (T_n, \varphi_n), \ldots$ and $(T'_0, \varphi'_0), (T'_1, \varphi'_1), \ldots, (T'_n, \varphi'_n), \ldots$ be two distinct conditional chase sequences with different orders of triggers in their micro-sequences. Then $(T_\omega, \varphi_\omega) \cong (T'_\omega, \varphi'_\omega)$. Even more, $T_\omega = T'_\omega$.*

The theorem means that distinct orders in the micro-sequences only affect the syntactic form of the local conditions. The atoms in the table will be syntactically equal, and the corresponding local conditions equivalent.

We shall use the notation $Chase_\Sigma(T, \varphi)$ for one representative of $(T_\omega, \varphi_\omega)$ in the conditional chase sequence, starting with (T, φ) and Σ.

We conclude this section by stating the main property of the conditional chase.

THEOREM 2. $Rep(Chase_\Sigma(T, \varphi)) = \Sigma(Rep(T, \varphi))$.

4. CHASING GENERALIZED INSTANCES

In this section we compare and contrast the conditional chase with the various chases that have previously appeared in the literature, in particular in the context of data exchange. In data exchange the source instance is assumed to be a complete instance [4]. We therefore restrict ourselves in some places to the case where the initial table to be chased is actually a complete instance. We shall denote a complete instance I as a conditional table $(I, \mathbf{true})$. Since $Rep(I, \mathbf{true}) = \{I\}$ we have $Rep(Chase_\Sigma(I, \mathbf{true})) = \Sigma(I)$.

Incomplete Information. In the recent chase-literature a "generalized instance" is allowed to contain both variables and constants. On the other hand, query answers are defined in terms of instances without nulls. Libkin [10] points out that there really is no semantic justification for this discrepancy, and that ignoring the fact that we are dealing with incomplete information has resulted in many confusions and anomalies in the literature. Since the appearance of Libkin's paper the situation has improved, but the connections to incomplete information are in many cases still scattered ad hoc observations. This section represents an effort to systematically relate recent results on the chase to the theory of incomplete information that our conditional chase is based upon.

An incomplete database is conceptually a set $\mathcal{I}$ of complete instances, or *possible worlds* I. Given a query Q and an incomplete database $\mathcal{I}$, the result of Q on $\mathcal{I}$ is $Q(\mathcal{I}) = \{Q(I) : I \in \mathcal{I}\}$. To this *exact answer* [6] there are two approximations [9],

$$\Box Q(\mathcal{I}) = \bigcap\{Q(I) : I \in \mathcal{I}\} \text{ and } \Diamond Q(\mathcal{I}) = \bigcup\{Q(I) : I \in \mathcal{I}\},$$

called the *certain* and the *possible answers*, respectively. We now introduce the notion of *co-initial* sets of possible worlds. This notion will provide us with a characterization of when two incomplete instances are "certain-answer equivalent" for *monotone queries*. We will couple this with a characterization of "certain answer equivalence" with respect to *first order queries*.

DEFINITION 7. Let $\mathcal{I}$ and $\mathcal{J}$ be sets of possible worlds. The we say that $\mathcal{I}$ *precedes* $\mathcal{J}$, if for all $J \in \mathcal{J}$, there is an $I \in \mathcal{I}$, such that $I \subseteq J$. This is denoted $\mathcal{I} \preceq \mathcal{J}$. If both $\mathcal{I} \preceq \mathcal{J}$, and $\mathcal{J} \preceq \mathcal{I}$, we say that $\mathcal{I}$ and $\mathcal{J}$ are *co-initial*, and write $\mathcal{I} \approx \mathcal{J}$.

It is easy to see that co-initiality is an equivalence relation.

We now have two partial orders on sets of possible worlds, namely $\mathcal{I} \preceq \mathcal{J}$, and the more obvious $\mathcal{I} \subseteq \mathcal{J}$. Intuitively, $\mathcal{I} \preceq \mathcal{J}$ means that $\mathcal{I}$ has less positive or monotone information than $\mathcal{J}$. The order $\mathcal{I} \subseteq \mathcal{J}$ means that $\mathcal{I}$ has less absolute information than $\mathcal{J}$. This intuition is justified by the following folklore theorem.

THEOREM 3. *Suppose that there is a least one finite relation in $\mathcal{I} \cup \mathcal{J}$. Then*

1. *$\mathcal{I} \preceq \mathcal{J}$ ($\mathcal{I} \approx \mathcal{J}$) if and only if $\Box Q(\mathcal{I}) \subseteq \Box Q(\mathcal{J})$ (resp. $\Box Q(\mathcal{I}) = \Box Q(\mathcal{J})$) for all monotone queries Q.*
2. *$\mathcal{I} \subseteq \mathcal{J}$ ($\mathcal{I} = \mathcal{J}$) if and only if $\Box Q(\mathcal{I}) \subseteq \Box Q(\mathcal{J})$ (resp. $\Box Q(\mathcal{I}) = \Box Q(\mathcal{J})$) for all first order queries Q.*

The theorem assumes that queries can mention constants (elements in `Cons`). If this is disallowed, we need to replace equality of instances with general isomorphism in the definitions of $\preceq$ and $\subseteq$. A general isomorphism is defined as an isomorphism that is not necessarily the identity on `Cons`.

Generalized instances. As recently noted in e.g. [7, 10], instances with nulls serve as a representation of sets of ground instances (instances without nulls). A generalized instance is thus actually a classical tableau. In order to clearly distinguish between generalized and ground instances we shall write the former in blackboard bold font and continue to use italics for ground instances. Thus $\mathbb{I}$ denotes a generalized instance, while I denotes a ground one. Homomorphisms on generalized instances are defined exactly as for tableaux. We can let the set of all homomorphisms induce partial preorder on the set of all generalized instances, by stating that $\mathbb{I}$ is "hom-less than" $\mathbb{J}$ if there exists a homomorphism from $\mathbb{I}$ to $\mathbb{J}$. Using the notation of [3, 4] we write this as $\mathbb{I} \to \mathbb{J}$, or as $h : \mathbb{I} \to \mathbb{J}$, when we want to emphasize a particular homomorphism h. Homomorphic equivalence between $\mathbb{I}$ and $\mathbb{J}$ is denoted $\mathbb{I} \leftrightarrow \mathbb{J}$. By identifying homomorphically equivalent generalized instances, $\mathbb{I} \to \mathbb{J}$ will denote a true partial order.

Since a generalized instance $\mathbb{I}$ also is a tableau, $Rep(\mathbb{I})$ is well defined, and contains, as expected, all the ground instances obtained by substituting in $\mathbb{I}$ the variables with constants in every possible way. The following lemma provides the basic connection between generalized instances and incomplete information.

LEMMA 3. *Let $\mathbb{I}$ and $\mathbb{J}$ be generalized instances. Then*

1. *$\mathbb{I} \to \mathbb{J}$ if and only if $Rep(\mathbb{I}) \preceq Rep(\mathbb{J})$.*
2. *$\mathbb{I} \leftrightarrow \mathbb{J}$ if and only if $Rep(\mathbb{I}) \approx Rep(\mathbb{J})$.*

A generalized instance $\mathbb{I}$ is said to satisfy a set Σ of dependencies, if $\mathbb{I}$ as an interpretation satisfies Σ, that is if $\mathbb{I} \models \Sigma$ in the model-theoretic sense. Given a ground instance I and a set Σ of dependencies, the set of all generalized instances $\mathbb{K}$ with universe $\mathsf{Cons} \cup \mathsf{Vars}$ for which (1): there is a homomorphism $h : I \to \mathbb{K}$, and (2): $I \cup \mathbb{K} \models \Sigma$, is called the *set of solutions for I and Σ*, and is denoted $Sol(I, \Sigma)$. A solution $\mathbb{J} \in Sol(I, \Sigma)$, is said to be `hom`-*universal* if (3): for each solution $\mathbb{K} \in Sol(I, \Sigma)$, there is a homomorphism $h : \mathbb{J} \to \mathbb{K}$.

Let $Sol_{\mathtt{g}}(I, \Sigma)$ denote the subset of all ground instances in $Sol(I, \Sigma)$. The next lemma relates the above notion of solutions to our notion of constructible solutions and gives a characterization of universal solutions.

LEMMA 4. *Let I be a ground instance, and Σ a set of dependencies. Then*

1. *$\Sigma(I) \subset Sol_{\mathtt{g}}(I, \Sigma)$, and $Sol_{\mathtt{g}}(I, \Sigma) \approx \Sigma(Rep(I))$.*
2. *$\mathbb{J}$ is `hom`-universal for I and Σ iff $Rep(\mathbb{J})$ is the $\preceq$-smallest set such that $\{I\} \preceq Rep(\mathbb{J}) \preceq Sol_{\mathtt{g}}(I, \Sigma)$.*

The standard chase. Let $\mathbb{I}$ be a generalized instance, ξ a tuple generating dependency $\forall \mathbf{x}, \mathbf{y} : \alpha(\mathbf{x}, \mathbf{y}) \to \exists \mathbf{z}\ \beta(\mathbf{x}, \mathbf{z})$, and let $\mathbf{a}, \mathbf{b}$ be vectors of constants. Then we say that ξ *fails on $\mathbb{I}$ with $\mathbf{a}, \mathbf{b}$* if $\mathbb{I} \models \alpha(\mathbf{a}, \mathbf{b})$, and $\mathbb{I} \not\models \exists \mathbf{z}\ \beta(\mathbf{a}, \mathbf{z})$. The *standard chase step* [3, 4] from generalized instance $\mathbb{I}$ to a generalized instance $\mathbb{J}$ can be taken if ξ *fails* on $\mathbb{I}$ with some $\mathbf{a}, \mathbf{b}$ and $\mathbb{J} = \mathbb{I} \cup \{\beta(\mathbf{a}, \mathbf{z}')\}$, where $\mathbf{z}'$ is a vector of fresh variables. The standard chase step is denoted $\mathbb{I} \xrightarrow{\xi, \mathbf{a}, \mathbf{b}} \mathbb{J}$.

The *standard chase* [3, 4] on a generalized instance $\mathbb{I}$ is defined as a sequence $\mathbb{I} = \mathbb{I}_0, \mathbb{I}_1, \mathbb{I}_2, \ldots, \mathbb{I}_n, \ldots$ where for each i we have that $\mathbb{I}_i \xrightarrow{\xi, \mathbf{a}, \mathbf{b}} \mathbb{I}_{i+1}$, for some $\xi \in \Sigma$ and vectors $\mathbf{a}$ and $\mathbf{b}$ of constants. In the rest of this paper, we consider all the sequences as *fair*, that is, if there is an instance $\mathbb{I}_i$ for which ξ with $\mathbf{a}$,$\mathbf{b}$ fails, then there exists a $j \geq i$ such that ξ with $\mathbf{a}$,$\mathbf{b}$ does not fail for $\mathbb{I}_j$. The standard chase result for the sequence is $\cup_{i \in \omega} \mathbb{I}_i$. Note that a chase sequence is not necessarily unique, since there is nondeterminism involved in each step when choosing ξ, $\mathbf{a}$, $\mathbf{b}$, and $\mathbf{z}'$. It is however known [3, 4] that all standard chase results are homomorphically equivalent. Let therefore $Chase_{\Sigma}^{\mathtt{hom}}(\mathbb{I})$ denote the set of all standard chase results. The following is the fundamental result for the standard chase.

THEOREM 4. ([3, 4]) *Let I be a ground instance. Any element in $Chase_{\Sigma}^{\mathtt{hom}}(I)$ is a universal solution for $Sol(I, \Sigma)$.*

If $\mathscr{I}$ is a set of generalized instances we define the set of possible worlds $Rep(\mathscr{I}) = \cup\{Rep(\mathbb{I}) : \mathbb{I} \in \mathscr{I}\}$. We can now compare the standard chase with our conditional chase. Let us denote the conditional chase with $Chase_{\Sigma}^{\mathtt{cond}}(.,.)$. Recall from Theorem 2 that $\Sigma(Rep(T, \varphi)) = Rep(Chase_{\Sigma}^{\mathtt{cond}}(T, \varphi))$.

THEOREM 5.

1. $Rep(Chase_\Sigma^{\mathtt{hom}}(\mathbb{I})) \preceq Rep(Chase_\Sigma^{\mathtt{cond}}(\mathbb{I}, \mathbf{true}))$.
2. $Rep(Chase_\Sigma^{\mathtt{cond}}(\mathbb{I}, \mathbf{true})) \not\preceq Rep(Chase_\Sigma^{\mathtt{hom}}(\mathbb{I}))$.

The oblivious chase. The oblivious chase [2], proceeds as the standard chase, except that each dependency is fired once for each homomorphism that maps its body into the instance. The firing happens irrespectively of whether the head is satisfied or not. Calì et al. [2] show that the results of the standard and the oblivious chases are homomorphically equivalent, even if one (or both) of them is nonterminating. We denote with $Chase_\Sigma^{\mathtt{obl}}(\mathbb{I})$ one representative instance of the oblivious chase.

THEOREM 6. [2] $Chase_\Sigma^{\mathtt{obl}}(\mathbb{I}) \leftrightarrow Chase_\Sigma^{\mathtt{hom}}(\mathbb{I})$.

COROLLARY 1. $Rep(Chase_\Sigma^{\mathtt{obl}}(\mathbb{I})) \approx Rep(Chase_\Sigma^{\mathtt{hom}}(\mathbb{I}))$.

The core chase. The core chase was introduced by Deutch et al. in [3]. A core chase step consists of firing all dependencies in parallel in all possible ways, and then computing the core of the result. Formally, a core chase step for an instance $\mathbb{I}$ and set of tgd's Σ gives instance $\mathbb{J} = core(\mathbb{K})$, where

$$\mathbb{K} = \bigcup \{\mathbb{L} \,:\, \mathbb{I} \xrightarrow{\xi,\mathbf{a},\mathbf{b}} \mathbb{L},\ \xi \in \Sigma,\ \mathbf{a},\mathbf{b} \in dom(\mathbb{I})\}.$$

A sequence $\mathbb{I}_0, \mathbb{I}_1, \mathbb{I}_2, \ldots, \mathbb{I}_n, \ldots$, is a *core chase sequence* if $\mathbb{I}_i$ is obtained from $\mathbb{I}_{i-1}$ and Σ in one core chase step. As the result of a core chase step is unique up to isomorphism, we denote with $Chase_\Sigma^{\mathtt{core}}(\mathbb{I})$ one representative instance.

THEOREM 7.

1. $Rep(Chase_\Sigma^{\mathtt{core}}(\mathbb{I})) \subset Rep(Chase_\Sigma^{\mathtt{hom}}(\mathbb{I}))$.
2. $Rep(Chase_\Sigma^{\mathtt{core}}(\mathbb{I})) \neq Rep(Chase_\Sigma^{\mathtt{hom}}(\mathbb{I}))$.
3. $Rep(Chase_\Sigma^{\mathtt{core}}(\mathbb{I})) \approx Rep(Chase_\Sigma^{\mathtt{hom}}(\mathbb{I}))$.

As can be seen from the previous theorem and Theorem 3, the (semantic) difference between the result of the standard chase and the core chase will only become manifest when we are dealing with the closed world assumption and queries with negation.

The extended core chase. As a generalized instance cannot always adequately represent $Sol(I, \Sigma)$, Deutch et a. introduced in [3] the extended core chase. Without going into details of the extended core chase, we will, for comparative purposes, give the give the characterization of the result of this chase in terms of emb-universal modelsets [3]. A *modelset* is a finite set $\mathscr{I}$ of generalized instances $\mathbb{I}$.

The following partial order on on modelsets is defined in [3]: $\mathscr{I}$ is "emb-smaller" than $\mathscr{J}$, denoted $\mathscr{I} \dashrightarrow \mathscr{J}$, if for all $\mathbb{J} \in \mathscr{J}$ there is an $\mathbb{I} \in \mathscr{I}$, such that $e : \mathbb{I} \to \mathbb{J}$, for some embedding e. If we have both $\mathscr{I} \dashrightarrow \mathscr{J}$ and $\mathscr{J} \dashrightarrow \mathscr{I}$, we say that $\mathscr{I}$ and $\mathscr{J}$ are emb-equivalent, and denote it $\mathscr{I} \dashleftarrow\dashrightarrow \mathscr{J}$.

DEFINITION 8. [3]. Let I be a ground instance, and $\mathscr{J}$ a finite set of generalized instances. Then $\mathscr{J}$ is said to be $\mathtt{emb}$-universal for I and Σ if (1): $\mathscr{J} \subseteq Sol(I, \Sigma)$, (2): $\mathscr{J} \dashrightarrow Sol(I, \Sigma)$, (3): $\mathscr{J}$ is finite, and (4): there is no $\mathscr{K} \subset \mathscr{J}$, such that $\mathscr{K} \dashrightarrow \mathscr{J}$.

Deutsch et al. show that whenever the extended core chase, here denoted $Chase_\Sigma^{\mathtt{emb}}(I)$, terminates on a ground instance I it computes a finite set of generalized instances that is an $\mathtt{emb}$-universal modelset for $Sol(I, \Sigma)$.

THEOREM 8. [3] $Chase_\Sigma^{\mathtt{emb}}(I)$ *is* $\mathtt{emb}$-*universal for* $Sol(I, \Sigma)$.

Thus $Chase_\Sigma^{\mathtt{emb}}(I) \dashrightarrow Sol(I, \Sigma)$.

In order to obtain a deeper characterization of the extended core chase we relate it to the theory of incomplete information. We have

LEMMA 5. *Let $\mathscr{I}$ and $\mathscr{J}$ be finite sets of generalized instances. Then*

1. $\mathscr{I} \dashrightarrow \mathscr{J}$ *if and only if* $Rep(\mathscr{I}) \preceq Rep(\mathscr{J})$.
2. $\mathscr{I} \dashleftarrow\dashrightarrow \mathscr{J}$ *if and only if* $Rep(\mathscr{I}) \approx Rep(\mathscr{J})$.

In [3] the emb-universal modelset was used in data exchange for computing certain answers to (unions of) conjunctive queries containing negation and inequalities. As we shall see in Example 4 in the next section, the emb-universal model set is not a good choice when dealing with closed world semantics.

The following proposition gives a characterization of the result of the extended core chase. $\Sigma_{min}(.)$ denotes the set of minimal elements in $\Sigma(.)$.

PROPOSITION 3. $Rep(Chase_\Sigma^{\mathtt{emb}}(\mathbb{I})) \preceq \Sigma_{min}(Rep(\mathbb{I}))$, *and* $Rep(Chase_\Sigma^{\mathtt{emb}}(\mathbb{I})) \preceq \Sigma(Rep(\mathbb{I}))$.

5. CERTAIN ANSWERS

A data exchange setting is a scenario where the predicates are divided into two disjoint classes, namely the *source predicates*, and the *target predicates*. A tuple generating dependency where the antecedent consists only of source predicates, and the consequent only of target predicates is called a *source-to-target tgd* (abbreviated st-tdg). A *target dependency* is a tgd or egd, where all atoms are over target predicate symbols only. A *source instance* I is a ground instance over the source predicates. The st-tgd's describe how data in a source instance I is mapped to an initially empty target instance J. A generalized target instance $\mathbb{J}$ is said to be a solution to the data exchange setting if $I \cup \mathbb{J} \models \Sigma_{st}$, where Σ_{st} contains the st-tgd's. If target dependencies, usually denoted Σ_t, are present they also have to be satisfied, that is, $I \cup \mathbb{J} \models \Sigma_t$ is also required. By letting $\Sigma = \Sigma_{st} \cup \Sigma_t$, the valid target instances are simply those in $Sol(I, \Sigma)$. When issuing a query Q on the target database the certain answer is defined as [4]

$$cert(Q, I, \Sigma) = \bigcap \{Q(\mathbb{J}) : \mathbb{J} \in Sol(I, \Sigma)\},$$

where $Q(\mathbb{J})$ is defined as $\bigcap \{Q(K) : K \in Rep(\mathbb{J})\}$. This seems like a rather strange definition, but in case of monotonic queries, the only ones considered in [4], it boils down to

$$cert(Q, I, \Sigma) = \bigcap \{Q(J) : J \in Sol_{\mathtt{g}}(I, \Sigma)\},$$

that is, the ground tuples that are in the answer to the query on all ground instances in $Sol(I, \Sigma)$.

If negation is used in the queries, Deutsch et al. propose in [3] to use the emb-universal modelset for evaluating them. The following example however shows that this is does not always give intuitive answers.

EXAMPLE 4. Consider the source to target dependency $R(x) \rightarrow T(x)$ and target dependency $S(x) \rightarrow T(x)$. Let the source instance I be $\{R(a)\}$. The emb-universal model set for I and the given dependencies is $\{\{T(a)\}, \{T(a), S(a)\}\}$. Now, consider the Boolean query $\exists x : T(x) \wedge \neg S(x)$. Clearly, under the emb-model set the certain answer is **false**, whereas under any reasonable closed world semantics the certain answer should be **true**.◂

Note that $Sol(I, \Sigma)$ is closed under supersets and homomorphic images, and thus encompasses an open world assumption. As pointed out by Libkin in [10], the tacit open world assumption in [4] makes (first order) query evaluation undecidable even in the simplest data exchange settings where the target is declared to be a copy of the source. Furthermore, it does not give an intuitive semantics for negation since no negative facts can be certainly true according to $Sol(I, \Sigma)$. Thus some sort of closed world assumption is needed in data exchange. The approach proposed by Libkin is based on the intuition that any facts in the target instance should follow logically from the source instance and the dependencies, and that no two nulls in the target should be gratuitously equated. The intuition is formalized using so called *justifications*, that actually are special cases of our triggers. These justifications conveniently restrict the solutions in $Sol(I, \Sigma)$ to the universal ones, called *closed world solutions* in [10], here denoted $Sol_{\mathtt{cwa}}(I, \Sigma)$. As a matter of fact, if the data exchange setting only contains source-to-target dependencies and no target dependencies, Libkin's semantics coincides with our set of constructible solutions.

Hernich and Schweikardt [8] extend Libkin's justification based approach by allowing Σ to also contain target dependencies and they generalize the set $Sol_{\mathtt{cwa}}(I, \Sigma)$. Their set of solutions has the peculiarity that there are generalized instances $\mathbb{J} \in Sol_{\mathtt{cwa}}(I, \Sigma)$, such that $Rep(\mathbb{J}) \notin Sat(\Sigma)$. Hernich and Schweikardt define the certain answer to a target query in data exchange as

$$cert(Q, I, \Sigma) = \bigcap_{K \in Rep(\mathbb{J}) \cap Sat(\Sigma)} \bigcap_{\mathbb{J} \in Sol_{\mathtt{cwa}}(I, \Sigma)} Q(K).$$

The intuition behind this definition is not completely transparent, and it can indeed produce counter-intuitive examples, as can be seen from the following example.

EXAMPLE 5. Let $\Sigma_{st} = \{\xi_1, \xi_2\}$ and $\Sigma_t = \{\xi_3\}$, where

$$\begin{aligned} \xi_1 &= P(x) \rightarrow G(x) \\ \xi_2 &= P(x) \rightarrow \exists x_1, x_2\ E(x, x_1, x_2) \\ \xi_3 &= E(x, y, y) \rightarrow F(x) \end{aligned}$$

Let the source instance I be $\{P(a)\}$. Then, according to [8], $Sol_{\mathtt{cwa}}(I, \Sigma) = \{\{P(a), G(a), E(a, X_1, X_2)\}\}$. Consider now the FO query $\{x : G(x) \wedge \neg F(x)\}$. Under the CWA-solution assumption the certain answer to the query is $\{a\}$. Clearly this is not an intuitive answer, as there is a possible world where X_1 and X_2 map to the same constant, and $F(a)$ will also be true in that world in virtue of target dependency ξ_3. Thus $F(a)$ is not certainly false, and the answer to the query should be empty.◂

On the other hand, based on our constructible solutions, we may define

$$cert(Q, I, \Sigma) = \bigcap_{K \in \Sigma(I)} Q(K).$$

This certain answer coincides with Libkin's when there are only source-to-target dependencies, and with the standard certain answer [4] for monotone queries. Returning to the previous example, we note that $\Sigma(I)$ can be computed by applying our conditional chase on $I = \{P(a)\}$, resulting in the following conditional table.

t	$\varphi(t)$
$G(a)$	**true**
$E(a, X_1, X_2)$	**true**
$F(a)$	$X_1 = X_2$

In this case, using standard conditional evaluation [9], we obtain $cert(Q, I, \Sigma) = \varnothing$, which is the most reasonable answer. Finally, we note that the anomalies with the certain answer in [8] are noted by Hernich who in a recent paper [7] proposes yet another semantics, where disjunction is interpreted inclusively. This, however, leads to undecidable query evaluation even when considering only source to target dependencies.

6. CONCLUSIONS

In this paper we have introduced a new closed world semantics for the chase, and we have argued that it is a more intuitive representation for the set of solutions in data exchange. We have also shown that the space of constructible solutions can be strongly represented using conditional tables obtainable through our novel conditional chase process.

The full version of this paper also considers the problem of the conditional chase termination. We show that whenever Σ is a richly acyclic [8] set of dependencies, the conditional chase with Σ always terminates. The full version also contains complexity results. Mainly, we show that for richly acyclic dependencies, computing certain answers to first order queries is coNP-complete and remains polynomial when considering unions of conjunctive queries. The full version also generalizes the conditional chase to sets consisting of both tuple and equality generating dependencies.

7. REFERENCES

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

[2] A. Calì, G. Gottlob, and M. Kifer. Taming the infinite chase. In *KR*, pages 70–80, 2008.

[3] A. Deutsch, A. Nash, and J. B. Remmel. The chase revisited. In *PODS*, pages 149–158, 2008.

[4] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: Semantics and query answering. In *ICDT*, pages 207–224, 2003.

[5] R. Fagin, P. G. Kolaitis, and L. Popa. Data exchange: getting to the core. In *PODS*, pages 90–101, 2003.

[6] G. Grahne and V. Kiricenko. Towards an algebraic theory of information integration. *Inf. Comput.*, 194(2):79–100, 2004.

[7] A. Hernich. Answering non-monotonic queries in data exchange. In *ICDT*, pages 143–154, 2010.

[8] A. Hernich and N. Schweikardt. Cwa-solutions for data exchange settings with target dependencies. In *PODS*, pages 113–122, 2007.

[9] T. Imielinski and W. L. Jr. Incomplete information in relational databases. *J. ACM*, 31(4):761–791, 1984.

[10] L. Libkin. Data exchange and incomplete information. In *PODS*, pages 60–69, 2006.