Universität Rostock Traditio et Innovatio

Masterarbeit

Datenintegration durch inverse Schemaabbildungen: Erweiterung der Rostocker GaLVE-Technik

eingereicht von: Jakob Zimmer

eingereicht am: 31.08.2021

Gutachter: Prof. Dr. rer. nat. habil. Andreas Heuer
apl. Prof. Dr.-Ing. habil. Meike Klettke

Zusammenfassung

Bei einer herkömmlichen Datenintegration stehen global nur die Daten zur Verfügung, die man auch ins globale Schema abgebildet hat. Ebenso nachteilig ist, dass man für Anfragen an die globale Datenbank das neue globale Schema nutzen muss. So müssen nicht nur neue Anfragen an das neue Schema gestellt werden, sondern auch Legacy-Anwendungen umgeschrieben werden, wenn diese auch auf die Integrationsdaten in der globalen Datenbank zugreifen sollen. Diese Nachteile können mit einer Erweiterung der Quelldatenbanken der Datenintegration behoben werden. Mit dem in dieser Arbeit vorgestellten Konzept entwickeln wir ein GaLVE-Verfahren, das mit Hilfe von inversen Schemaabbildungen eine Datenintegration als Erweiterung lokaler relationaler Datenbanken umsetzt. Indem wir beliebige durch S-T TGDs formulierte Schemaabbildungen invertieren, erweitern und nachbereiten, erhalten wir eine um neue Tupel und interessante Attribute der globalen Datenbank erweiterte Quellinstanz unter einem erweiterten Quellschema. Bestehende Anfragen an dieses erweiterte Quellschema können wie gewohnt angewendet werden und haben direkten Zugriff auf die neuen Tupel. Neue Anfragen können weiterhin auf dem bekannten Quellschema entwickelt werden und dann bei Bedarf die neuen Attribute in der Erweiterung des Quellschemas nutzen. Besonders interessant an unserem GaLVE-Verfahren ist der auf der bisher erfolgten Forschung zur Invertierung von Schemaabbildungen aufbauende, einfache Invertierungsalgorithmus, der mit den *Disjunktiven-Mengen* eine abgewandelte Darstellung der disjunktiven TGDs liefert, welche gute Weiterverarbeitungsmöglichkeiten bieten. Für die Erweiterung der Quelldatenbanken und dessen Nachbereitung stellen wir mit den *bEGDs* eine ebenso interessante Möglichkeit der Bereinigung von inkonsistenten Integrationsdaten vor.

Abstract

With conventional data integration, only the data that has been mapped into the global schema is available globally. Another disadvantage is that the new global schema must be used for queries to the global database. This means that not only new queries must be directed to the new schema, but legacy applications must also be rewritten if they should also access the integration data in the global database. These disadvantages can be solved by extending the source databases of the data integration. With the concept presented in this paper, we develop a GaLVE method that implements data integration as an extension of local relational databases using inverse schema mappings. By inverting, extending and post-processing arbitrary schema mappings formulated by S-T TGDs, we obtain a source instance extended by new tuples and interesting attributes of the global database under an extended source schema. Existing queries to this extended source schema can be processed as usual and have direct access to the new tuples. New queries can still be developed on the familiar source schema and then use the new attributes in the source schema extension as needed. Of particular interest in our GaLVE method is the simple inversion algorithm, which builds on previous research on the inversion of schema mappings and uses *disjunctive sets* to provide a modified representation of disjunctive TGDs, which offer good further processing possibilities. For the extension of the source databases and its postprocessing we present with the *bEGDs* an equally interesting possibility of the cleaning of inconsistent integration data.

Inhaltsverzeichnis

1. Einleitung

Bei einer konventionellen Datenintegration werden die Schemata lokaler Quelldatenbanken in ein globales Zielschema abgebildet. Mit Hilfe solcher Schemaabbildungen stellt das globale Schema den Ausgangspunkt für Anfragen auf die integrierten Informationen aus den lokalen Datenbanken dar. Diese Art der Datenintegration wird vor allem für Data-Warehouse-Anwendungen und für die Informationsintegration genutzt. Bei beiden werden Anfragen an ein globales Schema gerichtet, welches die gesamten integrierten Informationen zu Verfügung stellt. Dieses Vorgehen ist allerdings nicht für alle Anwendungszwecke anwendbar. So können Quellen nur die Informationen durch Anfragen an das globale Schema anfragen, die sie auch in selbiges abgebildet haben. Grund für die Nicht-Einbringung der Daten in das globale Schema können z. B. Privatsphäre- und Datenschutzbestimmungen sein, aber auch zu spezialisierte, für das globale Schema nicht relevante Daten könnten ausschließlich lokal zugreifbar bleiben. Ein weiteres Problem liegt darin, dass lokale Anwendungen ihre Anfragen nun an das neue globale Schema richten müssten, um auf die integrierten Informationen zuzugreifen. So müssten entweder das lokale und das globale Schema parallel genutzt oder alle Legacy-Anwendungen unter dem neuen globalen Schema umgeschrieben werden.

Eine Lösung für diese Probleme wurde zuerst in [FHLM96] als Projektbeschreibung mit dem Titel „Föderation von Datenbanksystemen unter Beibehaltung der lokalen Anfrageautonomie und -transparenz“ vorgestellt. Die Zielsetzung des Projektes beschrieb eine Erweiterung der lokalen Schemata um „fehlende semantische Ausdrucksstärke und um Konzepte [...] die im jeweiligen lokalen Schema und dem zugrundeliegendem Datenmodell, aber auch dem Datenbanksystem fehlen“, um Anfragen an ein globales Schema über diese Erweiterungen zu ermöglichen.

Bis auf eine Erwähnung der in der Projektbeschreibung geschilderten Technik in [Con97] (als Misch-Architektur für lokale Systeme ohne wirkliche globale Anwendung), wurde das Konzept erst wieder in der Masterarbeit von Georgi Straube [Str13] als *Global as Local View Extension* (GaLVE) aufgegriffen. Die Grundidee des GaLVE-Verfahrens blieb, die Daten aus einem globalen Schema zu nutzen, um die lokalen Quelldatenbanken zu erweitern. In Abbildung 1.1 wird der grobe Ablauf des GaLVE-Verfahrens skizziert. Es ist zu erkennen, wie die Schemata *S1*, *S2* und *S3* durch entsprechende Schemaabbildungen *M1*, *M2* und *M3* auf ein globales Schema T abgebildet werden. Anschließend werden einzelne Schemaabbildungen invertiert und so Schemaabbildungen zurück auf die Quellen abgebildet. In Abbildung 1.1 werden die Daten von T mit *M1'+* zurück nach *S1+*[1] abgebildet. Das Quellschema *S1* wird so um Informationen aus dem globalen Schema erweitert. Anfragen an das ursprüngliche Schema können (anstatt sie an T anpassen zu müssen) weiterhin auf dem bekannten Schema der Quellen ausgeführt werden und gleichzeitig auf einen Großteil der integrierten Informationen aus dem globalen Schema zugreifen. In Abbildung 1.1 kann die Anfrage *q1* weiterhin über dem ursprünglichen Schema ausgeführt werden und zugleich von den Erweiterungen profitieren, während Anfragen wie *q2* und *q3* an das globale Schema angepasst werden müssen, um auf die integrierten Daten aus allen Quellen zugreifen zu können (siehe *q2+* und *q3+*).

[1] Das + steht für ein erweitertes Quellschema, welches sich grundsätzlich nur in den erweiterten Komponenten vom Schema der ursprünglichen Quelle unterscheidet (siehe Abschnitt 4.3).

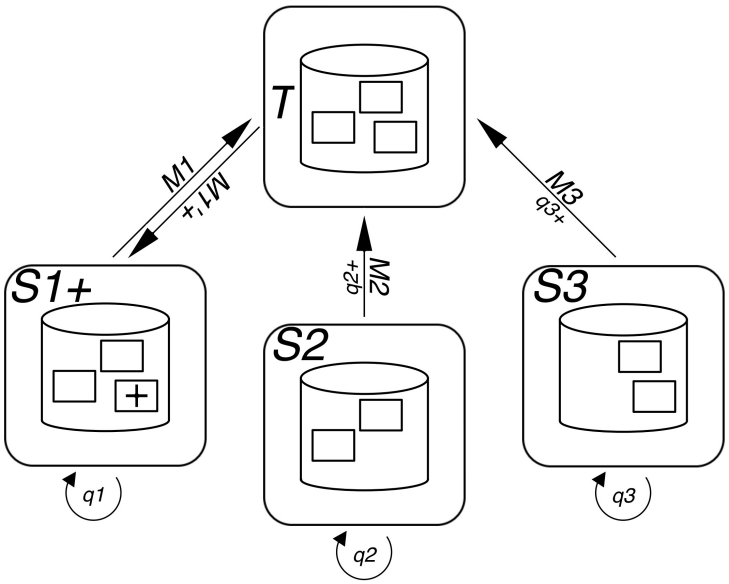

Abbildung 1.1.: Grober Ablauf des neuen GaLVE-Konzepts

In der vorherigen Arbeit von Georgi Straube [Str13] wurde die GaLVE-Technik unter dem Problem der Datenintegration mit heterogenen Quellen untersucht. Das dafür verwendete globale Schema im Entity-Attribute-Value-Modell konnte allerdings nicht zufriedenstellend mit Schemaabbildungen beschrieben werden, weshalb ein eigener Ansatz für die Abbildung in und Rückabbildung vom globalen Schema verfolgt wurde (genauer in Abschnitt 3.1). In dieser Arbeit werden wir alle Betrachtungen der Heterogenität verwerfen und nur Daten im Relationenmodell untersuchen. Das ermöglicht eine Umsetzung des GaLVE-Verfahrens mit Hilfe von Schemaabbildungen, wodurch wir eine normale Datenintegration für die Abbildung der Quellen in ein globales Schema nutzen können. Für die Rückabbildung in die Quelldatenbanken können wir so auch die theoretischen Grundlagen der Invertierung von Schemaabbildungen nutzen (siehe Abschnitt 2.3). Besonders hervorzuheben ist, dass wir Schemaabbildungen[2] ohne weitere Einschränkungen untersuchen werden, explizit auch solche, die mit Informationsverlust einhergehen.

Das Endziel dieser Arbeit ist eine allgemeine Berechnungsvorschrift für die invertierte Schemaabbildung zur Erweiterung der einzelnen Quellen sowie eine Beschreibung des gesamten neuen GaLVE-Verfahrens auf Basis des Relationenmodells. Auf dem Weg dahin werden wir unter anderem:

- Möglichkeiten finden, wie auch Schemaabbildungen mit Informationsverlust zugelassen und dennoch vollständige erweiterte Quellen gewährleistet werden können (siehe Abschnitt 4.2),
- die geeignete Theorie für die Invertierung der Schemaabbildung auf Basis von [Fag07, FKPT08, APR09, FKPT11a] auswählen und dabei Konzepte der *disjunktiven TGDs* sowie des *Disjunktiven-CHASEs* in Verbindung mit *Sicheren Antworten* untersuchen und dann für unsere Zwecke nutzen (siehe Abschnitte 2.3, 3.2 und 4.2),
- unter der Bedingung, bestehende Anfragen an die Quelldatenbank nicht umschreiben zu müssen, die Erweiterungen der Quellen um neue Tupel und Attribute beschreiben (siehe Abschnitt 4.3).

Die prototypische Implementierung des gesamten GaLVE-Verfahrens wird als Erweiterung von ChaTEAU umgesetzt. ChaTEAU ist ein an der Uni Rostock entwickeltes Java-Tool, welches den Standard-CHASE auf Instanzen und Anfragen mit (S-T) TGDs und EGDs umsetzt [AH19]. Die Hauptaufgaben werden die Invertierung der Schemaabbildungen sowie die Erweiterung des Quellschemas und die damit verbundene Erweiterung der inversen Schemaabbildung sein (siehe Kapitel 5).

[2]Durch S-T TGDs spezifizierte Schemaabbildungen (siehe Definition 2.10).

Abgrenzend bleibt zu erwähnen, dass sich diese Arbeit nicht mit weiteren Datenintegrationsproblemen befassen wird. Explizit wird das Matching, die Integration und die Abbildung der Quellschemata auf das globale Schema als gegeben angenommen. Da alle Daten im Relationenmodell vorliegen werden, entfallen Überlegungen zu heterogenen Datenstrukturen. Auch das in der Vorarbeit untersuchte Entity-Attribute-Value-Modell wird nicht weiter genutzt werden. Ebenso ist eine Behandlung der Probleme, die bei der Komposition der Anfrage, der inversen Schemaabbildung und der Schemaabbildung zur Datenintegration in das globale Schema entstehen, nicht weiter geplant. Eine genauere Betrachtung der für diese Arbeit getroffenen Annahmen und Einschränkungen folgt im Abschnitt 4.1.

1.1. Allgemeines Beispiel

Die Beispiele, die wir in dieser Arbeit verwenden werden, spiegeln vereinfachte Datenbanken für die Studierendenverwaltung an Universitäten wider. Jede Universität speichert allgemeine Informationen über ihre Studierenden, deren Teilnahme an Kursen und die Noten, die die Studierenden in den Kursen erhalten. Zusätzlich werden Sonderleistungen wie Praktika und Softskills erfasst. Für den Austausch von Studierenden wurde ein globales Datenbankschema entwickelt. Die Uni-Datenbanken sowie das globale Datenbankschema sind kein Abbild eines gelungenen Datenbank- oder Integrationsentwurfs, sondern dienen dazu, einige Probleme beim Entwurf der GaLVE-Technik zu veranschaulichen. Betrachten wir nun die Beispiel-Schemata zweier Universitäten sowie das globale Integrationsschema:

- Uni-Rostock (S_R) = {

  R-STUDIERENDE = {Matrikelnr, Nachname, Vorname, Studiengang, Bemerkung},

  R-TEILNAHME = {Teilnahmenr, Modulnr, Matrikelnr},

  R-NOTEN = {Teilnahmenr, Semester, Note},

  R-ZUSATZ = {Matrikelnr, Leistung}},

- Uni-Kiel (S_K) = {

  K-STUDIERENDE = {Matrikelnr, Nachname, Vorname, Abschluss},

  K-NOTEN = {Modulnr, Matrikelnr, Semester, Note},

  K-PRAKTIKUM = {Matrikelnr},

  K-SOFTSKILL = {Matrikelnr}},

- Globales Schema (T) = {

  T-STUDIERENDE = {Matrikelnr, Nachname, Vorname, Studiengang, Abschluss},

  T-NOTEN = {Modulnr, Matrikelnr, Semester, Note},

  T-ZUSATZ = {Matrikelnr}}.

Die Relationen wurden bereits so benannt, dass ihre Bezeichner disjunkt innerhalb aller Datenbanken sind. Dieser Schritt ist für die Anwendung der Theorie von Schemaabbildungen zwischen einer Quelle und einem Ziel notwendig (siehe Definition 2.10) und erfolgt normalerweise in der Vorbereitung der Datenintegration. Wie bereits in der Einleitung erwähnt, wird sich diese Arbeit nicht mit den typischen

Datenintegrationsproblemen befassen. Aus diesem Grund gelten auch alle Schlüssel[3] als globale Identifikatoren. Alle gleich benannten Attribute gehören außerdem zu den gleichen Wertebereichen, welche wie folgt definiert sind:

- dom(Matrikelnr) := $\mathbb{N}$ = {1, 2, 3, ...}
- dom(Nachname) := STRING = {Fieber, Sonne, Mueller,...[4]}
- dom(Vorname) := STRING = {Fabian, Sarah, Max,...[5]}
- dom(Studiengang) := {Informatik, ITTI}
- dom(Abschluss) := {Bachelor, Master}
- dom(Bemerkung) := STRING = {Tel:***, IBAN:***,...[6]}
- dom(Teilnahmenr) := $\mathbb{N}$
- dom(Modulnr) := $\mathbb{N}$
- dom(Semester) := {WS 20/21, SS 21[7]}
- dom(Note) := {1.0, 1.3, 1.7, 2.0, 2.3, 2.7, 3.0, 3.3, 3.7, 4.0, 5.0}
- dom(Leistung) := {Praktikum, Softskill}

Ein erster erkennbarer, für das Konzept-Kapitel interessanter Unterschied zwischen den Datenbankschemata ist z. B. die Abwesenheit des Studiengang-Attributs in der K-STUDIERENDE-Relation. Auch die Vereinigung der K-PRAKTIKUM- und K-SOFTSKILL-Relationen in T-ZUSATZ wird eine Rolle für die Invertierung von möglichen Schemaabbildungen spielen.

Abschließend werden wir noch kleine Beispiel-Instanzen angeben, die die lokalen Uni-Datenbanken sowie die materialisierte globale Datenbank nach der Integration zeigen. Mögliche Abbildungsvorschriften, die die Datenintegration von den Uni-Datenbanken in die globale Datenbank beschreiben, werden im 2. Kapitel nach der Definition 2.10 von Schemaabbildungen gegeben. In diesem Kapitel werden die Instanzen der Übersichtlichkeit halber als Tabellen 1.1 bis 1.11 dargestellt. Diese sind nur Beispieldatensätze, die genauen Attributwerte der Instanzen werden jeweils vor dem Gebrauch definiert. In den nachfolgenden Kapiteln werden wir Instanzen als Fakten (siehe Definition 2.6) darstellen, anstatt sie in der Tabellendarstellung anzugeben. Dafür werden wir Ausdrücke der Form $R_i(w_1, ..., w_p)$ mit R_i als Relationennamen und $w_q \in \{w_1, ..., w_p\}$ als Attribut- oder Nullwerte eines Tupels verwenden. Die Tabelle 1.5 würde als Instanz I somit wie folgt definiert werden:

$$I = \{\text{K-STUDIERENDE}(2, \text{Sonne}, \text{Sarah}, \text{Bachelor}),\\ \text{K-STUDIERENDE}(3, \text{Mueller}, \text{Max}, \text{Master})\}.$$

[3] Unterstrichene Attribute in der Schemabeschreibung
[4] Weitere Nachnamen vom Datentyp STRING
[5] Weitere Vornamen vom Datentyp STRING
[6] U. A. sensible oder persönliche Informationen vom Datentyp STRING wie Telefonnummern oder Erkrankungen
[7] Eingeschränkt auf die relevanten Attributwerte

Matrikelnr	Nachname	Vorname	Studiengang	Bemerkung
1	Fieber	Fabian	Informatik	Tel: 0123456789
2	Sonne	Sarah	ITTI	Corona(+) am 07.12.2020

Tabelle 1.1.: R-STUDIERENDE

Teilnahmenr	Modulnr	Matrikelnr
1	1	1
2	2	1
3	1	2

Tabelle 1.2.: R-TEILNAHME

Teilnahmenr	Semester	Note
1	WS 20/21	1.7
2	SS 21	3.0
3	WS 20/21	2.3

Tabelle 1.3.: R-NOTEN

Matrikelnr	Leistung
1	Praktikum
1	Softskill
2	Praktikum

Tabelle 1.4.: R-ZUSATZ

Matrikelnr	Nachname	Vorname	Abschluss
2	Sonne	Sarah	Bachelor
3	Mueller	Max	Master

Tabelle 1.5.: K-STUDIERENDE

Modulnr	Matrikelnr	Semester	Note
1	3	WS 20/21	2.0
2	2	SS 21	1.3

Tabelle 1.6.: K-NOTEN

Matrikelnr
3

Tabelle 1.7.: K-PRAKTIKUM

Matrikelnr
2
3

Tabelle 1.8.: K-SOFTSKILL

Matrikelnr	Nachname	Vorname	Studiengang	Abschluss
1	Fieber	Fabian	Informatik	
2	Sonne	Sarah	ITTI	Bachelor
3	Mueller	Max		Master

Tabelle 1.9.: T-STUDIERENDE

Modulnr	Matrikelnr	Semester	Note
1	1	WS 20/21	1.7
2	1	SS 21	3.0
1	2	WS 20/21	2.3
1	3	WS 20/21	2.0
2	2	SS 21	1.3

Tabelle 1.10.: T-NOTEN

Matrikelnr
1
2
3

Tabelle 1.11.: T-ZUSATZ

1.2. Aufbau der Arbeit

Um das Konzept dieser Arbeit nachvollziehbar aufzubereiten, werden wir im 2. Kapitel kurz die Grundlagen des Relationenmodells erklären. Auch die Grundlagen der Datenintegration, der Schemaabbildungen sowie des CHASEs als Basis-Algorithmus für die Schemaabbildung werden wir hier erläutern.

Das 3. Kapitel gibt einen kleinen Überblick über den Stand der Forschung. Den Schwerpunkt legen wir auf die Analyse der Forschungsergebnisse zur Invertierung von Schemaabbildungen in [Fag07, FKPT08, APR09, FKPT11a]. Diese werden wir untereinander vergleichen, um ein geeignetes Invertierungsverfahren für die GaLVE-Technik auszuwählen.

Im Kapitel 3 untersuchen wir ebenfalls die Herangehensweise der Vorarbeit von Georgi Straube [Str13] sowie die Arbeit [AH18b] von Tanja Auge und Andreas Heuer. Letztere zeigt einen weiteren interessanten Blickwinkel auf die Theorie der Invertierung von Schemaabbildungen, indem sie diese mit dem Provenance-Management verbindet, welches sich mit der Herkunft von Anfrageergebnissen beschäftigt.

Im Kapitel 4 stellen wir dann das neue Konzept des GaLVE-Verfahrens auf Basis des Relationenmodells vor. Dieses neue GaLVE-Verfahren bildet den Kern dieser Arbeit. Hauptziel wird dabei die Berechnung der inversen Schemaabbildung sowie die Erweiterung des Quellschemas in Zusammenhang mit der Erweiterung der inversen Schemaabbildung sein. Außerdem klären wir, wie in dem vorgestellten Verfahren das Problem der inkonsistenten Daten bei der Integration aus einem globalen Schema gelöst werden kann.

Im darauffolgenden 5. Kapitel zeigen wir, wie die Implementierung der einzelnen Techniken des GaLVE-Verfahrens umgesetzt wurde. Dazu beschreiben wir ebenfalls, wie diese Funktionalität in ChaTEAU eingearbeitet wurde und wie das gesamte GaLVE-Verfahren in ChaTEAU ablaufen könnte.

Das Kapitel 6 zum Fazit und Ausblick wartet neben einer Zusammenfassung dieser Arbeit auch noch mit einigen offenen Problemen und einem Ausblick auf, welcher den Ansatz für künftige Aufgaben bilden könnte. Im Anhang befinden sich neben dem Literaturverzeichnis zusätzlich noch einige laufende Beispiele von ChaTEAUs GaLVE-Erweiterung und der gesamte Programmcode der Implementierung.

2. Grundlagen

Der Abschnitt 2.1 dieses Kapitels definiert kurz die Konzepte des Relationenmodells nach [HSS18]. Die Umstellung des GaLVE-Verfahrens auf die ausschließliche Nutzung des Relationenmodells ermöglicht erst die Anwendung aller folgenden Theorien. Im Abschnitt 2.2 werden wir kurz auf den Ablauf und die damit verbundenen Probleme der Datenintegration auf Basis von [DHI12, ÖV11] eingehen. Das GaLVE-Verfahren wird an die Datenintegration anknüpfen. Die vorgestellten Probleme werden daher nicht Bestandteil unseres vorgestellten Konzepts sein und nur die Aufgaben zeigen, die vor dem GaLVE-Verfahren, wie es in Kapitel 4 vorgestellt wird, gelöst werden müssen. Die Theorien rund um Schemaabbildungen werden wir im Abschnitt 2.3 definieren. Anschließend wird der CHASE-Algorithmus eingeführt (siehe Abschnitt 2.4). Dieser wird uns vor allem als Tool für die Berechnung des Ergebnisses der Schemaabbildung dienen. Alle Definitionen in diesem Kapitel werden anhand des laufenden allgemeinen Beispiels aus Abschnitt 1.1 anschaulich erklärt. Umgeformte Definitionen kennzeichnen wir mit dem Stichwort „nach“. Zum Abschluss (Abschnitt 2.5) geben wir eine Einordnung der Grundlagen in den weiteren Kontext dieser Arbeit.

2.1. Relationenmodell

Das Relationenmodell beschreibt die Daten einer Datenbank mit Hilfe eines Datenbankschemas, welches wiederum aus mehreren Relationenschemata zusammengesetzt wird. Diese Relationenschemata bestehen aus Attributen und beschreiben die Relationen. In ihnen liegen die eigentlichen Daten als sogenannte Tupel, die indes aus den einzelnen Werten bestehen. Im Folgenden werden wir die angesprochenen Teile des Relationenmodells formal definieren.

Definition 2.1. (Attribute und Wertebereiche, [HSS18]): Sei $\mathcal{U}$ eine nicht-leere, endliche Menge, genannt *Universum*. Ein $\mathcal{A} \in \mathcal{U}$ heißt *Attribut*. Sei $\mathcal{D} = \{D_1, ..., D_m\}$ die nicht-leere, endliche Menge der *Wertebereiche* D_i mit $m \in \mathbb{N}$, so existiert eine total definierte Funktion dom: $\mathcal{U} \longrightarrow \mathcal{D}$. Den Funktionswert von dom(A) nennen wir *Wertebereich* von A.

Definition 2.2. (Benannte Nullwerte): Sei $\mathcal{D} = \{D_1, ..., D_m\}$ die nicht-leere, endliche Menge der Wertebereiche D_i, so ist $\mathcal{N} = \{\eta_1, ..., \eta_n\}$ eine abzählbar unendliche Menge *benannter Nullwerte*, wobei $\forall i \in \{1, ..., m\}$, $\forall j \in \{1, ..., n\} : \eta_j \notin D_i$.

Definition 2.3. (Relationenschema und Relation, nach [HSS18]): Ein *Relationenschema* ist eine Menge $R \subseteq \mathcal{U}$. Für $R = \{A_1, ..., A_n\}$ mit $n \in \mathbb{N}$, ist die *Relation r über R* (geschrieben: *r(R)*) als eine endliche Menge von Abbildungen

$$t : R \longrightarrow \bigcup_{i=1}^{m} D_i \cup \mathcal{N}$$

definiert, die *Tupel* genannt werden und für die $t(a) \in dom(A) \cup \mathcal{N}$ gilt. $t(A)$ schränkt dabei die Abbildung t auf $A \in R$ ein.

Definition 2.4. (Datenbankschema, [HSS18]): Ein *Datenbankschema* ist eine Menge von Relationenschemata $S := \{R_1, ..., R_p\}$ mit $p \in \mathbb{N}$.

Definition 2.5. (Instanz, [HSS18]): Für ein Datenbankschema S mit Relationenschemata $R_1, ..., R_p$, nennen wir die Menge von Relationen $d := \{r_1, ..., r_p\}$ *Instanz* der Datenbank mit $r_i(R_i)$ $\forall i \in \{1, ..., p\}$.

Eine Instanz repräsentiert die aktuell vorhandenen Daten einer Datenbank. Äquivalent zur Instanz einer Datenbank werden wir eine Relation als Instanz zu einem Relationenschema bezeichnen.

Für unser allgemeines Beispiel (siehe Abschnitt 1.1) wären die Datensätze der Tabelle 1.1 eine Instanz zum Relationenschema R-STUDIERENDE = {Matrikelnr, Nachname, Vorname, Studiengang, Bemerkung}. Die einzelnen Tupel der R-STUDIERENDE-Relation entsprechen den beiden Zeilen unter dem Relationenschema im Tabellenkopf. Wenn wir später von unseren Beispielinstanzen sprechen, dann werden immer (soweit nicht anders angegeben) die Datenbankinstanzen, bestehend aus den Datensätzen der Relationen in den Tabellen 1.1 bis 1.11 unter den Datenbankschemata S_R={R-STUDIERENDE, R-TEILNAHME, R-NOTEN, R-ZUSATZ}, S_K={K-STUDIERENDE, K-NOTEN, K-PRAKTIKUM, K-SOFTSKILL} und T={T-STUDIERENDE, T-NOTEN, T-ZUSATZ} gemeint sein. Die Instanzen werden wir dann nicht mehr mit Tabellen sondern als Menge ihrer *Fakten* beschreiben.

Definition 2.6. (Fakt einer Instanz, nach [FKPT11a]): Sei ein Schema S geben. Sein außerdem $r(R)$ eine Relation über $R = \{A_1, ..., A_n\}$ mit $R \in S$. Ein atomarer Ausdruck der Form $R(a_1, ...a_n)$ heißt *Fakt* der Instanz über S, wenn das Tupel $(a_1, ...a_n) \in r(R)$ ist.

Die Schreibweise einer Instanz als Menge von atomaren Ausdrücken erlaubt uns später etwa die Anwendung der Schemaabbildungen per CHASE-Algorithmus (siehe Abschnitt 2.4.2). Die Instanz I_{S_K} über dem Schema S_K würde wie folgt definiert werden:

$$
\begin{aligned}
I_{S_K} = \{&\text{K-STUDIERENDE}(2, \text{Sonne}, \text{Sarah}, \text{Bachelor}),\\
&\text{K-STUDIERENDE}(3, \text{Mueller}, \text{Max}, \text{Master}),\\
&\text{K-NOTEN}(1, 3, \text{WS } 20/21, 2.0),\\
&\text{K-NOTEN}(2, 2, \text{SS } 21, 1.3),\\
&\text{K-PRAKTIKUM}(3),\\
&\text{K-SOFTSKILL}(2),\\
&\text{K-SOFTSKILL}(3)\}.
\end{aligned}
$$

Im nächsten Abschnitt erklären wir kurz das Vorgehen und die Probleme der Datenintegration. Das neue GaLVE-Verfahren (siehe Konzept-Kapitel) wird später an die Datenintegration anknüpfen.

2.2. Datenintegration

Die Informationen aus diesem sehr kompakten Abschnitt stammen aus [DHI12, ÖV11]. Da die angesprochenen Problemen und deren Lösungen nicht direkter Bestandteil des in dieser Arbeit vorgestellten Konzepts sein werden, wird für genauere Informationen auf die Literatur verwiesen.

Grob zusammengefasst befasst sich die Datenintegration mit der Zusammenführung und Zur-Verfügung-Stellung von Daten, die in autonomen und möglicherweise heterogenen Datenbanken liegen. Wird die Datenintegration schon bei der Entwicklung eines verteilten Datenbanksystems beachtet, können die Schemata der lokalen Datenbanken aus einem vorher spezifizierten globalen Schema abgeleitet werden. So können die meisten Probleme der Datenintegration wie strukturelle Konflikte oder heterogene Schemata per Design ausgeschlossen werden. Dieser Top-down-Ansatz ist allerdings nur für stark integrierte verteilte Datenbanksysteme anwendbar. Sobald die Quelldatenbanken bereits existieren (wie in den von uns betrachteten Fällen) und im Nachhinein in ein globales Schema integriert werden sollen, spricht man von einem bottom-up-Ansatz. Für diesen kann man wiederum unterscheiden, ob die Daten im globalen Schema *materialisiert* oder nur *virtuell* vorliegen sollen.

Ein typisches Beispiel für die *materialisierte Datenintegration* sind Data-Warehouse-Anwendungen. Auf der materialisierten globalen Datenbank (die Data Warehouse genannt wird) können komplexe Analyseprozesse ausgeführt werden, während die Quellen weiterhin für zeitlich relevante Aufgaben bereitstehen. Die eigentliche Integration der Daten erfolgt hier im sogenannten *ETL-Prozess* (**E**xtract, **T**ransform, **L**oad). Der erste Schritt extrahiert die Daten aus den Quelldatenbanken, um sie dann im zweiten Schritt in die Struktur des globalen Schemas passend zu transformieren, wonach sie schließlich in die globale Datenbank geladen werden können.
Im Falle einer *virtuellen Datenintegration* verschiebt sich die Komplexität der Datenbereitstellung auf den Anfrageprozess. Es müssen allerdings nur die Daten aus den Quellen geladen werden, die auch wirklich vom globalen Schema angefragt werden. Das führt zu einem geringeren Speicherverbrauch (die Daten liegen nur in den Quellen), aber auch zu einer rechenintensiven Umformung und Bearbeitung der Anfrage bei deren Ausführung.
Die Wahl zwischen materialisierter und virtueller Datenintegration wird sich für die GaLVE-Technik gleich doppelt stellen. Zum einen kann das globale Schema materialisiert oder nur virtuell erstellt werden, zum anderen kann auch die Erweiterung der lokalen Schemata in beiden Varianten umgesetzt werden. Die sich daraus ergebenden Varianten des GaLVE-Verfahrens, werden wir im Abschnitt 4.4.2 vorstellen.

Der von uns betrachtete Bottom-up-Ansatz unterteilt sich auch noch dahingehend, ob das globale Schema vor der Datenintegration gegeben ist oder im Zusammenhang mit der Datenintegration erstellt wird. Ist das globale Schema gegeben, müssen nur noch Abbildungen von den lokalen Schemata auf das globale Schema gefunden werden. Diese Abbildungen können wiederum in *Local as View* (LaV) und *Global as View* (GaV) unterteilt werden (siehe formale Definition 2.11). LaV-Abbildungen definieren die lokalen Schemata über Sichten auf dem globalen Schema. Die Anfragen an das globale Schema können also nur soviel Informationen liefern, wie aus den einzelnen Quelldatenbanken bereitgestellt wird. GaV-Abbildungen beschreiben genau entgegengesetzt das globale Schema über einer Menge von Sichten auf den lokalen Schemata. Anfragen können hier auf die kombinierten Informationen der Quelldatenbanken zugreifen, dafür aber nicht auf die Informationen, die nur in einzelnen Quellen vorhanden ist. *Global Local as View*-Abbildungen (GLaV) umgehen die Einschränkungen von LaV- und GaV-Abbildungen, indem sie sowohl die lokalen Schemata wie auch das globale Schema über Sichten darstellen.
Der beschriebene Bottom-up-Ansatz wird zusammen mit einem gegebenen globalen Schema, LaV-, GaV- und GLaV-Abbildungen Ausgangspunkt für das Konzept-Kapitel sein. Im weiteren Teil dieses Abschnitts werden wir den Ablauf des Bottom-up-Ansatzes noch etwas genauer beschreiben.

Egal ob die Generierung des globalen Schemas noch erfolgen muss oder dieses schon vordefiniert ist, läuft der Bottom-up-Ansatz der Datenintegration in zwei Schritten ab. Zuerst müssen die lokalen Schemata in ein einheitliches Datenbankmodell übersetzt werden. Da wir in dieser Arbeit keine heterogenen Datenbanken betrachten (alle Quellen liegen im Relationenmodell vor) entfällt dieser Schritt für uns. Der zweite Schritt generiert gegebenenfalls das globale Schema und die Abbildungen von den lokalen Schemata auf das globale Schema. Dies erfolgt grundsätzlich in drei Schritten (auch *MIA-Prozess* genannt):

- ***M**atching* der Elemente der lokalen Schemata untereinander (bei gegebenem globalen Schema, Matching der lokalen Schemata und des globalen Schemas).
- ***I**ntegration* der gemeinsamen Elemente der lokalen Schemata in ein globales Schema (wenn dieses noch nicht existiert).
- ***A**bbildung* der Elemente der einzelnen lokalen Schemata auf die Elemente des globalen Schemas.

Matching Beim Schema-Matching wird nach Übereinstimmungen in den Elementen der einzelnen lokalen Schemata gesucht. Sollte das globale Schema schon gegeben sein, werden stattdessen Übereinstimmungen zwischen den einzelnen lokalen Schemata und dem globalen Schema gesucht.
Die größte Herausforderung beim Schema-Matching stellt die Schema-Heterogenität zwischen den lokalen Schemata dar. So werden gleiche Dinge oft unterschiedlich in den Schemata dargestellt, aber auch im Gegenteil unterschiedliche Dinge gleich erfasst. Diese strukturellen Konflikte können unter anderem in den Datentypen, den Abhängigkeiten, den Schlüsseln oder dem Verhalten der Datenbank vorliegen. Unabhängig von der Schema-Heterogenität erschweren auch fehlende und unvollständige Informationen über die Schemata und die Instanzen das Schema-Matching. Darunter fällt z. B. die Verwendung von Abkürzungen oder kryptischen Benennungen der Elemente einer Datenbank. Diese können manchmal weder von Algorithmen noch durch die Datenbankadministratoren korrekt aufgelöst werden.
Lösungsansätze für diese Probleme gibt es einige. So suchen schema-basierte Matcher Übereinstimmungen auf elementarer oder struktureller Schemaebene. Instanz-basierte Matcher untersuchen auf elementarer Instanzebene in den Daten nach Übereinstimmungen. Für beide Techniken gibt es verschiedene linguistische -, abhängigkeits-basierte und lernende Ansätze.

Integration Falls ein globales Schema schon während des Schema-Matchings gegeben war, wurden dort bereits Übereinstimmungen zwischen ihm und den lokalen Schemata gefunden. Der Schritt der Schema-Integration ist somit nicht mehr nötig. Falls das globale Schema hingegen noch nicht existierte, wird es nun aus den Übereinstimmungen der lokalen Schemata generiert. Zusätzlich werden die Übereinstimmungen zwischen den lokalen Schemata und dem neu erstellten globalen Schema ermittelt.

Abbildung Die gefundenen Übereinstimmungen werden im Schema-Mapping-Schritt genutzt, um Schemaabbildungen von den lokalen Schemata in das globale Schema zu generieren. Mit diesen Schemaabbildungen können die Daten aus den lokalen Datenbanken in das globale Schema überführt werden. Wie bereits erwähnt, werden die Schemaabbildungen im Falle einer materialisierten Datenintegration direkt angewendet. Bei der Ausführung werden die Daten aus den Quellen extrahiert, passend umgeformt und in die globale Datenbank geladen. Bei einer virtueller Datenintegration werden die Schemaabbildungen erst beim Anfrageprozess verwendet, um die angefragten Daten aus den Quellen zu laden.

Data Cleaning Anschließend an den MIA-Prozess kann es notwendig sein, die integrierten Daten auf Inkonsistenzen zu überprüfen. So können Daten aus verschiedenen konsistenten und fehlerfreien Quellen durchaus inkonsistent oder fehlerhaft im globalen Schema sein. Das *Data Cleaning* ist dabei sowohl bei der materialisierten als auch bei der virtuellen Datenintegration notwendig. Der materialisierte Fall führt das Data Cleaning bereits bei der Erstellung des globalen Schemas aus, während es bei der virtuellen Datenintegration binnen des Anfrageprozesses abläuft. Data Cleaning behandelt sowohl Fehler auf der Schema- als auch auf der Instanzebene. Auf Schemaebene können z. B. Einschränkungen der Attributwerte verletzt werden. Auf Instanzebene treten dagegen typische Inkonsistenzen zwischen den Daten auf.

Im folgenden Abschnitt widmen wir uns den Grundlagen der Schemaabbildungen. Dort werden wir auch eine Beispiel-Schemaabbildung für unser allgemeines Beispiel (siehe Abschnitt 1.1) entwerfen.

2.3. Schemaabbildung

Die Abbildungen zwischen den Quell-Datenbanken und dem globalen Schema werden wir mit Hilfe von *source-to-target tuple-generating dependencies* (kurz: S-T TGD) beschreiben. Diese sind eine besondere Form von *tuple-generating dependencies* (kurz: TGD), welche wir neben den *equality-generating dependencies* (kurz: EGD) im Weiteren zusammengefasst als *Integritätsbedingungen* bezeichnen. Allgemein sind Integritätsbedingungen deskriptive Bedingungen an einen Datenbestand, die die möglichen Zustände einer Datenbank limitieren, um eine gewisse Konsistenz zu gewährleisten. Bekanntere Integritätsbedingungen sind z. B. *Schlüssel*, die für die Tupel einer Relation eine identifizierende Attributmenge bilden oder die etwas allgemeineren *funktionalen Abhängigkeiten*, bei denen eine bestimmte Attributmenge in einer Relation den Wert einer anderen Attributmenge bestimmt. Beide sind Spezialformen von EGDs. Bekanntere Spezialisierungen von TGDs sind *Verbund-Abhängigkeiten*, welche garantieren, dass die Zerlegung einer Relation beim Verbund wieder die Originalrelation selbst ergibt (Verbundtreue).

Definition 2.7. (S-T TGD, [DHI12, AH18a]): *Tuple-generating dependencies* sind prädikatenlogische Ausdrücke erster Ordnung der Form

$$\forall \mathbf{x} : (\phi(\mathbf{x}) \longrightarrow \exists \mathbf{y} : \psi(\mathbf{x}, \mathbf{y})),$$

wobei $\phi(\mathbf{x})$ und $\psi(\mathbf{x}, \mathbf{y})$ Konjunktionen von atomaren Ausdrücken der Form $R_i(a_1, ..., a_p)$ sind und jedes a_j (mit $j \in \{1, ..., p\}$) für $\phi(\mathbf{x})$ eine allquantifizierte Variable aus $\mathbf{x}$ oder eine Konstante bzw. für $\psi(\mathbf{x}, \mathbf{y})$ eine allquantifizierte Variable aus $\mathbf{x}$, eine existenzquantifizierte Variable aus $\mathbf{y}$ oder eine Konstante ist. Der Bezeichner R_i steht für den Bezeichner eines Relationenschemas. Jede Variable aus $\mathbf{x}$ kommt in $\phi(\mathbf{x})$, aber nicht notwendiger Weise in $\psi(\mathbf{x}, \mathbf{y})$ vor. Eine TGD, bei der die Ausdrücke in $\phi(\mathbf{x})$ Relationenschemata eines Quellschemas S und die Ausdrücke in $\psi(\mathbf{x}, \mathbf{y})$ Relationenschemata eines Zielschemas T repräsentieren, nennt man auch *source-to-target tuple-generating dependency*.

Definition 2.8. (EGD, [DHI12, AH18a]): *Equality-generating dependencies* sind prädikatenlogische Ausdrücke erster Ordnung der Form

$$\forall \mathbf{x} : (\phi(\mathbf{x}) \longrightarrow (x_1 = x_2), ..., (x_n = x_m)),$$

wobei $\phi(\mathbf{x})$ eine Konjunktion von atomaren Ausdrücken der Form $R_i(a_1, ..., a_p)$ ist und jedes a_j (mit $j \in \{1, ..., p\}$) eine allquantifizierte Variable aus $\mathbf{x}$ oder eine Konstante ist. Der Bezeichner R_i steht wie bei den TGDs für den Bezeichner eines Relationenschemas. Alle Variablen in den Gleichheitsatomen auf der rechten Seite kommen auch in $\phi(\mathbf{x})$ vor.

Wir werden $\phi(\mathbf{x})$ als den *Rumpf* einer TGD bzw. EGD bezeichnen und $\psi(\mathbf{x}, \mathbf{y})$ in TGDs bzw. die Gleichheitsatome der EGDs als den *Kopf* einer solchen. TGDs haben nur im Kopf existenzquantifizierte Variablen, EGDs gar keine. Der Einfachheit halber werden wir im Weiteren die Allquantifizierung von $\mathbf{x}$ vor den TGDs und EGDs weglassen.

Eine mögliche Integritätsbedingung für unser allgemeines Beispiel (siehe Abschnitt 1.1) wäre z. B. die Forderung, dass alle Studierende aus R-STUDIERENDE, die eine Note in R-NOTEN haben, auch als Teilnehmer des entsprechenden Moduls in R-TEILNAHME auftauchen müssen. Aus den Relationen R-STUDIERENDE und R-NOTEN wollen wir also Tupel für die R-TEILNAHME-Relation generieren. So formuliert erkennt man, dass diese Forderung leicht mit einer TGD dargestellt werden kann, bei der R-STUDIERENDE und R-NOTEN im TGD-Rumpf auftauchen müssen und R-TEILNAHME in den TGD-Kopf gehört:

$$\text{R-STUDIERENDE}(x_{\text{Ma}}, x_{\text{Na}}, x_{\text{Vo}}, x_{\text{St}}, x_{\text{Be}}) \wedge \text{R-NOTEN}(x_{\text{Te}}, x_{\text{Se}}, x_{\text{No}})$$
$$\longrightarrow \exists y_{\text{Mo}} : \text{R-TEILNAHME}(x_{\text{Te}}, y_{\text{Mo}}, x_{\text{Ma}}).$$

Da das Attribut für die Modulnummer nicht in R-STUDIERENDE oder R-NOTEN vorkommt, wird es im R-TEILNAHME-Ausdruck im TGD-Kopf als existenzquantifizierte y-Variable geschrieben.

Eine Beispiel-EGD könnte lauten: Wenn in R-STUDIERENDE Tupel für Studierende mit der gleichen Matrikelnummer vorkommen, dann müssen auch deren Namen übereinstimmen:

$$\text{R-STUDIERENDE}(x_{\text{Ma}}, x_{\text{Na}_1}, x_{\text{Vo}_1}, x_{\text{St}_1}, x_{\text{Be}_1}) \wedge \text{R-STUDIERENDE}(x_{\text{Ma}}, x_{\text{Na}_2}, x_{\text{Vo}_2}, x_{\text{St}_2}, x_{\text{Be}_2})$$
$$\longrightarrow (x_{\text{Na}_1} = x_{\text{Na}_2}), (x_{\text{Vo}_1} = x_{\text{Vo}_2}).$$

Die Bedingung der gleichen Matrikelnummern wird über die Gleichbenennung der Variablen für die Matrikelnummer (x_{Ma}) im EGD-Rumpf angegeben. Die daraus folgende Gleichheit des Namens steht in den Gleichheitsatomen im Kopf der EGD. Die Schlüsseleigenschaft der Matrikelnummer könnte mit einer ähnlichen EGD ausgedrückt werden, bei der auch eine Gleichheit des Studiengangs und der Bemerkung gefordert wird. Beispiele für S-T TGDs folgen der Definition 2.10.

Definition 2.9. (Homomorphismus, nach [GMS12]): Seien I_1 und I_2 zwei Instanzen über demselben Schema mit Konstanten und Nullwerten in den Tupeln der Relationen. Ein *Homomorphismus* $h : I_1 \longrightarrow I_2$ ist eine Abbildung von Konstanten und Nullwerten auf Konstanten und Nullwerten unter folgenden Regeln:

(a1) Eine Konstante c kann nur auf sich selbst abgebildet werden: $h(c) = c$.

(a2) Ein Nullwert η_i kann auf eine Konstante c, sich selbst oder einen anderen Nullwert η_j mit $i \neq j$ abgebildet werden: $h(\eta_i) = [c|\eta_i|\eta_j]$.

Dabei gilt: Für jedes Tupel $t = (a_1, ..., a_n)$ aus den Relationen in I_1 existiert ein Tupel $h(t)$ in den Relationen aus I_2, mit $h(t) = (h(a_1), ..., h(a_n))$.

Die Instanzen I_1 und I_2 gelten genau dann als *homomorph äquivalent* (geschrieben: $I_1 \longleftrightarrow I_2$), wenn es einen Homomorphismus von I_1 nach I_2 ($h_1 : I_1 \longrightarrow I_2$) und einen Homomorphismus von I_2 nach I_1 ($h_2 : I_2 \longrightarrow I_1$) gibt.

Seien zwei Beispielinstanzen I_1 und I_2 wie folgt gegeben:

$$I_1 = \{\text{R-STUDIERENDE}(1,\ \eta_{\text{Na}}\ , \text{Fabian}, \text{Informatik},\ \eta_{\text{Be}_1})\},$$
$$I_2 = \{\text{R-STUDIERENDE}(1, \text{Fieber}, \text{Fabian}, \text{Informatik},\ \eta_{\text{Be}_2})\}.$$

Instanz I_1 enthält anstatt konkreter Attributwerte für den Nachnamen und die Bemerkung die Nullwerte η_{Na} und η_{Be_1}. Die Instanz I_2 enthält einen Nullwert η_{Be_2} für die Bemerkung. Es kann ein Homomorphismus von I_1 nach I_2 ($h : I_1 \longrightarrow I_2$) definiert werden, der η_{Na} auf die Konstante „Fieber“ ($h(\eta_{\text{Na}}) = \text{Fieber}$), den Nullwert η_{Be_1} auf den Nullwert η_{Be_2} ($h(\eta_{\text{Be}_1}) = \eta_{\text{Be}_2}$) und die anderen Attributwerte auf sich selbst ($h(1) = 1$, $h(\text{Fabian}) = \text{Fabian}$, $h(\text{Informatik}) = \text{Informatik}$) abbildet.

Definition 2.10. (Schemaabbildung, Lösung, universelle Lösung, nach [FKPT11a]): Sei S ein Quellschema und T ein Zielschema mit disjunkten Relationenschemata. Sei Σ eine Menge von Integritätsbedingungen, die die Beziehung zwischen S und T beschreibt (typischerweise S-T TGDs). Das Triple

$\mathcal{M} = (S, T, \Sigma)$ ist eine *syntaktische Sicht* auf eine *Schemaabbildung* von S nach T charakterisiert durch Σ. Eine *semantische Sicht* auf eine *Schemaabbildung* wird über die binäre Relation

$$\{(I, J) \mid I \text{ ist Instanz über } S, J \text{ ist Instanz über } T, (I, J) \models \Sigma\}$$

definiert. Wir nennen J *Lösung* für I unter $\mathcal{M}$, wenn $(I, J) \in \mathcal{M}$. Die Menge aller Lösungen für I unter $\mathcal{M}$ bezeichnen wir mit $Sol_{\mathcal{M}}(I)$. Eine Lösung $J \in Sol_{\mathcal{M}}(I)$ heißt eine *universelle Lösung* für I unter $\mathcal{M}$, wenn $J \longrightarrow J'$ für alle $J' \in Sol_{\mathcal{M}}(I)$ gilt.

Einen Spezialfall von Schemaabbildungen stellen Anfragen dar. Diese bilden ein Quellschema mit nur einer S-T TGD auf ein Zielschema ab, welches aus einer Ergebnis-Relation für das Anfrageergebnis besteht. So kann die Anfrage, die die Namen der Studierenden mit einem Masterabschluss zusammen mit ihren Noten ausgeben soll, wie folgt geschrieben werden:

```
SELECT Vorname, Nachname, Note
FROM K_STUDIERENDE NATURAL JOIN K_NOTEN
WHERE Abschluss = 'Master'
```

$$\begin{aligned}
\mathcal{M} =& \{S_K, \{\text{E}\}, \{\sigma\}\} \text{ mit} \\
\sigma =& \text{K-STUDIERENDE}(x_{\text{Ma}}, x_{\text{Na}}, x_{\text{Vo}}, \text{Master}) \wedge \text{K-NOTEN}(x_{\text{Mo}}, x_{\text{Ma}}, x_{\text{Se}}, x_{\text{No}}) \\
& \longrightarrow \text{E}(x_{\text{Vo}}, x_{\text{Na}}, x_{\text{No}}).
\end{aligned}$$

Mit dieser einfachen Darstellung von Anfragen können wir die gleichen Algorithmen für die Ausführung von Schemaabbildungen und Anfragen verwenden (siehe Definition 2.19). Die Einschränkungen, die sich aus der Nutzung dieser einfachen Darstellung von Anfragen ergeben, werden wir im Abschnitt 4.1 genauer diskutieren.

Nach der Definition von Schemaabbildungen können wir jetzt Beispielabbildungen für unser allgemeines Beispiel (siehe Abschnitt 1.1) erstellen. Bei der Integration von verschiedenen Quellen in ein globales Schema benötigen wir so viele Schemaabbildungen, wie wir Quellen haben. In unserem Fall (für die Integration der Rostocker und Kieler Uni-Datenbanken in das globale Schema) sind das die Schemaabbildungen $\mathcal{M}_R = (S_R, T, \Sigma_R)$ und $\mathcal{M}_K = (S_K, T, \Sigma_K)$. Die möglichen Mengen von Abbildungen (S-T TGDs) Σ_R und Σ_K seien wie folgt definiert:

$$\begin{aligned}
\Sigma_R =& \{\sigma_{R1}, \sigma_{R2}, \sigma_{R3}\} \text{ mit} \\
\sigma_{R1} =& \text{R-STUDIERENDE}(x_{\text{Ma}}, x_{\text{Na}}, x_{\text{Vo}}, x_{\text{St}}, x_{\text{Be}}) \\
& \longrightarrow \exists y_{\text{Ab}} : \text{T-STUDIERENDE}(x_{\text{Ma}}, x_{\text{Na}}, x_{\text{Vo}}, x_{\text{St}}, y_{\text{Ab}}), \\
\sigma_{R2} =& \text{R-TEILNAHME}(x_{\text{Te}}, x_{\text{Mo}}, x_{\text{Ma}}) \wedge \text{R-NOTEN}(x_{\text{Te}}, x_{\text{Se}}, x_{\text{No}}) \\
& \longrightarrow \text{T-NOTEN}(x_{\text{Mo}}, x_{\text{Ma}}, x_{\text{Se}}, x_{\text{No}}), \\
\sigma_{R3} =& \text{R-ZUSATZ}(x_{\text{Ma}}, x_{\text{Le}}) \longrightarrow \text{T-ZUSATZ}(x_{\text{Ma}}),
\end{aligned}$$

$$\begin{aligned}
\Sigma_K =& \{\sigma_{K1}, \sigma_{K2}, \sigma_{K3}, \sigma_{K4}\} \text{ mit} \\
\sigma_{K1} =& \text{K-STUDIERENDE}(x_{\text{Ma}}, x_{\text{Na}}, x_{\text{Vo}}, x_{\text{Ab}}) \\
& \longrightarrow \exists y_{\text{St}} : \text{T-STUDIERENDE}(x_{\text{Ma}}, x_{\text{Na}}, x_{\text{Vo}}, y_{\text{St}}, x_{\text{Ab}}), \\
\sigma_{K2} =& \text{K-NOTEN}(x_{\text{Mo}}, x_{\text{Ma}}, x_{\text{Se}}, x_{\text{No}}) \longrightarrow \text{T-NOTEN}(x_{\text{Mo}}, x_{\text{Ma}}, x_{\text{Se}}, x_{\text{No}}), \\
\sigma_{K3} =& \text{K-PRAKTUKUM}(x_{\text{Ma}}) \longrightarrow \text{T-ZUSATZ}(x_{\text{Ma}}), \\
\sigma_{K4} =& \text{K-SOFTSKILL}(x_{\text{Ma}}) \longrightarrow \text{T-ZUSATZ}(x_{\text{Ma}}).
\end{aligned}$$

Die Datenintegration mit den Schemaabbildungen $\mathcal{M}_R$ und $\mathcal{M}_K$ könnte mit dem Standard-CHASE auf Instanzen für Schemaabbildungen (siehe Definition 2.19) durchgeführt werden. Dafür würde der CHASE jeweils mit der Rostocker Datenbankinstanz und Σ_R sowie mit der Kieler Datenbankinstanz und Σ_K aufgerufen werden (ähnlich dem Beispiel im Abschnitt 2.4.2). Die Zwischenergebnisse I_{T_R} für die Rostocker Datenbankinstanz und I_{T_K} für die Kieler Datenbankinstanz bei der Datenintegration zwischen den einzelnen Quellen und dem globalen Schema T (bzw. der einzelnen CHASE-Ausführungen) würden folgendermaßen lauten:

$$
\begin{aligned}
I_{T_R} = \{&\text{T-STUDIERENDE}(1, \text{Fieber}, \text{Fabian}, \text{Informatik}, \eta_{\text{Ab}_1}),\\
&\text{T-STUDIERENDE}(2, \text{Sonne}, \text{Sarah}, \text{ITTI}, \eta_{\text{Ab}_2}),\\
&\text{T-NOTEN}(1, 1, \text{WS } 20/21, 1.7),\\
&\text{T-NOTEN}(2, 1, \text{SS } 21, 3.0),\\
&\text{T-NOTEN}(1, 2, \text{WS } 20/21, 2.3),\\
&\text{T-ZUSATZ}(1),\\
&\text{T-ZUSATZ}(2)\},
\end{aligned}
$$

$$
\begin{aligned}
I_{T_K} = \{&\text{T-STUDIERENDE}(2, \text{Sonne}, \text{Sarah}, \eta_{\text{St}_1}, \text{Bachelor}),\\
&\text{T-STUDIERENDE}(3, \text{Mueller}, \text{Max}, \eta_{\text{St}_2}, \text{Master}),\\
&\text{T-NOTEN}(1, 3, \text{WS } 20/21, 2.0),\\
&\text{T-NOTEN}(2, 2, \text{SS } 21, 1.3),\\
&\text{T-ZUSATZ}(2),\\
&\text{T-ZUSATZ}(3)\}.
\end{aligned}
$$

Die beiden Ergebnisinstanzen sind gleichzeitig die universelle Lösung für die jeweiligen Schemaabbildungen. Das liegt an den Eigenschaften des CHASE-Algorithmus (siehe Definition 2.19), was auch ein Grund dafür ist, dass wir ihn für die Anwendung der Schemaabbildungen verwenden.

Das Endergebnis der Datenintegration ist die Vereinigung der beiden Ergebnisinstanzen I_{T_R} und I_{T_K}. Dabei müssen wir etwaige Inkonsistenzen beachten. In unserem Fall können wir die Schlüsseleigenschaft der Matrikelnummer nutzen[1], um die Daten von Sarah Sonne aus I_{T_R} und I_{T_K} zusammen zu führen. Als Lösung erhalten wir die in den Tabellen 1.9, 1.10 und 1.11 beschriebene Instanz.

Im bisherigen Verlauf der Arbeit haben wir die Integritätsbedingungen für die Schemaabbildungen nur als S-T TGDs bezeichnet. Im Abschnitt 2.2 haben wir schon einen anderen Namen für S-T TGDs eingeführt, ohne diesen mit S-T TGDs zu verknüpfen. Die Rede ist von *Global Local as View-Abbildungen* (GLaV), welche als Kombination von *Global as View-* (GaV) und *Local as View-Abbildungen* (LaV) eingeführt wurden. Die Eigenschaften von GaV und LaV haben wir bereits im angesprochenen Abschnitt besprochen. Kurz zusammengefasst, beschreiben GaV-Abbildungen das globale Schema mit Views über den lokalen Schemata, während LaV-Abbildungen ein lokales Schema über Views auf dem globalen Schema beschreiben. In Definition 2.7 haben wir den Kopf und den Rumpf der S-T TGD als Konjunktionen von atomaren Ausdrücken beschrieben. Das entspricht genau der Kombination von GaV und LaV, da sowohl die lokalen Schemata als auch das globale Schema mit Views beschrieben werden. Anknüpfend an die formale Definition 2.7 der GLaV-S-T TGDs werden wir jetzt auch GaV- und LaV-Abbildungen formal definieren.

[1] Z. B. mittels CHASE auf der Vereinigung der Ergebnis-Instanzen und einer EGD, die bei gleicher Matrikelnummer die restlichen Attribute gleichsetzt.

Definition 2.11. (GaV, LaV, GLaV, nach [FKPT11b]): Eine S-T TGD, wie sie in Definition 2.7 beschrieben wurde, nennen wir auch *Global Local as View-Abbildung* (GLaV).
Eine S-T TGD, deren Kopf nur aus einem atomaren Ausdruck ohne existenzquantifizierte Variablen besteht, nennen wir *Global as View-Abbildung* (GaV). Sei die GaV

$$\forall \mathbf{x} : (\phi(\mathbf{x}) \longrightarrow R_T(\mathbf{x})),$$

gegeben, dann ist $R_T(\mathbf{x})$ ein Ausdruck über einem Zielschema und $\phi(\mathbf{x})$ eine Konjunktion von atomaren Ausdrücken über einem Quellschema. Eine Schemaabbildung, die nur nur aus GaV-Abbildungen besteht, nennen wir *GaV-Schemaabbildung*.
Eine S-T TGD, deren Rumpf nur aus einem atomaren Ausdruck besteht, nennen wir *Local as View-Abbildung* (LaV). Sei die LaV

$$\forall \mathbf{x} : (R_S(\mathbf{x}) \longrightarrow \exists \mathbf{y} : \psi(\mathbf{x}, \mathbf{y})),$$

gegeben, dann ist $R_S(\mathbf{x})$ ein Ausdruck über einem Quellschema und $\psi(\mathbf{x}, \mathbf{y})$ eine Konjunktion von atomaren Ausdrücken über einem Zielschema. Eine Schemaabbildung, die nur aus LaV-Abbildungen besteht, nennen wir *LaV-Schemaabbildung*.

Beispiele für GaV- und LaV-Abbildungen finden wir in den Beispiel-Schemaabbildungen zur Definition 2.10. Die S-T TGD in Abbildung σ_{R1} ist etwa eine LaV-Abbildung, da ihr Rumpf nur aus einem atomaren Ausdruck besteht. Weitere LaV-Abbildungen sind σ_{R3} sowie alle Abbildungen aus Σ_K. Eine GaV-Abbildung finden wir unter anderen in σ_{R2}, da der Kopf der S-T TGD nur aus einem atomaren Ausdruck besteht und keine existenzquantifizierten Variablen enthält. Die Abbildungen σ_{R3}, σ_{K2}, σ_{K3} und σ_{K4} sind ebenfalls GaV-Abbildungen. Da Letztere auch LaV-Abbildungen sind, sind diese sogar GaV-LaV-Abbildungen, was nicht mit GLaV-Abbildungen verwechselt werden darf.

Eine mögliche GLaV-Abbildung erhalten wir, wenn wir z. B. ausschließlich die Studierenden in das globale Schema übertragen, die bereits an einem Modul teilgenommen haben. Mit dem Verbund von R-STUDIERENDE- und R-TEILNAHME erhalten wir zwei Ausdrücke im Rumpf der S-T TGD, womit wir keine LaV-Abbildung mehr haben. Da T-STUDIERENDE noch ein unbekanntes Attribut für den Abschluss enthält, benötigen wir im Kopf der S-T TGD eine existenzquantifizierte Variable, was ein Ausschlusskriterium für GaV-Abbildungen ist. Die konstruierte GLaV-Abbildung sieht wie folgt aus:

$$\begin{aligned}\sigma_{GLaV} =& \text{R-STUDIERENDE}(x_{\text{Ma}}, x_{\text{Na}}, x_{\text{Vo}}, x_{\text{St}}, x_{\text{Be}}) \wedge \text{R-TEILNAHME}(x_{\text{Te}}, x_{\text{Mo}}, x_{\text{Ma}}) \\ &\longrightarrow \exists y_{\text{Ab}} : \text{T-STUDIERENDE}(x_{\text{Ma}}, x_{\text{Na}}, x_{\text{Vo}}, x_{\text{St}}, y_{\text{Ab}})\end{aligned}$$

Zum Abschluss dieses Abschnitts stellen wir noch eine Theorie vor, mit der es möglich ist, mehrere Schemaabbildungen zu verketten. Sollte sich z. B. das globale Schema verändern, kann man mit der *Komposition* der bestehenden Schemaabbildungen und der Schemaabbildung, die die Veränderung des globalen Schemas beschreibt, neue Schemaabbildungen berechnen, anstatt die bestehenden Schemaabbildungen manuell anzupassen.

Definition 2.12. (Komposition von Schemaabbildungen, [FKPT05]): Seien $\mathcal{M}_{12} = (S_1, S_2, \Sigma_{12})$ und $\mathcal{M}_{23} = (S_2, S_3, \Sigma_{23})$ Schemaabbildungen, sodass S_1, S_2 und S_3 keine gemeinsamen Relationenschemata haben. Eine Schemaabbildung $\mathcal{M}_{13} = (S_1, S_3, \Sigma_{13})$ ist eine *Komposition* von $\mathcal{M}_{12}$ und $\mathcal{M}_{23}$ (geschrieben: $\mathcal{M}_{13} = \mathcal{M}_{12} \circ \mathcal{M}_{23}$), wenn $\mathcal{M}_{13} = \{(I_1, I_3) \mid \exists I_2 : (I_1, I_2) \in \mathcal{M}_{12} \wedge (I_2, I_3) \in \mathcal{M}_{23}\}$.

Für die Komposition von zwei fortlaufenden Schemaabbildungen $\mathcal{M}_{12}$ und $\mathcal{M}_{23}$ ergeben sich nach [FKPT05] folgende interessante Eigenschaften:

- Wenn $\mathcal{M}_{12}$ und $\mathcal{M}_{23}$ nur aus GaV-Abbildungen bestehen, dann kann $\mathcal{M}_{12} \circ \mathcal{M}_{23}$ auch mittels GaV-Abbildungen ausgedrückt werden.
- Wenn $\mathcal{M}_{12}$ aus GaV-Abbildungen, $\mathcal{M}_{23}$ aber nicht nur aus GaV-Abbildungen besteht, dann kann $\mathcal{M}_{12} \circ \mathcal{M}_{23}$ nicht mittels GaV-Abbildungen ausgedrückt werden.
- Wenn $\mathcal{M}_{12}$ aus LaV- oder GLaV-Abbildungen besteht, dann kann $\mathcal{M}_{12} \circ \mathcal{M}_{23}$ nur[2] mittels *Second-Order TGDs* ausgedrückt werden.
- Wenn $\mathcal{M}_{12}$ oder $\mathcal{M}_{23}$ aus *Second-Order TGDs* besteht, dann kann $\mathcal{M}_{12} \circ \mathcal{M}_{23}$ nur mittels *Second-Order TGDs* ausgedrückt werden.

Definition 2.13. (Second-Order TGD, [FKPT05]): Sei S ein Quellschema und T ein Zielschema. Eine *Second-Order TGD* ist eine Formel der Form

$$\exists \mathbf{f}(\forall \mathbf{x}_1(\phi_1 \longrightarrow \psi_1) \wedge ... \wedge \forall \mathbf{x}_n(\phi_n \longrightarrow \psi_n))$$

wobei:

1. Jedes Element in $\mathbf{f}$ ist ein Funktionssymbol.
2. Jedes ϕ_i ist eine Konjunktion von
   - atomaren Ausdrücken der Form $R_S(a_1, ..., a_p)$, wobei R_S für den Bezeichner eines Relationenschemas aus S steht und $a_1, ..., a_p$ Variablen aus $\mathbf{x}_i$ sind, die nicht disjunkt sein müssen,

   und
   - Gleichheitsatomen der Form $t = t'$, wobei t und t' Terme basierend auf $\mathbf{x}_i$ und $\mathbf{f}$ sind.
3. Jedes ψ_i ist Konjunktion von atomaren Ausdrücken der Form $R_T(t_1, ..., t_q)$, wobei R_T für den Bezeichner eines Relationenschemas aus T steht und $t_1, ..., t_q$ Terme basierend auf $\mathbf{x}_i$ und $\mathbf{f}$ sind.
4. Jede Variable in $\mathbf{x}_i$ kommt in einem atomaren Ausdruck in ϕ_i vor.

Die Komposition von Schemaabbildungen sowie die Second-Order TGDs werden in dieser Arbeit nur für die Diskussion über mögliche weitere Ansätze der GaLVE-Technik in Abschnitt 4.4.2 wichtig sein. Im Konzept-Kapitel werden wir uns für eine Variante entscheiden, die ohne Komposition und Second-Order TGDs auskommt. Zum Abschluss dieses Abschnittes werden wir noch ein kurzes Beispiel für die Komposition von Schemaabbildungen und die Notwendigkeit von Second-Order TGDs geben.

Sei die Schemaabbildung $\mathcal{M}_{S_K T} = (S_K, T, \Sigma_{S_K T})$ mit $\Sigma_{S_K T} = \{\sigma_{S_K T}\}$ mit den uns bekannten Schemata S_K, T (siehe allgemeines Beispiel im Abschnitt 1.1) gegeben und

$$\sigma_{S_K T} = \text{K-NOTEN}(x_{\text{Mo}}, x_{\text{Ma}}, x_{\text{Se}}, x_{\text{No}}) \longrightarrow \text{T-NOTEN}(x_{\text{Mo}}, x_{\text{Ma}}, x_{\text{Se}}, x_{\text{No}}).$$

Sei außerdem die Schemaabbildung $\mathcal{M}_{TT'} = (T, T', \Sigma_{TT'})$ mit $\Sigma_{TT'} = \{\sigma_{TT'}\}$ gegeben und

$$\sigma_{TT'} = \text{T-NOTEN}(x_{\text{Mo}}, x_{\text{Ma}}, x_{\text{Se}}, x_{\text{No}}) \longrightarrow \exists y_{\text{St}} : \text{T'-NOTEN}(x_{\text{Mo}}, x_{\text{Ma}}, x_{\text{Se}}, y_{\text{St}}, x_{\text{No}}).$$

[2] Nach Absprache mit Prof. Andreas Heuer ist es durchaus möglich, dass die Komposition einer nicht-GaV-Abbildung mit einer anderen Abbildung mittels einer einfachen S-T TGD ausgedrückt werden kann. Eine Charakterisierung der Teilklasse von LaV- und GLaV-Abbildungen, bei der die Komposition wieder eine einfache S-T TGD ergibt, wurde noch nicht definiert.

Das Schema T' entspricht, abgesehen von dem ' in den Relationennamen, dem Schema T bis auf das zusätzliche Studiengang-Attribut in der T'-NOTEN-Relation. Anstatt $\Sigma_{S_K T}$ per Hand an das neue Schema T' anzupassen oder immer beide Schemaabbildungen hintereinander auszuführen (beachte, dass es normalerweise sehr viele Schemaabbildungen gibt), können wir mit $\mathcal{M}_{S_K T} \circ \mathcal{M}_{TT'}$ eine Schemaabbildung berechnen, die von S_K direkt in das neue Schema T' abbildet. Da $\Sigma_{S_K T}$ eine GaV-LaV-Abbildung und $\Sigma_{TT'}$ eine LaV-Abbildung ist, kann die Komposition in diesem Fall nur mit einer LaV-Abbildung (allgemein nicht mit einer GaV-Abbildung) ausgedrückt werden:

$$\sigma_{S_K T'} = \text{K-NOTEN}(x_{\text{Mo}}, x_{\text{Ma}}, x_{\text{Se}}, x_{\text{No}}) \longrightarrow \exists y_{\text{St}} : \text{T'-NOTEN}(x_{\text{Mo}}, x_{\text{Ma}}, x_{\text{Se}}, y_{\text{St}}, x_{\text{No}}).$$

Die Schemaabbildung $\mathcal{M}_{S_K T'} = (S_K, T', \Sigma_{S_K T'})$ mit $\Sigma_{S_K T'} = \{\sigma_{S_K T'}\}$ beschreibt nun die direkte Abbildung von S_K in das neue Schema T'.

Sei im zweiten Beispiel die Schemaabbildung $\mathcal{M}'_{S_K T} = (S, T, \Sigma'_{S_K T})$ mit $\Sigma'_{S_K T} = \{\sigma'_{S_K T}\}$ gegeben und

$$\sigma'_{S_K T} = \text{K-STUDIERENDE}(x_{\text{Ma}}, x_{\text{Na}}, x_{\text{Vo}}, x_{\text{Ab}}) \longrightarrow \exists y_{\text{St}} : \text{T-STUDIERENDE}(x_{\text{Ma}}, x_{\text{Na}}, x_{\text{Vo}}, y_{\text{St}}, x_{\text{Ab}}).$$

Sei außerdem die Schemaabbildung $\mathcal{M}'_{TT'} = (T, T', \Sigma'_{TT'})$ mit $\Sigma'_{TT'} = \{\sigma'_{TT'}\}$ gegeben und

$$\sigma'_{TT'} = \text{T-STUDIERENDE}(x_{\text{Ma}}, x_{\text{Na}}, x_{\text{Vo}}, x_{\text{St}}, x_{\text{Ab}}) \longrightarrow \text{T'-STUDIERENDE}(x_{\text{Ma}}, x_{\text{Na}}, x_{\text{Vo}}, x_{\text{St}}).$$

Das Schema T' entspricht, abgesehen von dem ' in den Relationennamen, dem Schema T bis auf das fehlende Abschluss-Attribut in der T'-NOTEN-Relation. Wieder können wir mit $\mathcal{M}'_{S_K T} \circ \mathcal{M}'_{TT'}$ eine Schemaabbildung berechnen, die von S_K direkt in das neue Schema T' abbildet. Diesmal ist $\Sigma'_{S_K T}$ aber keine GaV-Abbildung, was bedeutet, dass wir die Komposition nur mit Second-Order TGDs ausdrücken können. Das Problem liegt in unserem Fall an der existenzquantifizierten Variable im Kopf von $\sigma'_{S_K T}$. Diese führt bei der Ausführung der Abbildung zu unbekannten Werten für den Studiengang. Eine weitere Abbildung mit $\sigma'_{TT'}$ wird für alle möglichen Werte und Nullwerte für den Studiengang ausgeführt. Bei der Komposition muss so eine Funktion für diese existenzquantifizierte Variable eingeführt werden, die angibt, von welchen bekannten Werten dieser unbekannte Wert abhängt. Das führt zu folgender Second-Order TGD:

$$\begin{aligned}\sigma'_{S_K T'} = \exists f(\forall x_{\text{Ma}}, x_{\text{Na}}, x_{\text{Vo}}, x_{\text{Ab}}(&\text{K-STUDIERENDE}(x_{\text{Ma}}, x_{\text{Na}}, x_{\text{Vo}}, x_{\text{Ab}}) \\ &\longrightarrow \text{T'-STUDIERENDE}(x_{\text{Ma}}, x_{\text{Na}}, x_{\text{Vo}}, f(x_{\text{Ma}}, x_{\text{Na}}, x_{\text{Vo}}, x_{\text{Ab}}))).\end{aligned}$$

Die Schemaabbildung $\mathcal{M}'_{S_K T'} = (S, T', \Sigma'_{S_K T'})$ mit $\Sigma'_{S_K T'} = \{\sigma'_{S_K T'}\}$ beschreibt nun die direkte Abbildung von S_K in das neue Schema T'. Für die Anwendung der Abbildung müssen nun allerdings Techniken eingesetzt werden, die auch mit Second-Order TGDs umgehen können.

Im nächsten Abschnitt werden wir den CHASE-Algorithmus für die Ausführung von Schemaabbildungen einführen. Dieser wird mit unserer Definition nur mit einfachen S-T TGDs arbeiten und keine Second-Order TGDs unterstützen.

2.4. CHASE

Der CHASE-Algorithmus ist für diverse Anwendungsfälle geeignet und kann mit einer Vielzahl an verschiedenen Parametern arbeiten. In der Datenbankforschung gilt er daher auch als Universalalgorithmus. In [AH19] wurden die Anwendungsgebiete des CHASEs nach Art der übergebenen Abhängigkeiten und Objekte definiert. Die sich daraus ergebende Einordnung ist in Tabelle 2.1 aufgeführt. Die ersten drei Spalten der Tabelle ergeben sich aus der allgemeinen Definition des CHASEs:

$$\text{CHASE}_{\star}(\bigcirc) = \circledast$$

Der Parameter $\star$ symbolisiert eine Menge von Abhängigkeiten, die in ein Objekt $\bigcirc$, mit dem Ergebnis $\circledast$, eingechaset werden.

	Parameter $\star$	Objekt $\bigcirc$	Ergebnis $\circledast$	Ziel/Anwendungsgebiet
0.	Abhängigkeiten	DB-Schema[3]	DB-Schema mit Integritätsbedingungen	optimierter DB-Entwurf
I.	Abhängigkeiten	Anfragen	Anfragen	Semantische Optimierung
II.	Sichten	Anfragen	Anfragen auf Sichten	AQuV
II′.	Operationen	Anfragen	Anfragen auf Operationen	AQuO
III.	(S-T) TGDs, EGDs	Quell-DB	Ziel-DB	Datenaustausch, -integration
IV.	TGDs, EGDs	DB	modifizierte DB	Cleaning
V.	TGDs, EGDs	unvollständige DB	Anfrageergebnis	Sichere Antworten
VI.	(S-T) TGDs, EGDs	DB	Anfrageergebnis	invertierbare Auswertung

Tabelle 2.1.: Übersicht Anwendungsgebiete CHASE [AH19]

Der Fall *0* spiegelt die ursprüngliche Idee des CHASEs wieder – einen Datenbank-Entwurf mithilfe von Abhängigkeiten zu optimieren. Die Rede ist von funktionalen, Verbund-, und mehrwertigen Abhängigkeiten, die z. B. sicherstellen sollen, dass eine Zerlegung in Relationenschemata verlustfreie Verbunde garantiert [ABU79, MMS79].

Diese Abhängigkeiten können auch in Anfragen an eine Datenbank eingechaset werden. Dabei können Verbunde, die auf Grund von bestehenden Schlüsselbeziehungen irrelevant sind, entfernt werden, was die Anfragen um ein Vielfaches beschleunigen kann (Fall I). In den Fällen II und II' werden Sichten oder Operationen in die Anfragen eingechaset. Das kann eine Anfragebeantwortung ausschließlich anhand der Sichten und Operatoren ermöglichen, sodass z. B. Zugriffe auf den ganzen Datenbestand entfallen. Diese Verfahren werden *Answering Queries Using Views* bzw. *Answering Queries Using Operators* (AQuV bzw. AQuO) genannt. Mit AQuV werden beispielsweise in mobilen Anwendungen Speicherplatz und kostenintensive Zugriffe auf Remoteserver gespart, indem die Anfragen auf die lokal verfügbaren Sichten umgeschrieben werden [LMS95].

Neben dem CHASE auf Anfragen und Schemata lässt sich der Algorithmus auch auf Datenbanken anwenden. Dabei werden vor allem verschiedene Bedingungen in Form von TGDs, EGDs und S-T TGDs in Instanzen eingearbeitet. Das Ergebnis liegt dann, wie im Fall III, in Form von einer Zieldatenbank vor, welche Austauschdaten oder Integrationsdaten beinhaltet. Im Fall IV erhält man eine modifizierte Datenbank, die mithilfe der Bedingungen gecleant worden ist (siehe Data Cleaning im Abschnitt 2.2). Das *Cleaning* besteht vor allem darin, Duplikate zu eliminieren und Lücken in den Daten zu füllen. Es werden aber auch Inkonsistenzen in Datensätzen erkannt und in einigen Fällen automatisch behoben. Zuletzt kann das Ergebnis des CHASE auch ein Anfrageergebnis sein. So können im Fall V Sichere Antworten an eine unvollständige Datenbank garantiert werden oder wie im Fall VI die Herkunft der Daten in der

[3] PJ-Mapping

invertierten Auswertung einer Anfrage an eine Datenbank geklärt werden (auch *Provenance Management* genannt, mehr dazu in [AH18b, AH19] sowie Abschnitt 3.3).

Für diese Arbeit ist vor allem der Fall III vom Interesse. Der Fall V wird in Abschnitt 3.2 noch eine Rolle spielen, wenn wir vorstellen, wie die Theorie der gesicherten Antworten in Verbindung mit dem Disjunktiven-CHASE anwendet wird. Der VI. Fall wird in Abschnitt 3.3 genauer behandelt.

2.4.1. CHASE-Theorie

Die Hauptaufgabe des CHASEs ist die Suche nach Homomorphismen zwischen einer Menge von Integritätsbedingungen $\mathcal{B} := \{b_1, ..., b_m\}$ ($\star$ in Tabelle 2.1) und einem CHASE-Objekt $\mathcal{O}_i$ ($\bigcirc$ in Tabelle 2.1) welche *Trigger* genannt werden. Mithilfe sogenannter *aktiver Trigger* kann der CHASE in dem *CHASE-Schritt* die einzelnen Integritätsbedingungen $b_j \in \mathcal{B}$ in $\mathcal{O}_i$ einchasen. Als Ergebnis erhalten wir ein modifiziertes CHASE-Objekt $\mathcal{O}_{i+1}$. Zwischen $\mathcal{O}_i$ und $\mathcal{O}_{i+1}$ entsteht dann ebenfalls ein Homomorphismus, welcher einigen Ersetzungsregeln unterliegt. Für die von uns betrachteten Instanzen als CHASE-Objekt ergeben sich die Ersetzungsregeln, wie sie für die Homomorphismen in Definition 2.9 definiert wurden.

Zuerst werden wir die angesprochenen Trigger und darauf aufbauend den CHASE-Schritt auf Instanzen definieren. Nach der Definition des Standard-CHASEs auf Instanzen folgt noch die Bestimmung des Ergebnisses einer Schemaabbildung, die mit dem CHASE umgesetzt wurde.

Definition 2.14. (Trigger, nach [BKM$^+$17]): Der Homomorphismus g zwischen einer Integritätsbedingung $b \in \mathcal{B}$ [(S-T) TGD|EGD] und einem CHASE-Objekt $\mathcal{O}$ wird *Trigger* genannt. Dabei wird der Rumpf von b auf $\mathcal{O}$ abgebildet.

Sei $\mathcal{O}$ eine Instanz I, dann wird der Rumpf von b auf die Tupel aus den Relationen in I abgebildet (geschrieben: $g : \phi(\mathbf{x}) \longrightarrow I$), wobei die einzelnen Variablen x_i aus den atomaren Ausdrücken von $\phi(\mathbf{x})$ auf die einzelnen Attribut- und Nullwerte der Tupel abgebildet werden.

Im Weiteren werden wir Trigger als Abbildung von den Ausdrücken der Integritätsbedingungen auf die Tupel der Instanzen bezeichnen, anstatt ausführlich von Abbildungen der Variablen aus den Ausdrücken der Integritätsbedingungen auf die Attribut- und Nullwerte der Tupel aus den Instanzen zu sprechen.

Definition 2.15. (Aktiver Trigger, nach [BKM$^+$17]): Ein *aktiver Trigger* ist ein Trigger g, für den gilt:

- Wenn b eine (S-T) TGD ist, dann existiert keine Erweiterung von g zu einem Homomorphismus vom Kopf der (S-T) TGD zu $\mathcal{O}$ (geschrieben: $\psi(\mathbf{x}, \mathbf{y}) \longrightarrow \mathcal{O}$).

- Wenn b eine EGD ist, dann ist $g(x_m) \neq g(x_n)$ für mindestens ein Gleichheitsatom $(x_m = x_n)$ aus dem Kopf der EGD.

Ein CHASE-Objekt $\mathcal{O}$ *genügt* b genau dann, wenn kein aktiver Trigger für b in $\mathcal{O}$ existiert.

Betrachten wir nun ein Beispiel für Trigger und aktive Trigger. Sei als CHASE-Objekt die Instanz I und als Parameter die Integritätsbedingungen b_1 (TGD), b_2 (EGD) gegeben:

$$
\begin{aligned}
I =& \{\text{R-STUDIERENDE}(1,\ \eta_{\text{Na}}\ , \text{Fabian}, \text{Informatik}, \text{Tel: } 0123456789),\\
& \text{R-STUDIERENDE}(1, \text{Fieber}, \text{Fabian}, \text{Informatik}, \text{Tel: } 0123456789),\\
& \text{R-TEILNAHME}(2, 2, 1),\\
& \text{R-NOTEN}(2, \text{SS } 21, 3.0)\},\\
b_1 =& \text{R-STUDIERENDE}(x_{\text{Ma}}, x_{\text{Na}}, x_{\text{Vo}}, x_{\text{St}}, x_{\text{Be}}) \wedge \text{R-TEILNAHME}(x_{\text{Te}}, x_{\text{Mo}}, x_{\text{Ma}})\\
& \longrightarrow \exists y_{\text{Se}}, y_{\text{No}} : \text{R-NOTEN}(x_{\text{Te}}, y_{\text{Se}}, y_{\text{No}}),\\
b_2 =& \text{R-STUDIERENDE}(x_{\text{Ma}}, x_{\text{Na}_1}, x_{\text{Vo}_1}, x_{\text{St}_1}, x_{\text{Be}_1}) \wedge \text{R-STUDIERENDE}(x_{\text{Ma}}, x_{\text{Na}_2}, x_{\text{Vo}_2}, x_{\text{St}_2}, x_{\text{Be}_2})\\
& \longrightarrow (x_{\text{Na}_1} = x_{\text{Na}_2}), (x_{\text{Vo}_1} = x_{\text{Vo}_2}), (x_{\text{Be}_1} = x_{\text{Be}_2}).
\end{aligned}
$$

Für die Instanz I und die Integritätsbedingung b_1 existieren zwei Trigger g_0 und g_1. Beide bilden den R-STUDIERENDE-Ausdruck aus dem Rumpf von b_1 jeweils auf eines der R-STUDIERENDE-Tupel aus I sowie den R-TEILNAHME-Ausdruck aus dem Rumpf von b_1 auf das R-TEILNAHME-Tupel aus I ab. Die Abbildungen in g_0 gestalten sich wie folgt: $g_0(x_{\text{Ma}}) = 1$, $g_0(x_{\text{Na}}) = \eta_{\text{Na}}$, $g_0(x_{\text{Vo}}) = \text{Fabian}$, $g_0(x_{\text{St}}) = \text{Informatik}$, $g_0(x_{\text{Be}}) = \text{Tel: } 0123456789$, $g_0(x_{\text{Te}}) = 2$, $g_0(x_{\text{Mo}}) = 2$, die Abbildungen in g_1 sind identisch mit denen aus g_0, bis auf die Abbildung des Namens $g_1(x_{\text{Na}}) = \text{Fieber}$, da dieser im zweiten Tupel explizit gegeben ist. Da es auch eine Erweiterung von g_0 und g_1 hin zu einer Abbildung des R-NOTEN-Ausdrucks aus dem Kopf von b_1 auf das R-NOTEN-Tupel in I gibt, sind beide keine aktiven Trigger ($g_0(x_{\text{Se}}) = \text{SS } 21$, $g_0(x_{\text{No}}) = 3.0$, g_1 identisch g_0).

Im Gegensatz zu b_1 gibt es für die Integritätsbedingung b_2 einen aktiven Trigger auf I. Hier werden die R-STUDIERENDE-Ausdrücke aus dem Rumpf von b_2 auf die beiden R-STUDIERENDE-Tupel aus I abgebildet. Die Abbildungen in g_2 gestalten sich damit wie folgt: $g_2(x_{\text{Ma}}) = 1$, $g_2(x_{\text{Na}_1}) = \eta_{\text{Na}}$, $g_2(x_{\text{Na}_2}) = \text{Fieber}$, $g_2(x_{\text{Vo}_1}) = \text{Fabian}$, $g_2(x_{\text{Vo}_2}) = \text{Fabian}$, $g_2(x_{\text{St}_1}) = \text{Informatik}$, $g_2(x_{\text{St}_2}) = \text{Informatik}$, $g_2(x_{\text{Be}_1}) = \text{Tel: } 0123456789$, $g_2(x_{\text{Be}_2}) = \text{Tel: } 0123456789$. Der Trigger ist aktiv, da für das Gleichheitsatom $(x_{\text{Na}_1} = x_{\text{Na}_2})$, mit den Abbildungen $g_2(x_{\text{Na}_1}) = \eta_{\text{Na}}$ und $g_2(x_{\text{Na}_2}) = \text{Fieber}$, die Ungleichheit $g_2(x_{\text{Na}_1}) \neq g_2(x_{\text{Na}_2})$ bzw. $\eta_{\text{Na}} \neq \text{Fieber}$ gilt.

Definition 2.16. (CHASE-Schritt, [FKMP03]): Sei $g : \phi(\mathbf{x}) \longrightarrow \mathcal{O}_i$ ein aktiver Trigger für eine Integritätsbedingung $b \in \mathcal{B}$ und ein CHASE-Objekt $\mathcal{O}_i$. Die Anwendung von b unter g auf $\mathcal{O}_i$ mit dem Ergebnis eines modifizieren CHASE-Objektes $\mathcal{O}_{i+1}$ nennt man einen *CHASE-Schritt* $\mathcal{O}_i \xrightarrow{b,g} \mathcal{O}_{i+1}$.

Algorithmus 1 beschreibt das Konzept des Standard-CHASE auf Instanzen. Die Zeilen 5 bis 11 stellen den CHASE-Schritt auf Instanzen dar.

Algorithmus 1 Standard-CHASE auf Instanzen($\mathcal{B}$, I)

Input: Menge von Integritätsbedingungen $\mathcal{B}$ [(S-T) TGDs, EGDs], Instanz I_0
Output: Modifizierte Instanz I_m

```
1:  while I_i ≠ ⊥ ∧ ∃ aktive Trigger g_k für b_j ∈ B und I_i, 0 ≤ i ≤ m, j,k ∈ N_0 do
2:    for all b_j ∈ B do
3:      for all Trigger g_k für b_j, I_i do
4:        if g_k ist aktiver Trigger then
5:          if b_j ist (S-T) TGD then
6:            Erweitere g falls nötig, füge neue Tupel zu I_i hinzu (I_i --b_j,g_k--> I_{i+1})
7:          else if b_j ist EGD then
8:            if Verglichene Werte sind verschiedene Konstanten then
9:              I_m = ⊥ (I_i --b_j,g_k--> ⊥)
10:           else
11:             Ersetzte Nullwerte durch Nullwerte oder Konstanten (I_i --b_j,g_k--> I_{i+1})
```

Definition 2.17. (CHASE-Schritt auf Instanzen, [FKMP03]): Seien alle Bedingungen für einen CHASE-Schritt $I_i \xrightarrow{b,g} I_{i+1}$ nach Definition 2.16 mit einer Instanz I_i als CHASE-Objekt gegeben.

(a) Sei b eine TGD.

- Die Anwendung von b erweitert I_i um Tupel $R_l(w_1, ..., w_p)$ mit den Bezeichnern der Relationenschemata R_l gleich den Bezeichnern der Ausdrücke $R_l(a_1, ..., a_p)$ aus dem Kopf der TGD. Falls $a_q \in \{a_1, ..., a_p\}$ für eine existenzquantifizierte Variable y_q steht, wird der Homomorphismus g um die Abbildung von y_q auf einen neuen benannten Nullwert η_q erweitert. Die Attribut- und Nullwerte $w_q \in \{w_1, ..., w_p\}$ ergeben sich dabei aus der Anwendung des Homomorphismus auf die Variablen der Ausdrücke des TGD-Kopfes, $g(a_q)$.

(b) Sei b eine EGD.

- Falls die Abbildungen $g(x_n)$ und $g(x_m)$ für ein Gleichheitsatom $(x_n = x_m)$ unterschiedliche Konstanten ergeben (z. B. $g(x_n) = 1$, $g(x_m) = 2$), dann schlägt der CHASE fehl (z. B. $I_{i+1} = \perp$).

- Falls eine Variable des Gleichheitsatoms $(x_m = x_m)$ auf eine Konstante abgebildet wird und die andere auf einen benannten Nullwert (z. B. $g(x_m) = 1$, $g(x_n) = \eta_{\mathrm{Ma}}$), dann wird der Nullwert überall durch die Konstante ersetzt (z. B. $h(\eta_{\mathrm{Ma}}) = 1$ mit $h : I_i \longrightarrow I_{i+1}$).

- Falls beide Variablen des Gleichheitsatoms $(x_m = x_n)$ auf unterschiedlich benannte Nullwerte abgebildet werden (z. B. $g(x_m) = \eta_{\mathrm{No_1}}$, $g(x_n) = \eta_{\mathrm{No_2}}$), dann wird ein Nullwert überall durch den anderen ersetzt. Die Auswahl erfolgt über den durch die Namenskonvention vorgeschriebenen Index der Nullwerte. Die Konvention besagt, dass verschiedene Nullwerte für ein Attribut durchnummeriert werden. So kann man den Nullwert mit dem größeren Index durch den Nullwert mit dem kleineren Index ersetzen (z. B. $h(\eta_{\mathrm{No_2}}) = \eta_{\mathrm{No_1}}$ mit $h : I_i \longrightarrow I_{i+1}$).

  Diese Regeln werden für jedes Gleichheitsatom aus b auf I_i angewendet. Bei der Anwendung wird ein Homomorphismus h zwischen den Instanzen I_i und I_{i+1} gebildet, der die Werte, die ersetzt werden (z. B. $g(x_n) = \eta_{\mathrm{Ma}}$), auf die ersetzenden Werte (z. B. $g(x_m) = 1$) abbildet (z. B. $(h(g(x_n)) = g(x_m)) \longrightarrow h(\eta_{\mathrm{Ma}}) = 1$ mit $h : I_i \longrightarrow I_{i+1}$).

Die Anwendung eines CHASE-Schrittes überführt die Instanz I_i in eine modifizierte Instanz I_{i+1}. Schlägt ein CHASE-Schritt fehl, wird anstatt einer Instanz das Symbol für Falschheit $\perp$ zurückgegeben.

Definition 2.18. (Standard-CHASE auf Instanzen, [FKMP03]): Sei $\mathcal{B}$ eine Menge von (S-T) TGDs und EGDs und I eine Instanz.

- Eine *CHASE-Sequenz* auf I mit $\mathcal{B}$ (geschrieben: $CHASE_{\mathcal{B}}(I)$) ist eine (endliche oder unendliche) Sequenz von CHASE-Schritten $I_i \xrightarrow{b_j,g_k} I_{i+1}$, mit $i, j, k \in \mathbb{N}_0, I_0 = I$, $b_j \in \mathcal{B}$ und g_k Homomorphismus für b_j, I_i.

- Ein *endlicher CHASE* auf I mit $\mathcal{B}$ ist eine endliche Sequenz $I_i \xrightarrow{b_j,g_k} I_{i+1}, 0 \leq i \leq m$, mit der Bedingung, dass entweder (a) $I_m = \perp$ oder (b) es existiert keine Integritätsbedingung $b_j \in \mathcal{B}$ und es gibt keinen Homomorphismus g_k, sodass b_j mit g_k auf I_m abgebildet werden kann. Wir bezeichnen I_m als das Ergebnis des endlichen CHASEs – für den Fall (a) das Ergebnis eines *fehlgeschlagenen endlichen CHASEs* und für den Fall (b) das Ergebnis eines *erfolgreichen endlichen CHASEs*.

Mit anderen Worten werden beim Standard-CHASE solange CHASE-Schritte angewandt, bis der CHASE entweder fehlschlägt oder es keine aktiven Trigger mehr für alle $b_j \in \mathcal{B}$ auf dem CHASE-Objekt $\mathcal{O}_i$ gibt. Die Auswahl der Integritätsbedingungen erfolgt dabei nicht-deterministisch. Dieser Nichtdeterminismus in Verbindung mit möglichen nicht konfluenten Integritätsbedingungen führt dazu, dass die Terminierung

des Standard-CHASEs nicht entscheidbar ist und verschiedene CHASE-Sequenzen über dem gleichen Objekt mit gleichen Integritätsbedingungen unterschiedliche Ergebnisse haben können.

Definition 2.19. (Ergebnis für Schemaabbildungen mit Hilfe des CHASEs): Sei $\mathcal{M} = (S, T, \Sigma)$ eine Schemaabbildung mit Quellschema $S = \{R_{S1}, ..., R_{Sp}\}$, Zielschema $T = \{R_{T1}, ..., R_{Tq}\}$ mit $S \cap T = \emptyset$ und einer Menge Σ von S-T TGDs vom Quellschema zum Zielschema. Sei außerdem $I_{S \cup T}$ die Ergebnisinstanz (unter $S \cup T$) eines erfolgreichen endlichen CHASEs nach Definition 2.18 auf einer Quellinstanz I_S unter Quellschema S mit den Integritätsbedingungen $\mathcal{B}$, wobei $\Sigma \subseteq \mathcal{B}$. Die Ergebnisinstanz I_T von $\mathcal{M}$ ergibt sich durch das Einschränken von $I_{S \cup T}$ auf Relationen über dem Ziel-Schema T:

$$I_T = r_i(R_i) \in I_{S \cup T} : R_i \in T, i \in \{1, ..., p + q\}.$$

Sei $I_{S \cup T} = \perp$ das Ergebnis des fehlgeschlagenen endlichen CHASEs, für $\mathcal{M}$, I_S und $\mathcal{B}$ wie oben, dann ist $I_T = \perp$ das Ergebnis des fehlgeschlagenen CHASEs für Schemaabbildungen.
Sei $\mathcal{M}$ und I_S wie oben und sei $\mathcal{B} = \Sigma$, dann terminiert der Standard-CHASE für Schemaabbildungen $CHASE_{\mathcal{M}}(I_S)$ in Polynomialzeit mit der universellen Lösung I_T für I_S unter $\mathcal{M}$ [FKMP05].

2.4.2. CHASE-Beispiel

Seien die Instanz I_{S_R} und die Integritätsbedingung b unter dem Rostocker Universitätsschema S_R sowie die Schemaabbildung $\mathcal{M} = (S_R, T, \Sigma)$ wie folgt gegeben:

$$\begin{aligned}
S_R =& \{\text{R-STUDIERENDE}, \text{R-TEILNAHME}, \text{R-NOTEN}\},\\
T =& \{\text{T-STUDIERENDE}, \text{T-NOTEN}\},\\
\Sigma =& \{\sigma_1, \sigma_2\},\\
I_{S_R} =& \{\text{R-STUDIERENDE}(1, \text{Fieber}, \text{Fabian}, \text{Informatik}, \text{Tel: } 0123456789),\\
& \text{R-TEILNAHME}(2, 2, 1)\},\\
b =& \text{R-STUDIERENDE}(x_{\text{Ma}}, x_{\text{Na}}, x_{\text{Vo}}, x_{\text{St}}, x_{\text{Be}}) \wedge \text{R-TEILNAHME}(x_{\text{Te}}, x_{\text{Mo}}, x_{\text{Ma}})\\
& \longrightarrow \exists y_{\text{Se}}, y_{\text{No}} : \text{R-NOTEN}(x_{\text{Te}}, y_{\text{Se}}, y_{\text{No}}),\\
\sigma_1 =& \text{R-STUDIERENDE}(x_{\text{Ma}}, x_{\text{Na}}, x_{\text{Vo}}, x_{\text{St}}, x_{\text{Be}})\\
& \longrightarrow \exists y_{\text{Ab}} : \text{T-STUDIERENDE}(x_{\text{Ma}}, x_{\text{Na}}, x_{\text{Vo}}, x_{\text{St}}, y_{\text{Ab}}),\\
\sigma_2 =& \text{R-TEILNAHME}(x_{\text{Te}}, x_{\text{Mo}}, x_{\text{Ma}}) \wedge \text{R-NOTEN}(x_{\text{Te}}, x_{\text{Se}}, x_{\text{No}})\\
& \longrightarrow \text{T-NOTEN}(x_{\text{Mo}}, x_{\text{Ma}}, x_{\text{Se}}, x_{\text{No}}).
\end{aligned}$$

Wie in Algorithmus 1 in den Zeilen 1 bis 4 beschrieben, wird beim Standard-CHASE der CHASE-Schritt der Zeilen 5 bis 11 für alle aktiven Trigger angewandt. Das geschieht in unserem Beispiel für alle Integritätsbedingungen $b_j \in \mathcal{B} = \Sigma \cup b$, solange bis es für keine Integritätsbedingung einen aktiven Trigger auf der aktuellen Instanz gibt oder der CHASE fehlschlägt.

Für die Berechnung von $CHASE_{\Sigma \cup b}(I_{S_R})$ suchen wir also nach Triggern für die Integritätsbedingungen in $\mathcal{B} = \{\sigma_1, \sigma_2, b\}$ und die Instanz $I_{S_R} \equiv I_0$. Wählen[4] wir als Erstes die Integritätsbedingung σ_1. Die Trigger ergeben sich, indem zwischen dem Rumpf von σ_1 und den Tupeln von I_0 nach Homomorphismen gesucht wird. Es existiert ein Homomorphismus g_0 mit einer Abbildung zwischen dem R-STUDIERENDE-Ausdruck im Rumpf von σ_1 und dem R-STUDIERENDE-Tupel aus I_0. Der Homomorphismus bzw. der Trigger g_0 besteht aus den Abbildungen $g_0(x_{\text{Ma}}) = 1$, $g_0(x_{\text{Na}}) =$ Fieber, $g_0(x_{\text{Vo}}) =$ Fabian, $g_0(x_{\text{St}}) =$ Informatik, $g_0(x_{\text{Be}}) =$ Tel: 0123456789.

[4] Bei mehreren Integritätsbedingungen in $\mathcal{B}$ erfolgt die Auswahl nicht-deterministisch.

Bei der Prüfung auf einen aktiven Trigger wird nach Erweiterungen von g_0 zu einem Homomorphismus gesucht, der den Kopf von σ_1 auf I_0 abbildet. Da es keine Abbildung vom T-STUDIERENDE-Ausdruck nach I_0 gibt, können wir mit dem aktiven Trigger g_0 fortfahren und den CHASE-Schritt $I_0 \xrightarrow{\sigma_1, g_0} I_1$ durchführen. Dabei unterscheidet man σ_1 nach (S-T) TGDs und EGDs. In unserem Fall wird I_0 mit Tupeln erweitert, die aus der Anwendung von g_0 auf den Ausdrücken aus dem S-T TGD-Kopf stammen. Da im Kopf von σ_1 eine existenzquantifizierte Variable vorkommt, wird g_0 allerdings zuerst um die Abbildung von ihr auf einen neuen benannten Nullwert erweitert ($g_0(y_{\mathrm{Ab}}) = \eta_{\mathrm{Ab}}$). Danach kann das entsprechende T-STUDIERENDE-Tupel zu I_0 hinzugefügt werden. Da g_0 unser einziger Trigger für σ_1 ist, suchen wir nun nach Triggern für die restlichen Integritätsbedingungen und der modifizierten Instanz I_1:

$$\begin{aligned} I_1 = \{&\text{R-STUDIERENDE}(1, \text{Fieber}, \text{Fabian}, \text{Informatik}, \text{Tel: 0123456789}),\\ &\text{R-TEILNAHME}(1, 2, 1),\\ &\text{T-STUDIERENDE}(1, \text{Fieber}, \text{Fabian}, \text{Informatik}, \eta_{\mathrm{Ab}})\}. \end{aligned}$$

Wählen wir als nächstes σ_2. Da wir zwischen dem Rumpf von σ_2 und den Tupeln von I_1 keinen Homomorphismen finden (zum R-NOTEN-Ausdruck gibt es kein passendes Tupel in I_1), gibt es auch keine Trigger, für die wir die weiteren Schritte des CHASEs ausführen müssten.

Betrachten wir nun die übrige TGD b. Zwischen dem Rumpf von b und den Tupeln von I_1 existiert ein Homomorphismus g_1 mit den Abbildungen: $g_1(x_{\mathrm{Ma}}) = 1$, $g_1(x_{\mathrm{Na}}) = \text{Fieber}$, $g_1(x_{\mathrm{Vo}}) = \text{Fabian}$, $g_1(x_{\mathrm{St}}) = \text{Informatik}$, $g_1(x_{\mathrm{Be}}) = \text{Tel: 0123456789}$, $g_1(x_{\mathrm{Te}}) = 2$, $g_1(x_{\mathrm{Mo}}) = 2$. Da keine Erweiterungen von g_1 zu einem Homomorphismus existieren, der den Kopf von b auf I_1 abbildet, handelt es sich hierbei um einen aktiven Trigger. Bei der Durchführung des CHASE-Schrittes $I_1 \xrightarrow{b, g_1} I_2$, erweitern wir I_1 um Tupel, die sich aus der Anwendung von g_1 auf den Ausdruck aus dem Kopf von b ergeben. Dabei müssen wir g_1 erst wieder um Abbildungen von den existenzquantifizierten Variablen auf neue benannte Nullwerte erweitern ($g_1(y_{\mathrm{Se}}) = \eta_{\mathrm{Se}}$, $g_1(y_{\mathrm{No}}) = \eta_{\mathrm{No}}$). Das entsprechende NOTEN-Tupel kann nun zu I_1 hinzugefügt werden. Da es keine weiteren Trigger für b und keine weitere Integritätsbedingung in $\mathcal{B}$ gibt, wird kein weiter CHASE-Schritt mehr ausgeführt und wir erhalten die modifizierte Instanz I_2:

$$\begin{aligned} I_2 = \{&\text{R-STUDIERENDE}(1, \text{Fieber}, \text{Fabian}, \text{Informatik}, \text{Tel: 0123456789}),\\ &\text{R-TEILNAHME}(1, 2, 1),\\ &\text{R-NOTEN}(1, \eta_{\mathrm{Se}}, \eta_{\mathrm{No}}),\\ &\text{T-STUDIERENDE}(1, \text{Fieber}, \text{Fabian}, \text{Informatik}, \eta_{\mathrm{Ab}})\}. \end{aligned}$$

Im zweiten Durchlauf wählen wir wieder die Integritätsbedingung σ_1 und suchen nach Triggern zwischen ihr und I_2. Wieder finden wir nur einen Homomorphismus $g_2 \equiv g_0$. Diesmal finden wir bei der Prüfung auf aktive Trigger aber eine Erweiterung von g_2 mit $g_2(y_{\mathrm{Ab}}) = \eta_{\mathrm{Ab}}$ zu einem Homomorphismus, der den T-STUDIERENDE-Ausdruck vom S-T TGD-Kopf auf das neue T-STUDIERENDE-Tupel in I_2 abbildet. Somit handelt es sich bei g_2 um keinen aktiven Trigger zu I_2. Da g_2 unser einziger Trigger für σ_1 ist, wählen wir als nächstes σ_2.

Zwischen dem Rumpf von σ_2 und den Tupeln von I_2 existiert jetzt ein Homomorphismus g_3 mit jeweils einer Abbildung zwischen den R-TEILNAHME- und R-NOTEN-Ausdrücken im Rumpf von σ_2 und den R-TEILNAHME- sowie dem neuen R-NOTEN-Tupeln aus I_2. Im Detail ergeben sich die Abbildungen $g_3(x_{\mathrm{Te}}) = 2$, $g_3(x_{\mathrm{Mo}}) = 2$, $g_3(x_{\mathrm{Ma}}) = 1$, $g_3(x_{\mathrm{Se}}) = \eta_{\mathrm{Se}}$, $g_3(x_{\mathrm{No}}) = \eta_{\mathrm{No}}$. Dabei handelt es sich um einen aktiven Trigger, da keine Erweiterungen von g_3 existieren, die den Kopf von σ_2 auf I_2 abbilden. Bei der

Durchführung des CHASE-Schrittes $I_2 \xrightarrow{\sigma_2, g_3} I_3$, erweitern wir I_2 um das T-NOTEN-Tupel, das sich aus der Anwendung von g_3 auf den Ausdruck aus dem Kopf von σ_2 ergibt.

Da g_3 unser einziger Trigger für σ_2 war, können wir nun b betrachten. Für b finden wir genau wie für σ_1 einen Trigger, der aber kein aktiver Trigger ist, da ein entsprechendes R-NOTEN-Tupel bereits in I_3 existiert. Wir können also keinen weiteren CHASE-Schritt mehr ausführen und erhalten die modifizierte Instanz I_3:

$$\begin{aligned} I_3 = \{&\text{R-STUDIERENDE}(1, \text{Fieber}, \text{Fabian}, \text{Informatik}, \text{Tel: } 0123456789),\\ &\text{R-TEILNAHME}(1, 2, 1),\\ &\text{R-NOTEN}(1, \eta_{\text{Se}}, \eta_{\text{No}}),\\ &\text{T-STUDIERENDE}(1, \text{Fieber}, \text{Fabian}, \text{Informatik}, \eta_{\text{Ab}}),\\ &\text{T-NOTEN}(2, 1, \eta_{\text{Se}}, \eta_{\text{No}})\}. \end{aligned}$$

Im dritten Durchlauf finden wir für keine Integritätsbedingung in $\mathcal{B}$ einen aktiven Trigger. Wir führen also keinen CHASE-Schritt mehr aus und unsere Instanz nach dem Durchlauf entspricht der Instanz vor dem Durchlauf. Die Instanz I_3 ist also das endgültige Ergebnis des Standard-CHASEs.

Im letzten Schritt wenden wir die Definition 2.19 des Standard-CHASE auf Instanzen für Schemaabbildungen an. Dafür schränken wir I_3, welches im Moment unter dem Schema $S \cup T$ definiert ist, auf die Relationen des Schemas T ein. Unser Endergebnis von $CHASE_{\Sigma \cup b}(I_{S_R})$ ist somit:

$$\begin{aligned} I_T = \{&\text{T-STUDIERENDE}(1, \text{Fieber}, \text{Fabian}, \text{Informatik}, \eta_{\text{Ab}}),\\ &\text{T-NOTEN}(2, 1, \eta_{\text{Se}}, \eta_{\text{No}})\}. \end{aligned}$$

2.4.3. CHASE-Varianten

Wenn wir bis jetzt vom CHASE gesprochen haben, dann meinten wir immer den Standard-CHASE (siehe Definition 2.18). Neben diesem gibt es auch weitere CHASE-Arten, welche sich leicht im Ablauf unterscheiden und somit auch unterschiedliche Eigenschaften (wie z. B. in der Terminierung) aufweisen. Der *Oblivious-CHASE* vernachlässigt z. B. die Prüfung auf aktive Trigger. Somit ist seine Anwendung um einiges performanter als der Standard-CHASE. Dafür wendet der Oblivious-CHASE die (S-T) TGDs und EGDs aber auch dann an, wenn der Kopf einer Integritätsbedingung bereits erfüllt ist. Im Falle einer (S-T) TGD würde man in ungünstigen Fällen unendlich oft neue Tupel zu einer Instanz hinzufügen. Der Oblivious-CHASE wird in zwei Varianten unterschieden. Der *skolemisierte Oblivious-CHASE* kann im Gegensatz zum *naiven Oblivious-CHASE*, mithilfe einer Skolemfunktion die Anzahl der in Instanzen eingefügten Nullwerte reduzieren. Dabei fallen die Nullwerte in sich zusammen, wenn sie von denselben allquantifizierten Variablen abhängen.

Als Beispiel seien folgende Instanz und TGD gegeben:

$$\begin{aligned} I_0 = \{&\text{R-STUDIERENDE}(2, \text{Fieber}, \text{Fabian}, \text{Informatik}, \text{Tel: } 0123456789),\\ &\text{R-TEILNAHME}(2, 2, 1)\},\\ b = &\text{R-STUDIERENDE}(x_{\text{Ma}}, x_{\text{Na}}, x_{\text{Vo}}, x_{\text{St}}, x_{\text{Be}}) \wedge \text{R-TEILNAHME}(x_{\text{Te}}, x_{\text{Mo}}, x_{\text{Ma}})\\ &\longrightarrow \exists y_{\text{Se}}, y_{\text{No}} : \text{R-NOTEN}(x_{\text{Te}}, y_{\text{Se}}, y_{\text{No}}). \end{aligned}$$

Die Anwendung des naiven Oblivious-CHASE auf I_0 mit b ergibt:

$$\begin{aligned} I_1 = \{&\text{R-STUDIERENDE}(2, \text{Fieber}, \text{Fabian}, \text{Informatik}, \text{Tel: } 0123456789),\\ &\text{R-TEILNAHME}(2, 2, 1),\\ &\text{R-NOTEN}(2, \eta_{\text{Se}_1}, \eta_{\text{No}_1})\}. \end{aligned}$$

Da eine Überprüfung auf aktive Trigger (im Gegensatz zum Standard-CHASE) entfällt, wird der naive Oblivious-CHASE erneut angewandt:

$$\begin{aligned} I_2 = \{&\text{R-STUDIERENDE}(2, \text{Fieber}, \text{Fabian}, \text{Informatik}, \text{Tel: } 0123456789),\\ &\text{R-TEILNAHME}(2, 2, 1),\\ &\text{R-NOTEN}(2, \eta_{\text{Se}_1}, \eta_{\text{No}_1}),\\ &\text{R-NOTEN}(2, \eta_{\text{Se}_2}, \eta_{\text{No}_2})\}. \end{aligned}$$

Der naive Oblivious-CHASE würde in diesem Beispiel nicht terminieren und die TGD b immer wieder auf I_i anwenden. Die Instanz würde in diesem Fall unendlich groß werden.

Der Oblivious- sowie der Standard-CHASE haben das Problem, dass die Reihenfolge der eingechaseten Abhängigkeiten nicht-deterministisch ist. Somit ist die Terminierung des Algorithmus für ein gegebenes CHASE-Objekt und eine Menge von Integritätsbedingungen [(S-T) TGDs und EGDs] unentscheidbar (bei der Einschränkung der Integritätsbedingungen auf FDs und JDs bleibt die Terminierung entscheidbar [MMS79]). Aus diesem Grund sind Bedingungen, unter denen der CHASE terminiert, immer noch Stand der Forschung. Ebenso aus diesem Grund wurde der *Core-CHASE* in [DNR08] eingeführt. Mit zwei Zwischenschritten wird für den CHASE eine universelle Lösung[5] bestimmt, falls diese existiert. In dem sogenannten *parallelen CHASE-Schritt* werden als Erstes alle aktiven Trigger parallel angewendet. Im zweiten Schritt wird dann der sogenannte *Kern* berechnet. Dabei entfallen alle Lösungen aus dem parallelen CHASE-Schritt, die von anderen Lösungen überdeckt werden. Da dieser Kern bis auf Isomorphie für ein CHASE-Objekt einzigartig ist, gilt der Core-CHASE als deterministisch. Diese Eigenschaft kommt aber mit einer so hohen Komplexität daher, sodass der Core-CHASE in der Regel nicht in Anwendungsprogrammen eingesetzt wird.

Zusammengefasst terminiert der Core-CHASE genau dann, wenn es eine universelle Lösung gibt. Dies ist für den Standard-CHASE nicht immer der Fall. Für ihn ist es wahrscheinlicher, dass eine nicht-deterministische Folge von CHASE-Schritten terminiert, als dass alle Folgen von CHASE-Schritten terminieren. Noch seltener terminiert der Oblivious-CHASE, da der Test auf aktive Trigger entfällt. Der *naive* Oblivious-CHASE terminiert dabei für eine noch geringere Anzahl von Fällen als der *skolemisierte* Oblivious-CHASE, der in manchen Fällen die Anzahl der eingefügten Tupel reduziert.
Für genaue Definitionen und Beispiele zu den CHASE-Varianten wird hier auf [DHI12, GMS12] und auf das Lehrvideo von Andreas Görres und Yvonne Düwel aus dem Modul "Neueste Entwicklungen in der Informatik" (vorhanden am DBIS-Lehrstuhl der Uni Rostock) verwiesen.

[5] Bis auf Homomorphismen eindeutig [FKMP03]

2.5. Einordnung der Grundlagen in diese Arbeit

Die Grundlagen dieser Arbeit sind als Vorbereitung des Konzepts noch nicht ausreichend. Das liegt daran, dass ein großer Teil der im Konzept verwendeten Theorie mehr zum Stand der Forschung als zu den Grundlagen zählt. Bisher haben wir das Relationenmodell, die Datenintegration, einen Teil der Schemaabbildungen und den CHASE-Algorithmus eingeführt. Es fehlen noch wichtige Definitionen bezüglich der Invertierung von Schemaabbildungen.

Das Relationenmodell bietet die Grundlage aller in dieser Arbeit verwendeten Theorien. Ohne dessen Verwendung könnten wir uns nicht auf die vorgestellte Theorie der Schemaabbildungen konzentrieren und auch nicht den Standard-CHASE für dessen Ausführung verwenden. Probleme durch mögliche Heterogenität unter den Datenquellen können wir auf die Transformation der unterschiedlichsten Datenmodelle in das Relationenmodell überführen. Lösungen dafür sind in der Theorie der Datenintegration zu finden. Die GaLVE-Technik kann auf dem Fundament der Datenintegration aufbauen und diese erweitern. So werden wir das Finden eines globalen Schemas und den passenden Abbildungen immer als bereits abgeschlossenen Prozess vor dem GaLVE-Verfahren betrachten. Der CHASE-Algorithmus ist für das GaLVE-Verfahren essenziell. Als universelles Tool für die Ausführung von Abbildungen und Anfragen sowie für das Data Cleaning ist er aus der Datenbankforschung nicht mehr wegzudenken.

Der in den Grundlagen behandelte Teil der Schemaabbildungen ebnet das Fundament, um im nächsten Kapitel die Theorie und die Algorithmen für die Invertierung von Schemaabbildungen genauer vorzustellen. Neben diesen für das Konzept wichtigen Forschungsergebnissen zur Invertierung (siehe Abschnitt 3.2) werden wir auch die zuvor betrachtete Herangehensweise an das GaLVE-Verfahren mit heterogenen Datenquellen von Georgi Straube [Str13] (Abschnitt 3.1) betrachten. Abschließend folgt die Vorstellung einer weiteren, auf der Theorie der Invertierung von Schemaabbildungen aufbauenden Arbeit, welche diese mit dem Provenance-Management verbindet (siehe Abschnitt 3.3).

3. Stand der Forschung

Im ersten Abschnitt dieses Kapitels stellen wir der Vollständigkeit halber die ursprüngliche Idee zum GaLVE-Verfahren aus [FHLM96] sowie grob das Konzepts einer möglichen Umsetzung aus [Str13] vor. Letzteres setzt die GaLVE-Technik ohne die Verwendung von Schemaabbildungen um und unterscheidet sich grundlegend von dem Ansatz, der in dieser Arbeit verfolgt wird. Im zweiten Abschnitt werden wir anhand der Forschungsarbeiten [Fag07, FKPT08, APR09, FKPT11a] den aktuellen Stand der Forschung zur Invertierung von Schemaabbildungen vorstellen. Zum Abschluss des Kapitels zeigen wir eine weitere Forschungsrichtung, die auf der Invertierung von Schemaabbildungen aufsetzt. Dafür werden wir zusammenfassend zeigen, wie in [AH18b] die Rekonstruktion von Auswertungen im Forschungsdatenmanagement mit Hilfe der Verbindung von Provenance Management und der Schemaevolutionstheorie umgesetzt werden kann.

3.1. GaLVE mit heterogenen Quellen

Erstmals beschrieben wurde die GaLVE-Technik 1996 in der Projektbeschreibung für die „Föderation von Datenbanksystemen unter Beibehaltung der lokalen Anfrageautonomie und -transparenz“ [FHLM96]. Die Motivation lag wie auch heute noch in der Anfragebeantwortung „über einem globalen Schema durch Erweiterung der lokalen Schemata“. So wollte man Anfragen und Änderungsoperationen nicht über einem globalen Schema in der zugehörigen Sprache formulieren, sondern die Sprachen des jeweiligen lokalen Systems für eine lokale Ausführung nutzen, um lokale Anwendungen nicht mehr in das globale Schema migrieren zu müssen. Abweichend von der Herangehensweise in dieser Arbeit ging man noch von einer semantisch reicheren globalen Ebene aus und wollte die heterogenen lokalen Datenbankschemata auch um fehlende semantische Ausdrückstärke erweitern. Dies entfällt in unserer Betrachtung, da wir nur Daten zwischen Schemata im Relationenmodell übertragen und somit überall die gleiche Ausdrucksstärke voraussetzen. Ebenfalls nicht Bestandteil dieser Arbeit ist die in der Projektbeschreibung formulierte Entwicklung geeigneter Transaktionskonzepte für komplexe Änderungsoperationen über dem föderierten Datenbestand. Solche und ähnliche an das Konzept dieser Arbeit anknüpfende Aufgaben werden wir im Ausblick in Kapitel 6 genauer beschreiben.

Die in [FHLM96] vorgestellte Idee wurde nach der Veröffentlichung bis zur ersten Umsetzung in [Str13] nur ein einziges Mal aufgegriffen. So wurde sie in [Con97] genutzt, um verschiedene Architekturen für föderierte Datenbanksysteme zu vergleichen. Bezeichnet wurde die Technik hier als „Mischform“ bzw. „Misch-Architektur“ der Import-/Export-Schema-Architektur, der Multidatenbank-Architektur und der 5-Ebenen-Schema-Architektur. Der Grundgedanke wurde damit beschrieben, „dass es keine wirklichen globalen Anwendungen gibt, sondern alle Anwendungen lokal auf einem der Komponentensysteme laufen“.

Das erste Konzept der GaLVE-Technik folgte erst 2013 mit [Str13][1]. Die Arbeit entwickelte ein GaLVE-Verfahren unter der Zielsetzung der Datenintegration mit heterogenen lokalen Quelldatenbanken. Für die Hin- und Rückabbildung aus und in die heterogenen Quellschemata wurde ein globales Schema im

[1] Im Artikel [SBLH14] (peer-reviewed) wird die Anwendung des Konzepts der GaLVE-Technik aus [Str13] in einem klinischen Umfeld beschrieben.

Entity-Attribute-Value-Modell (EAV) gewählt. Die Umsetzung konzentrierte sich dabei auf die Abbildungen für Schemata im Relationenmodell und dem XML-Datenmodell. Das EAV-Modell wurde als globales Schema gewählt, da es durch sein generisches Schema sehr gut für die Speicherung von verschieden strukturierten Daten in verschiedenen Datenmodellen geeignet ist. Der Nachteil des EAV-Modells ist die Notwendigkeit, Metadaten auf Daten abzubilden, was es inkompatibel mit Schemaabbildungen in Logik erster Ordnung macht. Das folgende Beispiel zeigt die Übertragung einer relationalen Datenbank in ein Datenbankschema im EAV-Modell und das damit verbundene Problem, diese Übertragung mittels S-T TGDs zu spezifizieren. Die Definition des EAV-Modells ist implizit in der Beispielbeschreibung enthalten.

Sei die Instanz I der Rostocker Uni-Datenbank (siehe Abschnitt 1.1) wie folgt gegeben:

$$\begin{aligned} I = \{&\text{R-STUDIERENDE}(1,\ \eta_{\text{Na}}\ , \text{Fabian}, \text{Informatik}, \text{Tel: } 0123456789),\\ &\text{R-STUDIERENDE}(1, \text{Fieber}, \text{Fabian}, \text{Informatik}, \text{Tel: } 0123456789),\\ &\text{R-TEILNAHME}(2, 2, 1),\\ &\text{R-NOTEN}(2, \text{SS } 21, 3.0)\}. \end{aligned}$$

Ein Datenbankschema im EAV-Modell besteht aus den gleichnamigen Relationenschemata für *Entity*, *Attribute* und *Value*. Eine *Entität* (Entity) wird als Gegenstand, Konzept oder Ereignis beschrieben, welches mittels eines generischen Primärschlüssels identifiziert wird [DN07]. In der Entität-Relation kann auch der jeweilige Name, eine passende Beschreibung oder der zugehörige Typ gespeichert werden. Für unsere Uni-Datenbank erhält jeder Studierende, jede Teilnahme und jede Note einen Eintrag in der Entität-Relation in Tabelle 3.1.

ID	Name
1	R-STUDIERENDE
2	R-STUDIERENDE
3	R-TEILNAHME
4	R-NOTEN

Tabelle 3.1.: Entität-Relation der Rostocker Uni-Datenbank

Ein *Attribut* (Attribute) des EAV-Modells beschreibt genau wie ein Attribut des Relationenmodells die Eigenschaften der Entities. Wie bei einer Entität werden die Attribut-Einträge über einen Primärschlüssel identifiziert und können noch zusätzliche Eigenschaften wie den Namen, den Datentyp oder den Wertebereich speichern. So enthält die Attribut-Relation in Tabelle 3.2 alle Attribute der in I enthaltenen Relationen zusammen mit passenden Wertebereichen, wie wir sie schon im Abschnitt 1.1 für unser allgemeines Beispiel definiert haben.

ID	Name	Wertebereich
1	Matrikelnr	$\mathbb{N}$
2	Nachname	STRING
3	Vorname	STRING
4	Studiengang	{Informatik, ITTI}
5	Bemerkung	STRING
6	Teilnahmenr	$\mathbb{N}$
7	Modulnr	$\mathbb{N}$
8	Semester	{WS 20/21, SS 21}
9	Note	{1.0, 1.3, 1.7,..., 3.7, 4.0, 5.0}

Tabelle 3.2.: Attribut-Relation der Rostocker Uni-Datenbank

Zum Schluss benötigen wir nur noch eine Wert-Relation, welche die Entitäten mit den Attribut-Einträgen und ihren entsprechenden Attributwerten enthält. Ein *Wert* (Value) referenziert jeweils die Primärschlüssel eines Entität- und eines Attribut-Eintrages und speichert die Attributwerte unter einem eigenen Primärschlüssel ab. Für die genaue Speicherung der Attributwerte gibt es verschiedene Konzepte [Str13], so kann man z. B. alle Werte als Zeichenkette speichern, solange man unter dem entsprechenden Attribut-Eintrag den richtigen Datentyp und den Wertebereich für die Rückkonvertierung gesichert hat. In unserem Beispiel speichern wir für jeden Wert (nicht für den Nullwert η_{Na}) einen entsprechenden Eintrag in der Wert-Relation in Tabelle 3.3. Der Wert mit der ID 2 gehört so z. B. zu dem R-STUDIERENDE-Tupel mit der Entität-ID 1 und entspricht wegen der Attribute-ID 3 dem Attributwert eines Vornamens.

ID	Entität-ID	Attribut-ID	Wert
1	1	1	1
2	1	3	Fabian
3	1	4	Informatik
4	1	5	Tel: 0123456789
5	2	1	1
6	2	2	Fieber
7	2	3	Fabian
8	2	4	Informatik
9	2	5	Tel: 0123456789
10	3	6	2
11	3	7	2
12	3	1	1
13	4	6	2
14	4	8	SS 21
15	4	9	3.0

Tabelle 3.3.: Wert-Relation der Rostocker Uni-Datenbank

Versuchen wir die eben beschriebene Abbildung der Rostocker Uni-Datenbank in ein Schema im EAV-Modell mittels S-T TGDs auszudrücken, stoßen wir schnell auf das angesprochene Problem der Abbildung von Metadaten auf Daten. Explizit ist es mit S-T TGDs nicht möglich, die Metadaten eines Relationenschemas wie den Namen des Schemas oder die Namen der Attribute des Schemas in die Daten der Entität- und Attribut-Relationen zu übertragen.

Neben der Inkompatibilität der Schemaabbildungen und dem EAV-Modell wurden in [Str13] noch weitere Einschränkungen der Schemaabbildungen beschrieben, die einer Umsetzung des GaLVE-Verfahrens mittels der Theorie von Schemaabbildungen im Wege stehen. So ist laut [Str13] eine Rekonstruktion der kompletten lokalen Datenbank mittels Rückabbildung aus dem globalen Schema nur möglich, wenn zuvor eine informationsverlustfreie Schemaabbildung vom lokalen Schema in das globale Schema durchgeführt wurde. Solche Schemaabbildungen, die alle Informationen in das globale Schema übertragen, erzeugen im globalen Schema sehr viele Nullwerte, da auch Attribute, die nur in einem lokalen Schema vorkommen, für jedes weitere Schema gefüllt werden müssen. Dies führt zu einem sehr hohen Speicherverbrauch, welcher mit dem EAV-Modell verhindert werden kann. Im Weiteren wird das Problem aufgeführt, dass die Existenz einer Inversen ohne Informationsverlust[2] (siehe Fagin-Inverse Abschnitt 3.2.1) zu einer Schemaabbildung nicht entscheidbar[3] ist [Fag07]. Somit sei die Invertierung laut [Str13] nicht automatisierbar, was letztendlich dazu führt, dass die Theorie der Schemaabbildungen nicht weiter für das GaLVE-Verfahren betrachtet und sich vollkommen auf das EAV-Modell konzentriert wurde. In Abschnitt 4.1 werden wir

[2] Andere Inverse wie in [FKPT08, APR09] (siehe Abschnitt 3.2.2 und 3.2.3), wurde wegen ihrer nicht verlustfreien Rückabbildung nicht näher betrachtet.

[3] Nach [FKPT08] entscheidbar mit *subset property*, siehe Definition 3.4.

erklären, warum diese Annahme nicht notwendig ist und warum wir sie in unserem GaLVE-Verfahren (siehe Kapitel 4) verwerfen werden.

Da das EAV-Modell keine relationalen Konzepte wie z. B. Schlüssel- und Fremdschlüssel-Beziehungen speichern kann, wurde in [Str13] das *Erweiterte EAV-Modell* vorgestellt. Dieses wurde so entwickelt, dass eine zusicherungsbasierte Schemaintegration [SPD92, SP94, Con97] vom Relationenmodell sowie von XML-Schemata umgesetzt werden kann. Für die Hin- und Rückabbildung von den lokalen Schemata in das globale Schema im Erweiterten EAV-Modell wurden in [Str13] eigene Abbildungsregeln definiert. Der grobe Ablauf des GaLVE-Verfahrens nach [Str13] ist in der Abbildung 3.1 dargestellt. Die Abbildung der Quelldatenbanken *S1, S2, S3* in das globale Schema T im Erweiterten EAV-Modell werden mit den in der Arbeit beschriebenen Regeln durchgeführt. Das Schema *S1** stellt das Schema nach der Anwendung der Regeln für die Rückabbildung dar. An dieser Stelle ist es wichtig anzumerken, dass sich das ursprüngliche Schema $S1$ und das neu erstellte Schema $S1*$ nicht ähneln müssen. So ist unter Umständen eine Anfragetransformation aller Anfragen an das ursprüngliche Schema $S1$ zu Anfragen an das neue Schema $S1*$ nötig. In Abbildung 3.1 stellt die Anfrage *q1** eine an *S1** angepasste Anfrage dar, die dann auf die integrierten Informationen zugreifen kann. Wie eine Anfragetransformation für das neu erzeugte Schema ablaufen könnte, wurde in [Str13] nicht beschrieben.

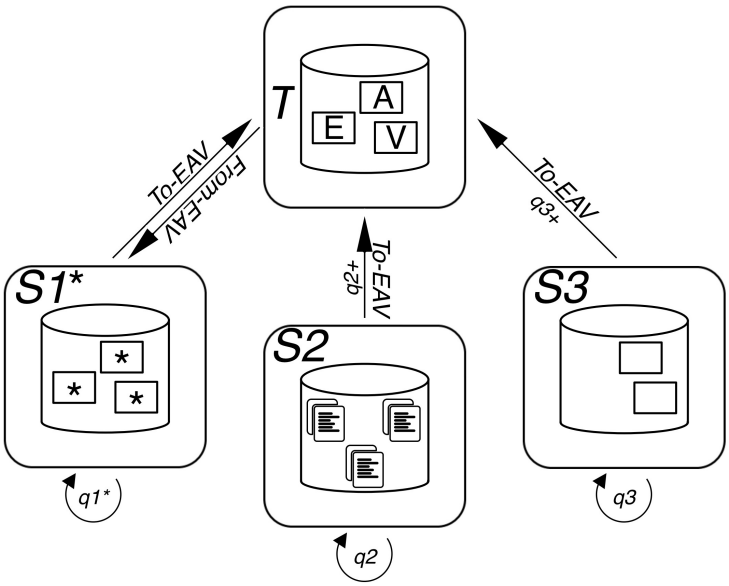

Abbildung 3.1.: Grober Ablauf des GaLVE-Konzepts nach [Str13]

3.2. Theorie der Invertierung

Die Invertierung von Schemaabbildungen ist eine fundamentale Operation für den Datenaustausch und die Schemaevolution. Zum ersten Mal formal von Fagin in [Fag07] definiert, folgten mit der Zeit eine Reihe von verschieden strengen Inversen-Definitionen mit unterschiedlichen Eigenschaften und Berechnungsvorschriften. Dieser Abschnitt fasst die wichtigsten Ergebnisse aus [Fag07, FKPT08, APR09, FKPT11a] chronologisch zusammen. Die Strukturierung der folgenden Abschnitte ist an die Arbeit [Pér13] angelehnt. In ihr werden die bisherigen Theorien zur Invertierung von Schemaabbildungen in Hinblick auf die Existenz, der Sprache und der Berechnung von Inversen verglichen. Wir werden uns im Weiteren auf die Definitionen der Inversen, ihrer Existenz und Sprache beschränken.

Die ersten Abschnitte 3.2.1 zu [Fag07], 3.2.2 zu [FKPT08] und 3.2.3 zu [APR09] unterliegen der Annahme, dass alle Quelldatenbanken nur Konstanten enthalten, während in den Zieldatenbanken auch Nullwerte erlaubt sind. Diese typische Annahme für den Datenaustausch [FKMP05] wird erst in Abschnitt 3.2.4 zu [FKPT11a] verworfen. Der Abschnitt und die darin vorgestellte Inverse wird für das Konzept dieser Arbeit von besonder Bedeutung sein.

3.2.1. Fagin-Inverse

In der Mathematik ergibt die Hintereinanderausführung einer Funktion und ihrer algebraischen Inversen die *Identität*. Aus dieser Tatsache heraus entstand für [Fag07] die Idee, in ähnlicher Weise die Inverse von Schemaabbildungen zu beschreiben. So soll die Komposition einer Abbildung mit ihrer Inversen eine Quelldatenbank unter einem Schema S in eine Kopie des Quellschemas $\hat{S} = \{\hat{R}|R \in S\}$ überführen, wobei $\hat{R}$ jeweils eine Kopie des Relationenschemas R ist. Die Menge der *Kopie*-S-T TGDs über S, die jede Relation der Quelle in das Kopie-Schema $\hat{S}$ überführt, kann wie folgt definiert werden:

$$\Sigma_{S-Kopie} = \{R(x_1, ..., x_k) \longrightarrow \hat{R}(x_1, ..., x_k)|R \text{ ist } k\text{-stelliges Relationschema in } S\}.$$

Mit Hilfe der Kopie-S-T TGDs $\Sigma_{S-Kopie}$ können wir jetzt eine Schemaabbildung $\mathcal{M}_{S-Kopie}$ von S nach $\hat{S}$ spezifizieren. Mit ihr können wir unsere erste Definition einer Inversen ableiten:

Definition 3.1. (Fagin-Inverse mit $\mathcal{M}_{S-Kopie}$, [Fag07]): Sei $\mathcal{M}$ eine Schemaabbildung von einem Quellschema S in ein Zielschema T und $\mathcal{M}'$ eine Schemaabbildung von T nach $\hat{S}$. Die Schemaabbildung $\mathcal{M}'$ ist eine *Fagin-Inverse* von $\mathcal{M}$ wenn $\mathcal{M} \circ \mathcal{M}' = \mathcal{M}_{S-Kopie}$ gilt.

Bisher bildet die inverse Schemaabbildung $\mathcal{M}'$ nur von T nach $\hat{S}$ und nicht nach S ab. Das liegt daran, dass eine S-T TGD die alle Relationen von S nach S kopiert, trivial wäre. Möchte man eine Inverse von T zurück auf S angeben, kann man daher keine S-T TGDs verwenden und muss die Schemaabbildungen mit den von ihnen definierten Instanzen ausdrücken. Seien I, J zwei Instanzen vom Schema S und $\hat{J}$ die Kopie von J in $\hat{S}$, dann ist $(I, \hat{J}) \in \mathcal{M}_{S-Kopie}$, wenn $I \subseteq J$ gilt. Die so angegebene Identitätsabbildung von S nach S

$$I\bar{d}_S = \{(I, J)|I, J \text{ sind Instanzen von } S \text{ und } I \subseteq J\}$$

kann nun dazu genutzt werden, eine Fagin-Inverse von T nach S zu definieren. Ausgehend von der folgenden Definition zur Fagin-Inversen werden wir in Bezug auf die Existenz einer Fagin-Inversen von der *Fagin-Invertierbarkeit* sprechen.

Definition 3.2. (Fagin-Inverse mit $I\bar{d}_S$, [Fag07]): Sei $\mathcal{M}$ eine Schemaabbildung von einem Quellschema S in ein Zielschema T und $\mathcal{M}'$ eine Schemaabbildung von T nach S. Die Schemaabbildung $\mathcal{M}'$ ist eine *Fagin-Inverse* von $\mathcal{M}$ wenn $\mathcal{M} \circ \mathcal{M}' = I\bar{d}_S$ gilt.

Geben wir nun ein Beispiel für eine Fagin-invertierbare Schemaabbildung mit unseren Uni-Datenbanken (siehe Abschnitt 1.1) an. Betrachten wir dafür die Schemaabbildung $\mathcal{M} = (S_K, T, \Sigma)$ der Studierenden der Kieler Uni-Datenbank im Schema S_K in das uns bekannte globale Schema T. Sei Σ durch die S-T TGD σ spezifiziert, dann ist es einfach zuerkennen, dass die zu $\mathcal{M}$ inverse Abbildung $\mathcal{M}' = (T, S_K, \Sigma')$ mit der S-T TGD σ' spezifiziert werden kann:

$$\sigma = \text{K-STUDIERENDE}(x_{\text{Ma}}, x_{\text{Na}}, x_{\text{Vo}}, x_{\text{Ab}}) \longrightarrow \exists y_{\text{St}} : \text{T-STUDIERENDE}(x_{\text{Ma}}, x_{\text{Na}}, x_{\text{Vo}}, y_{\text{St}}, x_{\text{Ab}}),$$
$$\sigma' = \text{T-STUDIERENDE}(x_{\text{Ma}}, x_{\text{Na}}, x_{\text{Vo}}, x_{\text{St}}, x_{\text{Ab}}) \longrightarrow \text{K-STUDIERENDE}(x_{\text{Ma}}, x_{\text{Na}}, x_{\text{Vo}}, x_{\text{Ab}}).$$

Die inverse Abbildung σ' in unserem Beispiel ist dadurch entstanden, dass wir die Pfeilrichtung von σ vertauscht haben. Im Allgemeinen kann eine Fagin-Inverse nicht so einfach erzeugt werden. Um das und auch alle kommenden Eigenschaften der folgenden Inversen einfach zeigen zu können, werden wir von dem Uni-Datenbank-Beispiel abkehren und abstraktere Beispiele verwenden. Das folgende Beispiel stammt aus [Fag07] und wurde ebenso in [Pér13] verwendet, um zu zeigen, dass die Umkehrung der Pfeilrichtungen nicht immer eine Fagin-Inverse erzeugt.

Sei eine Schemaabbildung $\mathcal{M}$ vom Quellschema $S = \{A = \{a\}, B = \{b\}\}$ zum Zielschema $T = \{P = \{p\}, Q = \{q\}, R = \{r\}\}$ durch folgende S-T TGDs spezifiziert ([Fag07, Pér13]):

$$A(x) \longrightarrow P(x),$$
$$A(x) \longrightarrow Q(x),$$
$$B(x) \longrightarrow R(x),$$
$$B(x) \longrightarrow Q(x).$$

Würde man nun eine Schemaabbildung $\mathcal{M}'$ dadurch erstellen, dass man einfach nur die Pfeilrichtungen der S-T TGDs in $\mathcal{M}$ vertauscht, so würde diese durch die S-T TGDs $P(x) \longrightarrow A(x)$, $Q(x) \longrightarrow A(x)$, $R(x) \longrightarrow B(x)$ und $Q(x) \longrightarrow B(x)$ spezifiziert sein. In der Hinabbildung mit $\mathcal{M}$ vereinigen wir mit den S-T TGDs $A(x) \longrightarrow Q(x)$ und $B(x) \longrightarrow Q(x)$ die Daten aus A und B in Q. Mit den entsprechenden umgedrehten S-T TGDs erzeugen wir bei der Rückabbildung für einen Wert in Q jeweils einen Wert in A und in B. Wir erzeugen also auch Werte in B, wenn der Wert in Q aus A stammt und umgekehrt. Betrachten wir beispielsweise die Instanz $I = \{A(1)\}$. Mit der Schemaabbildung $\mathcal{M}$ würden wir die Instanz $K = \{P(1), Q(1)\}$ erzeugen. Bilden wir diese mit der vermeintlichen Inversen $\mathcal{M}'$ ab, erhalten wir die Instanz $I = \{A(1), B(1)\}$. Aus nur einem Tupel für A erzeugen wir durch die Hin- und Rückabbildung mit $\mathcal{M}$ und $\mathcal{M}'$ zwei Tupel (jeweils eines für A und eines für B). Dieses Verhalten gilt für jede Instanz $K \in Sol_{\mathcal{M}}(I)$ und jede Instanz $J \in Sol_{\mathcal{M}'}(K)$, woraus folgt dass $(I, I) \notin \mathcal{M} \circ \mathcal{M}'$ und $\mathcal{M} \circ \mathcal{M}' \neq \bar{Id}_S$ ist. Die Schemaabbildung mit den vertauschten Pfeilrichtungen $\mathcal{M}'$ kann somit keine Inverse von $\mathcal{M}$ sein.

Dass $\mathcal{M}'$ keine Fagin-Inverse ist, liegt nicht daran, dass $\mathcal{M}$ nicht Fagin-invertierbar ist. Das ist anhand des folgenden Beispiels zu erkennen. Sei die Schemaabbildung $\mathcal{M}''$ von T nach S durch die S-T TGDs $P(x) \longrightarrow A(x), R(x) \longrightarrow B(x)$ spezifiziert, dann kann gezeigt werden, dass diese eine Fagin-Inverse von $\mathcal{M}$ ist. Auch ohne den Beweis sieht man, dass wir dadurch, dass wir Q nicht in der Rückabbildung nutzen, sondern unsere Daten nur aus P und R rückabbilden, auch nur die Daten erhalten, die auch vorher in der Quelle existierten.

Sprache der Fagin-Inversen Für die Darstellung einer Fagin-Inversen sind die S-T TGDs, so wie wir sie kennengelernt haben (mit Konjunktionen von atomaren Ausdrücken über Relationenschemata im Rumpf und Kopf, siehe Definition 2.7) nicht ausreichend. Das folgende Beispiel aus [FKPT08] zeigt, dass die S-T TGDs in manchen inversen Schemaabbildungen auch Ungleichheiten enthalten müssen. Sei das Quellschema $S = \{A = \{a_1, a_2\}, B = \{b\}\})$ und das Zielschema $T = \{P = \{p_1, p_2\}, Q = \{q\}, R = \{r\}\}$ gegeben. Eine Schemaabbildung $\mathcal{M}$ von S nach T sei durch folgende S-T TGDs spezifiziert:

$$
\begin{aligned}
&A(x_1, x_2) \longrightarrow P(x_1, x_2),\\
&B(x) \longrightarrow P(x, x),\\
&B(x) \longrightarrow Q(x),\\
&A(x, x) \longrightarrow R(x).
\end{aligned}
$$

Ähnlich wie im vorherigen Beispiel werden hier wieder Daten aus A und B vereinigt. Dieses Mal kann die Vereinigung nur mithilfe einer Ungleichheit aufgelöst werden. Die Werte in B können aus Q rekonstruiert werden, während die Werte in A für die a_1 und a_2 den gleichen Attributwert haben, aus R zurückgewonnen werden können. Für die Werte in A, die nicht aus den gleichen Attributwerten für a_1 und a_2 bestehen, muss man auf P zurückgreifen. Um die Werte in P auszuschließen, die aus B generiert wurden, müssen wir die Ungleichheit der beiden Attribute a_1 und a_2 fordern. So kann man mit der Schemaabbildung $\mathcal{M}'$ eine Fagin-Inverse von $\mathcal{M}$ angeben, die durch folgende S-T TGDs spezifiziert ist:

$$
\begin{aligned}
&P(x_1, x_2) \wedge (x_1 \neq x_2) \longrightarrow A(x_1, x_2),\\
&Q(x) \longrightarrow B(x),\\
&R(x) \longrightarrow A(x, x).
\end{aligned}
$$

Neben den Ungleichheiten werden für die Darstellung von Fagin-Inversen noch Prädikate $C(x)$ im Rumpf der inversen Abbildungen benötigt. Das Prädikat $C(x)$ bestimmt, dass x eine Konstante ist. Eine Schemaabbildung, die dieses Prädikat benötigt, wäre nach [Pér13] z. B. $\mathcal{M}$, spezifiziert durch die S-T TGD $A(x_1, x_2) \longrightarrow \exists y : (P(x_1, y) \wedge P(y, x_2))$. Eine Fagin-Inverse zu $\mathcal{M}$ könnte die Schemaabbildung $\mathcal{M}'$, spezifiziert mit der S-T TGD $P(x_1, x_3) \wedge P(x_3, x_2) \wedge C(x_1) \wedge C(x_2) \longrightarrow A(x_1, x_2)$ sein. Sei $\mathcal{M}''$ eine Schemaabbildung spezifiziert durch $P(x_1, x_3) \wedge P(x_3, x_2) \longrightarrow A(x_1, x_2)$, der S-T TGD aus $\mathcal{M}'$ ohne das Prädikat $C(x)$, dann lässt sich leicht zeigen, dass diese keine Fagin-Inverse zu $\mathcal{M}$ ist. Betrachten wir die Instanz $I = \{A(1, 2), A(2, 1)\}$ und bilden sie mit der Schemaabbildung $\mathcal{M}$ in die Instanz $K = \{P(1, \eta_1), P(\eta_1, 1), P(2, \eta_2), P(\eta_2, 2)\}$ ab. Die Rückabbildung von K mit der Schemaabbildung $\mathcal{M}'$ würden wieder I ergeben, sodass $(I, I) \in \mathcal{M} \circ \mathcal{M}'$ ist. Bilden wir K mit der Schemaabbildung $\mathcal{M}''$ ab, erhalten wir eine Instanz, die auch die Tupel $A(\eta_1, \eta_2)$ und $A(\eta_2, \eta_1)$ enthält. Mit I haben wir also eine Instanz gefunden, für die $(I, I) \notin \mathcal{M} \circ \mathcal{M}''$ gilt, was bedeutet, dass $\mathcal{M}''$ keine Fagin-Inverse sein kann.

In [Fag07] wird bewiesen, dass jede Fagin-invertierbare Schemaabbildung $\mathcal{M} = (S, T, \Sigma)$ eine Fagin-Inverse von T nach S besitzt, die durch S-T TGDs mit Ungleichheiten und Prädikaten $C(x)$ im Rumpf spezifiziert ist. Darauf aufbauend wird in [FKPT08] bewiesen, dass sowohl die Ungleichheiten als auch das Prädikate $C(x)$ notwendig für die Fagin-Inverse sind.

Existenz der Fagin-Inversen Bisher haben wir nur Schemaabbildungen betrachtet, die eine Fagin-Inverse besaßen. Wir haben aber noch nicht geklärt, wann eine Schemaabbildung überhaupt Fagin-invertierbar ist. In [Fag07] wird die Existenz einer Fagin-Inverse an Schemaabbildungen ohne Informationsverlust geknüpft. Verlorene Informationen können dabei in vielen verschiedenen Arten auftreten.

Betrachten wir beispielsweise folgende Schemaabbildungen aus [Pér13] (das Quell- und Zielschema sei implizit in den S-T TGDs gegeben):

$$\begin{aligned}\mathcal{M}_1 :& A(x_1, x_2) \longrightarrow P(x_1),\\ \mathcal{M}_2 :& A(x_1, x_2) \longrightarrow P(x_1) \wedge Q(x_2),\\ \mathcal{M}_3 :& A(x) \longrightarrow P(x),\\ & B(x) \longrightarrow P(x).\end{aligned}$$

Die Schemaabbildung $\mathcal{M}_1$ führt mittels der S-T TGD eine Projektion aus. Das Attribut, das herausprojiziert wird, ist nach der Abbildung nicht mehr rekonstruierbar und somit eine verlorene Information. In $\mathcal{M}_2$ werden zwar beide Attribute aus A übernommen, die Beziehung der beiden Attribute innerhalb von A geht allerdings durch die nicht verbundtreue Dekomposition in die unterschiedlichen Relationen verloren. Ein ähnlicher Fall ist in $\mathcal{M}_3$ zu finden. Durch die Vereinigung der Attribute aus A und B in P kann die genauer Herkunft dieser nicht mehr rekonstruiert werden.

Für den formellen Nachweis, ob eine Schemaabbildung Fagin-invertierbar ist, benötigen wir die folgende Definition der *unique-solutions property*:

Definition 3.3. (unique-solutions property, [Fag07]): Eine Schemaabbildung $\mathcal{M}$ von einem Quellschema S in ein Zielschema T genügt der *unique-solutions property*, wenn für jedes Paar von Instanzen I_1, I_2 von S gilt, dass $Sol_{\mathcal{M}}(I_1) = Sol_{\mathcal{M}}(I_2)$ auch $I_1 = I_2$ impliziert.

Genügt eine Schemaabbildung der unique-solutions property, dann hat sie eine Fagin-Inverse [Fag07]. Diese Bedingung ist im Allgemeinen nur notwendig [FKPT08]. Nur eingeschränkt auf die Teilklasse der LAV-Abbildungen, ist die unique-solutions property eine hinreichende Bedingung für die Fagin-Invertierbarkeit [Fag07]. Mit ihrer Hilfe können wir nun einfache Gegenbeispiele der Fagin-Invertierbarkeit aus [Pér13] betrachten.

Seien für die Schemaabbildung $\mathcal{M}_1$ (siehe Beispiel für Informationsverlust) die Instanzen $I_1 = \{A(1,2)\}$, $I_2 = \{A(1,3)\}$ gegeben. Da $Sol_{\mathcal{M}_1}(I_1) = Sol_{\mathcal{M}_1}(I_2)$ gilt, aber $I_1 \neq I_2$ ist, verletzt $\mathcal{M}_1$ die unique-solutions property und hat keine Fagin-Inverse. Auch für die Schemaabbildung $\mathcal{M}_2$ können wir mit den Instanzen $I_1 = \{A(1,2), A(3,4)\}$ und $I_2 = \{A(1,4), A(3,2)\}$ ein einfaches Beispiel angeben, für das zwar $Sol_{\mathcal{M}}(I_1) = Sol_{\mathcal{M}}(I_2)$ gilt, aber wieder $I_1 \neq I_2$ ist. Gleiches gilt auch für $\mathcal{M}_3$ und dem Instanzpaar $I_1 = \{A(1)\}$, $I_2 = \{B(1)\}$.

Da die unique-solutions property als notwendige Bedingung keine positiven Nachweise der Fagin-Invertierbarkeit zulässt, wurde in [FKPT08] die *subset property* als hinreichende Bedingung für die Existenz einer Fagin-Inversen eingeführt. So ist eine Schemaabbildung genau dann Fagin-invertierbar, wenn sie der subset property genügt [FKPT08].

Definition 3.4. (subset property, [FKPT08]): Eine Schemaabbildung $\mathcal{M}$ von einem Quellschema S in ein Zielschema T genügt der *subset property*, wenn für jedes Paar von Instanzen I_1, I_2 von S gilt, dass $Sol_{\mathcal{M}}(I_1) \subseteq Sol_{\mathcal{M}}(I_2)$ auch $I_2 \subseteq I_1$ impliziert.

Im nächsten Unterabschnitt werden wir uns einer relaxteren Inversen-Definition widmen, die auch Schemaabbildungen mit Informationsverlust zulässt. Mit ihr werden wir dann inverse Schemaabbildungen für $\mathcal{M}_1$, $\mathcal{M}_2$ und $\mathcal{M}_3$ angeben können.

3.2.2. Quasi-Inverse

Betrachten wir noch einmal unsere Schemaabbildung $\mathcal{M}_1 : A(x_1, x_2) \longrightarrow P(x_1)$ aus dem vorherigen Abschnitt zusammen mit der Instanz $I = \{A(1, 2)\}$. Wir haben bereits geklärt, dass $\mathcal{M}_1$ eine Schemaabbildung mit Informationsverlust darstellt, nicht die unique-solutions property erfüllt und somit auch nicht Fagin-invertierbar ist. Sei nun eine Schemaabbildung $\mathcal{M}'_1$ spezifiziert durch die S-T TGD $P(x) \longrightarrow \exists y : A(x, y)$. Bilden wir I mit $\mathcal{M}_1$ ab, erhalten wir die Instanz $K = \{P(1)\}$. Bilden wir K mit $\mathcal{M}'_1$ ab, erhalten wir $I' = \{A(1, \eta)\}$. Würden wir I' erneut mit $\mathcal{M}_1$ abbilden, erhalten wir wieder K. Unsere Schemaabbildung $\mathcal{M}'_1$ liefert uns mit I' also eine *datenaustauschäquivalente* Instanz zu I unter $\mathcal{M}_1$, sprich eine Instanz, die für $\mathcal{M}_1$ das gleiche Ergebnis liefert. Die Schemaabbildung $\mathcal{M}_1$ spannt demnach eine Äquivalenzrelation $\sim_{\mathcal{M}_1}$ auf, welche wir allgemein wie folgt definieren können: Sei $\mathcal{M}$ eine Schemaabbildung von S nach T, dann gilt $I_1 \sim_{\mathcal{M}} I_2$ für zwei Instanzen aus S, wenn $Sol_{\mathcal{M}}(I_1) = Sol_{\mathcal{M}}(I_2)$. In unserem Beispiel gilt $I \sim_{\mathcal{M}_1} I'$.

Auf Basis der Äquivalenzrelation $\sim_{\mathcal{M}}$ kann man nun eine *Quasi-Inverse* definieren, bei der nicht zwischen Quellinstanzen unterschieden wird, die äquivalent für den Datenaustausch sind [FKPT08, Pér13]. Zuvor müssen wir noch die Schreibweise der Relation $D[\sim_{\mathcal{M}}]$ definieren. Sei $\mathcal{D}$ eine beliebige Schemaabbildung[4] von S nach S und $\mathcal{M}$ eine Schemaabbildung von S nach T, dann ist die Relation $D[\sim_{\mathcal{M}}]$ definiert mit $D[\sim_{\mathcal{M}}] = \{(I_1, I_2) | \exists I'_1, I'_2 : I_1 \sim_{\mathcal{M}} I'_2, I_2 \sim_{\mathcal{M}} I'_2, (I'_1, I'_2) \in D\}$ [Pér13]. Mit der Hilfe von $D[\sim_{\mathcal{M}}]$ definieren wir nun die Quasi-Inverse wie eine Fagin-Inverse modulo $\sim_{\mathcal{M}}$ [FKPT08, Pér13]:

Definition 3.5. (Quasi-Inverse, [FKPT08, Pér13]): Sei $\mathcal{M}$ eine Schemaabbildung von einem Quellschema S in ein Zielschema T und $\mathcal{M}'$ eine Schemaabbildung von T nach S. Die Schemaabbildung $\mathcal{M}'$ ist eine *Quasi-Inverse* von $\mathcal{M}$, wenn $(\mathcal{M} \circ \mathcal{M}')[\sim_{\mathcal{M}}] = I\bar{d}_S[\sim_{\mathcal{M}}]$ gilt.

Wie nennen Schemaabbildungen, für die eine Quasi-Inverse existiert *Quasi-invertierbar*. Die Quasi-Inverse ist eine Verallgemeinerung der Fagin-Inversen. In [FKPT08] wurde bewiesen, dass eine Fagin-invertierbare Schemaabbildung $\mathcal{M}$ von S nach T genau dann die Fagin-Inverse $\mathcal{M}'$ hat, wenn $\mathcal{M}'$ auch eine Quasi-Inverse von $\mathcal{M}$ ist.

Sprache der Quasi-Inversen Im vorherigen Abschnitt haben wir gezeigt, dass die drei Schemaabbildungen $\mathcal{M}_1 : A(x_1, x_2) \longrightarrow P(x_1)$, $\mathcal{M}_2 : A(x_1, x_2) \longrightarrow P(x_1) \wedge Q(x_2)$ und $\mathcal{M}_3 : A(x) \longrightarrow P(x)$, $B(x) \longrightarrow P(x)$ keine Fagin-Inverse haben. Alle drei Schemaabbildungen haben allerdings eine Quasi-Inverse. Es kann gezeigt werden, dass die Inversen von $\mathcal{M}_1$, $\mathcal{M}_2$ und $\mathcal{M}_3$ die entsprechenden Schemaabbildungen $\mathcal{M}'_1 : P(x) \longrightarrow \exists y : A(x, y)$, $\mathcal{M}'_2 : P(x_1) \wedge Q(x_2) \longrightarrow \exists y_1, y_2 : A(x_1, y_1) \wedge A(y_2, x_2)$ und $\mathcal{M}'_3 : P(x) \longrightarrow A(x) \vee B(x)$ sind.

Wir haben schon festgestellt, dass Fagin-Inverse auch Quasi-Inverse sind, so benötigen wir zum Ausdruck einer Quasi-Inversen auch mindestens Ungleichheiten und Konstanten-Prädikate $C(x)$ im Rumpf der S-T TGDs. Die Quasi-Inverse $\mathcal{M}'_3$ ist ein Beispiel dafür, dass wir noch zusätzlich Disjunktionen im Kopf der S-T TGDs benötigen. Laut [FKPT08] gilt, dass eine Quasi-invertierbare Schemaabbildung $\mathcal{M} = (S, T, \Sigma)$ eine Quasi-Inverse von T nach S besitzt, die durch S-T TGDs mit Ungleichheiten und Prädikaten $C(x)$ im Rumpf sowie Disjunktionen im Kopf spezifiziert ist. In [FKPT08] wird ebenfalls bewiesen, dass die Disjunktionen notwendig für die Quasi-Inversen sind sowie dass Quasi- und Fagin-Inverse auch durch S-T TGDs mit Gleichheiten und Disjunktionen im Kopf und dem Prädikat $C(x)$ im Rumpf spezifiziert werden können.

[4] Das ist dasselbe wie eine binäre Funktion über den Instanzen des Quellschemas S [Pér13].

Existenz der Quasi-Inversen Bisher konnten wir für alle nicht Fagin-invertierbaren Schemaabbildungen Quasi-Inverse angeben. Das ist nicht immer möglich. Auch die Existenz einer Quasi-Inversen hängt an einer Bedingung. So ist eine Schemaabbildung $\mathcal{M}$ nur genau dann Quasi-invertierbar, wenn sie der $[\sim_{\mathcal{M}}]$-*subset property* genügt [FKPT08].

Definition 3.6. ($[\sim_{\mathcal{M}}]$-subset property, [FKPT08]): Eine Schemaabbildung $\mathcal{M}$ von einem Quellschema S in ein Zielschema T genügt der $[\sim_{\mathcal{M}}]$-*subset property*, wenn für jedes Paar von Instanzen I_1, I_2 von S gilt, dass $Sol_{\mathcal{M}}(I_1) \subseteq Sol_{\mathcal{M}}(I_2)$ impliziert, dass Instanzen I_1', I_2' existieren, für die $I_1 \sim_{\mathcal{M}} I_1'$, $I_2 \sim_{\mathcal{M}} I_2'$ und $I_2' \subseteq I_1'$ gilt.

Ein Beispiel, für das keine Quasi-Inverse existiert, ist in [FKPT08] mit der Schemaabbildung $\mathcal{M}$ von $S = \{A = \{a_1, a_2\}$ nach $T = \{P = \{p_1, p_2\}, Q = \{q\})\}$ gegeben, die durch die S-T TGD

$$A(x_1, x_3) \wedge A(x_3, x_2) \longrightarrow P(x_1, x_2) \wedge Q(x_3)$$

spezifiziert wird. In [FKPT08] wurde gezeigt, dass $\mathcal{M}$ nicht der $[\sim_{\mathcal{M}}]$-subset property genügt. Ein Gegenbeispiel ist auch in [Pér13] aufgeführt.

Die Quasi-Inverse ist eine sinnvolle Erweiterung der Inversen-Theorie. Mit ihr können wir auch Abbildungen mit Informationsverlust invertieren, solange das Ergebnis äquivalent für den Datenaustausch ist. Es gibt dennoch viele Schemaabbildungen, die nicht Quasi-invertierbar sind. Im nächsten Abschnitt werden wir einen Formalismus kennenlernen, der auch die nicht Quasi-invertierbaren Abbildungen umfasst.

3.2.3. Recovery und Maximum Recovery

Die *Recovery* und *Maximum Recovery* wurden in [APR09] als Alternative zur Fagin- und Quasi-Inverse vorgestellt. Die Maximum Recovery ist eine Verallgemeinerung der Fagin-Inverse, mit dem Vorteil, dass sie für jede durch S-T TGDs spezifizierte Schemaabbildung existiert. Betrachten wir erneut das Gegenbeispiel der Quasi-Inversen aus dem letzten Abschnitt $\mathcal{M} : A(x_1, x_3) \wedge A(x_3, x_2) \longrightarrow P(x_1, x_2) \wedge Q(x_3)$. In [Pér13] wurde für $\mathcal{M}$ eine Rückabbildung $\mathcal{M}'$ angeben, die spezifiziert mit folgenden S-T TGDs, bestmöglich die abgebildeten Daten wiedergewinnen kann:

$$P(x_1, x_2) \longrightarrow \exists y : A(x_1, y) \wedge A(y, x_2),$$
$$Q(x) \longrightarrow \exists y_1, y_2 : A(y_1, x) \wedge A(x, y_2).$$

Diese bestmögliche Rückabbildung von gültigen Daten bzw. der größtmöglichen Menge gültiger Daten in Bezug auf $\mathcal{M}$ ist dabei auch der Grundgedanke hinter der Recovery und der Maximum Recovery. Formal nennen wir eine Abbildung $\mathcal{M}'$ von T nach S Recovery der Schemaabbildung $\mathcal{M}$ von S nach T, wenn sich jede Instanz I von S selbst im Lösungsraum unter $\mathcal{M} \circ \mathcal{M}'$ enthält.

Definition 3.7. (Recovery, [APR09]): Sei $\mathcal{M}$ eine Schemaabbildung von einem Quellschema S in ein Zielschema T und $\mathcal{M}'$ eine Schemaabbildung von T nach S. Die Schemaabbildung $\mathcal{M}'$ ist eine *Recovery* von $\mathcal{M}$, wenn $(I, I) \in \mathcal{M} \circ \mathcal{M}'$ für jede Instanz I von S gilt.

Für unser zuvor aufgeführtes Beispiel $\mathcal{M} : A(x_1, x_3) \wedge A(x_3, x_2) \longrightarrow P(x_1, x_2) \wedge Q(x_3)$ wird in [Pér13] gezeigt, dass eine Schemaabbildung $\mathcal{M}_1' : P(x_1, x_2) \longrightarrow \exists y : A(x_1, y) \wedge A(y, x_2)$ eine Recovery für $\mathcal{M}$ ist. Gleiches gilt für die Schemaabbildung $\mathcal{M}_2' : Q(x) \longrightarrow \exists y_1, y_2 : A(y_1, x) \wedge A(x, y_2)$. Beide Schemaabbildungen erlauben jeweils eine Wiederherstellung der Quelldatenbanken mit gültigen Daten.

Anders sieht es mit der Schemaabbildung $\mathcal{M}'_3 : P(x_1, x_2) \wedge Q(x_3) \longrightarrow A(x_1, x_3) \wedge A(x_3, x_2)$ aus. Mit $\mathcal{M}'_3$ wird nicht gewährleistet, dass die Rückabbildung eine gültige Wiederherstellung ist, weshalb sie keine Recovery von $\mathcal{M}$ ist. Das zeigt auch folgendes Beispiel. Sei die Instanz $I = \{A(1,2), A(2,1)\}$ gegeben, dann enthält die Instanz K, die durch die Abbildung von I mit $\mathcal{M}$ entsteht, die Tupel $P(1,1), P(2,2), Q(1), Q(2)$. Bildet man K mit $\mathcal{M}'_3$ erneut nach S ab, enthält man $J = \{A(1,2), A(2,1),\ A(1,1), A(2,2)\}$, woraus $(I, I) \notin \mathcal{M} \circ \mathcal{M}_3$ folgt. Das verletzt die Bedingung einer Recovery.

Für das letzte Beispiel haben wir gesehen, dass es mehrere Recoveries $\mathcal{M}'$ für eine Schemaabbildung $\mathcal{M}$ geben kann. Die Recoveries bilden dabei eine Ordnung über der Größe des von $\mathcal{M} \circ \mathcal{M}'$ aufgespannten Lösungsraumes. Es gilt, dass umso mehr Informationen wiederhergestellt werden können, je kleiner der Lösungsraum ist. Seien $\mathcal{M}'_1$ und $\mathcal{M}'_2$ zwei beliebige Recoveries von $\mathcal{M}$ und gelte $\mathcal{M} \circ \mathcal{M}'_2 \subseteq \mathcal{M} \circ \mathcal{M}'_1$, dann sagen wir, dass $\mathcal{M}'_2$ eine informativere Recovery als $\mathcal{M}'_1$ ist, da der Lösungsraum von $\mathcal{M}'_2$ kleiner ist als der von $\mathcal{M}'_1$ [Pér13]. Mit dieser Formulierung können wir die Maximum Recovery als Recovery definieren, die die meisten gültigen Informationen unter allen Recoverys von $\mathcal{M}$ zurückliefert.

Definition 3.8. (Maximum Recovery, [APR09]): Sei $\mathcal{M}$ eine Schemaabbildung von einem Quellschema S in ein Zielschema T und $\mathcal{M}'$ eine Schemaabbildung von T nach S. Die Schemaabbildung $\mathcal{M}'$ ist eine *Maximum Recovery* von $\mathcal{M}$, wenn sie eine Recovery von $\mathcal{M}$ ist und für jede Recovery $\mathcal{M}''$ von $\mathcal{M}$ gilt, dass $\mathcal{M} \circ \mathcal{M}' \subseteq \mathcal{M} \circ \mathcal{M}''$ ist.

In [APR09] wurde bewiesen, dass eine Schemaabbildung $\mathcal{M}'$ von T nach S genau dann eine Maximum Recovery einer Schemaabbildung $\mathcal{M}$ von S nach T ist, wenn $\mathcal{M} = \mathcal{M} \circ \mathcal{M}' \circ \mathcal{M}$ gilt. In [Pér13] wird mit dieser Bedingung bewiesen, dass für unsere Schemaabbildung $\mathcal{M} : A(x_1, x_3) \wedge A(x_3, x_2) \longrightarrow P(x_1, x_2) \wedge Q(x_3)$ die zuvor intuitiv angegebene Schemaabbildung $\mathcal{M}'$, spezifiziert durch die S-T TGDs:

$$P(x_1, x_2) \longrightarrow \exists y : A(x_1, y) \wedge A(y, x_2),$$
$$Q(x) \longrightarrow \exists y_1, y_2 : A(y_1, x) \wedge A(x, y_2)$$

wirklich die Maximum Recovery von $\mathcal{M}$ ist.

Wir haben bereits beschrieben, dass die Maximum Recovery die Fagin-Inverse verallgemeinert. Aus diesem Grund kann man genau dann für eine durch S-T TGDs spezifizierte Fagin-invertierbare Schemaabbildung $\mathcal{M}$ eine Fagin-Inverse $\mathcal{M}'$ angeben, wenn $\mathcal{M}'$ auch eine Maximum Recovery von $\mathcal{M}$ ist [APR09]. Für durch S-T TGDs spezifizierte Quasi-invertierbare Schemaabbildung $\mathcal{M}$ gilt, dass die Schemaabbildung $\mathcal{M}'$ dann eine Quasi-Inverse von $\mathcal{M}$ ist, wenn $\mathcal{M}'$ eine Maximum Recovery ist. Es gilt außerdem, dass wenn $\mathcal{M}'$ eine Quasi-Inverse und eine Recovery von $\mathcal{M}$ ist, $\mathcal{M}'$ auch eine Maximum Recovery von $\mathcal{M}$ ist [APR09].

Sprache der Maximum Recovery Genau wie bei den Quasi-Inversen werden für eine Recovery oder Maximum Recovery für die Rückabbildung von durch S-T TGD spezifizierten Schemaabbildungen S-T TGDs mit Gleichheiten und Disjunktionen im Kopf sowie dem Prädikat $C(x)$ im Rumpf bzw. S-T TGDs mit Disjunktionen im Kopf sowie Ungleichheiten und Prädikaten $C(x)$ im Rumpf benötigt. Die Notwendigkeit der Disjunktion wird in [Pér11] bewiesen, die Notwendigkeit vom Prädikat $C(x)$ in [APR09].

Existenz der Maximum Recovery Eine Recovery sowie Maximum Recovery existiert für jede durch S-T TGDs spezifizierte Schemaabbildung [APR09]. Hier ist es wichtig zu erwähnen, dass diese Eigenschaft nur für durch S-T TGDs spezifizierte Schemaabbildung gilt. So können wir die Definition der Maximum Recovery im Gegensatz zur Fagin- oder Quasi-Inversen auch auf allgemeinere Schemaabbildungen anwenden, für diese ist dann aber nicht klar, dass immer eine Recovery und Maximum Recovery existiert.

3.2.4. Maximum Extended Recovery

Mit der Maximum Recovery haben wir im vorherigen Abschnitt eine Inverse kennengelernt, die für jede durch S-T TGDs spezifizierte Schemaabbildung existiert. Leider wurde sie, genau wie die Fagin- und Quasi-Inverse unter der für den Datenaustausch typischen Annahme definiert, dass die Quelldatenbank einer Abbildung nur Konstanten beinhalten darf. Nullwerte können erst im Zielschema entstehen. Viele der Hauptaussagen zu den bisher vorgestellten Inversen konnten nur unter dieser Annahme getroffen werden. Da inverse Schemaabbildungen auf Schemaabbildungen folgen, welche aber Nullwerte erzeugen können, vermindert diese Annahme die Anwendbarkeit aller bisher betrachteten Inversen [FKPT11a]. Als Lösung für diesen Widerspruch wurde in [FKPT11a] die *Maximum Extended Recovery* eingeführt. Sie existiert ebenso für alle durch S-T TGDs spezifizierten Schemaabbildungen und erlaubt gleichzeitig Nullwerte in den Quelldatenbanken.

Um die Nullwerte in der Quelldatenbank bei einer Schemaabbildung erlauben zu können, muss man die Schemaabbildung unter einem Homomorphismus betrachten. Der Homomorphismus ermöglicht es dann zwischen Werten und Nullwerten differenzieren zu können.

Definition 3.9. (Homomorphe Erweiterung, [FKPT11a]): Sei $\mathcal{M}$ eine Schemaabbildung. Die *homomorphe Erweiterung* $e(\mathcal{M})$ wird definiert mit:

$$e(\mathcal{M}) = \{(I, J) | \exists I', J', h, g : (I', J') \in \mathcal{M} \wedge h : I \longrightarrow I' \wedge g : J \longrightarrow J'\}.$$

Wenn eine Quellinstanz Nullwerte enthält, kann man für eine Schemaabbildung $\mathcal{M}$ nun deren homomorphe Erweiterung $e(\mathcal{M})$ benutzen. In [FKPT11a] wurde gezeigt, dass für jede durch S-T TGDs spezifizierte Schemaabbildung $\mathcal{M}$, bei der Nullwerte in der Quell- und Zielinstanz vorkommen können, die homomorphe Erweiterung $e(\mathcal{M})$ eine Maximum Recovery hat. So können wir nun eine Extended Recovery und Maximum Extended Recovery definieren.

Definition 3.10. (Extended Recovery, Maximum Extended Recovery, [FKPT11a]): Sei $\mathcal{M}$ eine Schemaabbildung von einem Quellschema S in ein Zielschema T, für die die Quell- und Zielinstanzen Nullwerte beinhalten können. Eine Schemaabbildung $\mathcal{M}'$ ist eine *Extended Recovery* von $\mathcal{M}$, wenn für jede Instanz I von S gilt, dass $(I, I) \in e(\mathcal{M}) \circ e(\mathcal{M}')$ ist. Eine Schemaabbildung $\mathcal{M}'$ ist eine *Maximum Extended Recovery* von $\mathcal{M}$, wenn sie eine Extended Recovery von $\mathcal{M}$ ist und für jede weitere Extended Recovery $\mathcal{M}''$ von $\mathcal{M}$ gilt, dass $e(\mathcal{M}) \circ e(\mathcal{M}') \subseteq e(\mathcal{M}) \circ e(\mathcal{M}'')$ ist.

Wie man erkennen kann, gibt es eine starke Verbindung zwischen der Maximum Recovery und der Maximum Extended Recovery. In [APRR09] wurde gezeigt, dass für eine Schemaabbildung $\mathcal{M}$ genau dann eine Maximum Extended Recovery existiert, wenn $e(\mathcal{M})$ eine Maximum Recovery hat. Es gilt sogar, dass $\mathcal{M}'$ genau dann eine Maximum Extended Recovery von $\mathcal{M}$ ist, wenn $e(\mathcal{M}')$ eine Maximum Recovery von $e(\mathcal{M})$ ist.

Sprache der Maximum Extended Recovery In [FKPT11a] wurde offengelassen, ob für jede durch S-T TGDs spezifizierte Schemaabbildung eine Maximum Extended Recovery in Logik erster Ordnung existiert. Der in [FKPT11a] vorgestellte Algorithmus (siehe Algorithmus 2) zur Berechnung der Maximum Extended Recovery einer Schemaabbildung liefert nur ein Ergebnis in existenzieller Logik zweiter Ordnung.

Existenz der Maximum Extended Recovery Jede durch S-T TGDs spezifizierte Schemaabbildung hat eine Maximum Extended Recovery [FKPT11a]. Das folgt direkt daraus, dass zum einen für jede durch S-T TGDs spezifizierte Schemaabbildung $\mathcal{M}$ mit Nullwerten in der Quell- und Zielinstanz die homomorphe Erweiterung $e(\mathcal{M})$ eine Maximum Recovery hat und zum anderen eine Schemaabbildung genau dann eine Maximum Extended Recovery hat, wenn $e(\mathcal{M})$ eine Maximum Recovery hat, wobei $\mathcal{M}'$ genau dann eine Maximum Extended Recovery von $\mathcal{M}$ ist, wenn $e(\mathcal{M}')$ eine Maximum Recovery von $e(\mathcal{M})$ ist.

Algorithmus Maximum Extended Recovery In [FKPT11a] wird mit Algorithmus 2 ein Polynomialzeitalgorithmus für die Berechnung der Maximum Extended Recovery $\mathcal{M}'$ einer Schemaabbildung $\mathcal{M}$ vorgestellt. Das Ergebnis des Algorithmus 2 ist sogar eine *Strong Maximum Extended Recovery*, was bedeutet, dass $\mathcal{M}'$ eine Extended Recovery von $\mathcal{M}$ ist und für jede Extended Recovery $\mathcal{M}''$ von $\mathcal{M}$ gilt, dass $e(\mathcal{M}') \subseteq e(\mathcal{M}'')$ ist. Da es ein offenes Problem ist, ob für jede Schemaabbildung eine Maximum Extended Recovery in Logik erster Ordnung existiert, liegt die berechnete Maximum Extended Recovery in existenzieller Logik zweiter Ordnung vor.

Der Algorithmus 2 berechnet im ersten Schritt eine *skolemisierte Form* einer S-T TGD. Das bedeutet, dass jede existenzquantifizierte Variable durch eine neue Skolemfunktion ersetzt wird. Die im Algorithmus 2 verwendeten Prädikate $Null(x)$ sind mit $\neg C(x)$ definiert.

Algorithmus 2 MaxExtendedRecovery($\mathcal{M}$) [FKPT11a]

Input: Schemaabbildung $\mathcal{M} = (S, T, \Sigma)$, wobei Σ eine endliche Menge von S-T TGDs ist.
Output: Strong Maximum Extended Recovery $\mathcal{M}' = (T, S, \sigma')$ von $\mathcal{M}$, wobei σ' eine Formel in existenzieller Logik zweiter Ordnung ist.

1. (Erstelle eine skolemisierte und normalisierte Form von Σ.) Erstelle Σ_1 aus Σ, indem jede S-T TGD in Σ skolemisiert wird. Erstelle Σ_2 aus Σ_1, indem jede S-T TGD $\alpha \longrightarrow (\beta_1 \wedge ... \wedge \beta_r)$ in Σ_1 durch S-T TGDs der Form $\alpha \longrightarrow \beta_1, ... \alpha \longrightarrow \beta_r$ ersetzt wird, bei denen β_i in Kopf atomar ist.

2. (Erstelle Formeln, die einen Homomorphismus und Skolemfunktionen beschreiben.) Bestehe $\mathbf{f}$ aus den Skolemfunktionssymbolen, die im vorherigen Schritt erstellt wurden, und sei h ein neues unäres Funktionssymbol (das den Homomorphismus repräsentieren wird). Bestehe Σ_1' aus Formeln folgender Form (Allquantifizierung wurde vernachlässigt):
   a. $dom_T(x) \wedge Constant(x) \longrightarrow (h(x) = x)$
   b. $dom_S(\mathbf{x}) \longrightarrow Null(f(\mathbf{x}))$, für jede f in $\mathbf{f}$
   c. $dom_S(x_1) \wedge ... \wedge dom_S(x_n) \wedge dom_S(x_1') \wedge ... \wedge dom_S(x_n')$
   $\wedge (f(x_1, ..., x_n) = f(x_1', ..., x_n')) \longrightarrow ((x_1 = x_1') \wedge ... \wedge (x_n = x_n'))$, für jede f in $\mathbf{f}$
   d. $dom_S(\mathbf{x}) \wedge dom_S(\mathbf{x}') \longrightarrow (f(\mathbf{x}) \neq f'(\mathbf{x}'))$, für disjunkte f, f' in $\mathbf{f}$
   e. $dom_S(\mathbf{x}) \wedge dom_{S \cup T}(y) \longrightarrow (f(\mathbf{x}) \neq y)$

   So besagt (a), dass h jede Konstante auf sich selbst abbildet, (b), dass der Wertebereich jeder Skolemfunktion nur aus Nullwerten besteht, (c) und (d), dass die Skolemfunktionen jeweils disjunkte Nullwerte erzeugen, und (e), dass diese Werte alle neu sind.

3. (Erstelle Formeln über den Homomorphismus.) Für jedes Relationenschema R von T (wobei R k-stellig ist) seien $y_1, ..., y_k$ neue, eindeutige Variablen und S_R eine leere Menge. Für jede S-T TGD $\alpha(\mathbf{x}) \longrightarrow R(t_1, ..., t_k)$ in Σ_2 (wobei der Ausdruck im Kopf über dem Relationenschema R definiert ist),füge zu S_R die Formel $\exists \mathbf{x}(\alpha(\mathbf{x}) \wedge (h(y_1) = t_1) \wedge ... \wedge (h(y_k) = t_k))$ hinzu. Sei τ_R die disjunktive TGD $R(y_1, ..., y_k) \longrightarrow \bigvee\{\Phi | \Phi \in S_R\}$. Bestehe $Sigma_2'$ aus den disjunktiven TGDs τ_R für jedes Relationenschema R in T. Sei σ die Konjunktion der Elemente von $\Sigma_1 \cup \Sigma_2$, und sei σ' die Formel $\exists h \exists \mathbf{f} \sigma$.

Return $\mathcal{M}' = (T, S, \sigma')$.

Den Algorithmus 2 werden wir nun anhand seiner Anwendung auf ein Beispiel aus [FKPT11a] genauer erklären. Sei die Schemaabbildung $\mathcal{M} = (S, T, \Sigma)$ mit $S = \{A = \{a_1, a_2\}, B = \{b\}, C = \{c_1, c_2, c_3, c_4\}\}$ und $T = \{Q = \{q_1, q_2, q_3, q_4\}, R = \{r_1, r_2\}\}$ gegeben und bestehe Σ aus den zwei S-T TGDs:

$$A(x_1, x_2) \wedge B(x_1) \longrightarrow Q(x_1, x_2, x_1, x_1),$$
$$C(x_1, x_2, x_2, x_3) \longrightarrow \exists y : Q(x_1, x_2, x_2, y) \wedge R(y, x_2).$$

Der erste Schritt des Algorithmus 2 startet damit, die S-T TGDs in Σ zu skolemisieren. Dafür müssen wir die existenzquantifizierte Variable der zweiten S-T TGD durch eine Skolemfunktion ersetzen. Als Skolemfunktion wählen wir die, die von den allquantifizierten Variablen des Rumpfes der S-T TGD abhängt. Hätten wir weitere existenzquantifizierte Variablen in unseren S-T TGDs, würden wir neue Skolemfunktionen nach dem gleichen Muster erstellen. In unserem Beispiel erstellen wir für die Variable y die Skolemfunktion $f(x_1, x_2, x_3)$ und erhalten Σ_1 mit:

$$A(x_1, x_2) \wedge B(x_1) \longrightarrow Q(x_1, x_2, x_1, x_1),$$
$$C(x_1, x_2, x_2, x_3) \longrightarrow Q(x_1, x_2, x_2, f(x_1, x_2, x_3)) \wedge R(f(x_1, x_2, x_3), x_2).$$

Im weiteren Verlauf des ersten Schrittes folgt die Normalisierung der Konjunktionen in den S-T TGDs aus Σ_1. Wieder müssen wir nur die zweite S-T TGD behandeln, da die erste nur aus einem Konjunkt im Kopf besteht. Die Normalisierung erfordert das Aufspalten der Konjunkte der zweiten S-T TGD in zwei neue S-T TGDs. Beide erhalten den ursprünglichen Rumpf und jeweils ein Konjunkt im Kopf. Dementsprechend folgt Σ_2 mit:

$$A(x_1, x_2) \wedge B(x_1) \longrightarrow Q(x_1, x_2, x_1, x_1),$$
$$C(x_1, x_2, x_2, x_3) \longrightarrow Q(x_1, x_2, x_2, f(x_1, x_2, x_3)),$$
$$C(x_1, x_2, x_2, x_3) \longrightarrow R(f(x_1, x_2, x_3), x_2).$$

Da der zweite Schritt des Algorithmus 2 nur aus der Definition der Formeln Σ_1' besteht, müssen wir als nächstes den dritten Schritt abarbeiten. Dafür sammeln wir für jedes Relationenschema R in T die Rümpfe der S-T TGDs, in denen R im Kopf vorkommt, in einer Menge S_R auf. Diese Rümpfe müssen noch mit Abbildungen von neuen eindeutigen y-Variablen auf die Variablen und Skolemfunktionen in R konjugiert werden. In unserem Beispiel ergibt sich die Menge S_Q für die Relation Q und die Menge S_R für die Relation R wie folgt:

$$\begin{aligned} S_Q = \{&(A(x_1, x_2) \wedge B(x_1) \wedge (h(y_{Q_1}) = x_1) \wedge (h(y_{Q_2}) = x_2) \wedge (h(y_{Q_3}) = x_1) \wedge (h(y_{Q_4}) = x_1)), \\ &(C(x_1, x_2, x_2, x_3) \wedge (h(y_{Q_1}) = x_1) \wedge (h(y_{Q_2}) = x_2) \wedge (h(y_{Q_3}) = x_2) \wedge (h(y_{Q_4}) = f(x_1, x_2, x_3)))\}, \\ S_R = \{&(C(x_1, x_2, x_2, x_3) \wedge (h(y_{R_1}) = f(x_1, x_2, x_3)) \wedge (h(y_{R_2}) = x_2))\}. \end{aligned}$$

Die eben erzeugten Mengen können wir nun nutzen, um die disjunktiven TGDs der Maximum Extended Recovery zu generieren. disjunktive TGDs sind TGDs (siehe Definition 2.7), deren Kopf aus Disjunktionen von Konjunktionen atomarer Ausdrücke besteht. Sie haben die Form

$$\forall \mathbf{x} : (\phi(\mathbf{x}) \longrightarrow \exists \mathbf{y}_1 : \psi_1(\mathbf{x}, \mathbf{y}_1) \vee ... \vee \exists \mathbf{y}_n : \psi_n(\mathbf{x}, \mathbf{y}_n)).$$

Für ihre Generierung werden wir jedes Element aus einer Menge S_R als Disjunkt im Kopf einer disjunktiven TGD hinzufügen. Der Rumpf der disjunktiven TGD besteht aus dem R-Ausdruck über den neuen eindeutigen y-Variablen. Aus S_Q entsteht so eine disjunktive TGD mit zwei Disjunkten, während die einelementige Menge S_R eine einfache TGD liefert.

Das so erzeugte Σ_2' sieht folgendermaßen aus:

$$\begin{aligned} Q(y_{Q_1}, y_{Q_2}, y_{Q_3}, y_{Q_4}) \longrightarrow & \exists x_1, x_2 : (A(x_1, x_2) \wedge B(x_1) \\ & \wedge (h(y_{Q_1}) = x_1) \wedge (h(y_{Q_2}) = x_2) \wedge (h(y_{Q_3}) = x_1) \wedge (h(y_{Q_4}) = x_1)) \\ & \vee \exists x_1, x_2, x_3 : (C(x_1, x_2, x_2, x_3) \\ & \wedge (h(y_{Q_1}) = x_1) \wedge (h(y_{Q_2}) = x_2) \wedge (h(y_{Q_3}) = x_2) \wedge (h(y_{Q_4}) = f(x_1, x_2, x_3))), \\ R(y_{R_1}, y_{R_2}) \longrightarrow & \exists x_1, x_2, x_3 : C(x_1, x_2, x_2, x_3) \wedge (h(y_{R_1}) = f(x_1, x_2, x_3)) \wedge (h(y_{R_2}) = x_2). \end{aligned}$$

Im letzten Teil des dritten Schrittes werden die Formeln des zweiten Schrittes mit Σ_2' vereinigt. Das so erzeugte $\sigma = \Sigma_1' \cup \Sigma_2'$ muss dann nur noch um die Existenzquantifizierung für den Homomorphismus und für die Skolemfunktionen erweitert werden. Das Endergebnis ist $\sigma' = \exists h \exists \mathbf{f} \sigma$.

Disjunktiver-CHASE Wir haben den CHASE-Algorithmus bisher nur für die Anwendung von Schemaabbildungen kennengelernt, die durch einfache S-T TGDs spezifiziert sind (siehe Definition 2.19). Mit MaxExtendedRecovery($\mathcal{M}$) (siehe Algorithmus 2) erzeugen wir nun disjunktive TGDs, welche eine abgeänderte Form des CHASE-Algorithmus erfordern. Leider wird der *Disjunktive-CHASE* in [FKPT11a] für *disjunktive relaxte CHASE-Inversen* und nicht für Maximum Extended Recoveries vorgestellt. Die *relaxte CHASE-Inverse* werden wir erst im nächsten Abschnitt kennenlernen. Die Definition der disjunktiven relaxten CHASE-Inversen ist für diese Arbeit nicht von Bedeutung und kann in [FKPT11a] nachgelesen werden. Für uns ist nur wichtig, dass für eine durch S-T TGDs spezifizierte Schemaabbildung $\mathcal{M}$ und eine durch disjunktive TGDs spezifizierte Rückabbildung $\mathcal{M}'$ die Aussagen, dass $\mathcal{M}'$ eine Maximum Extended Recovery von $\mathcal{M}$ und dass $\mathcal{M}'$ eine disjunktive relaxte CHASE-Inversen von $\mathcal{M}$ ist, äquivalent sind [FKPT11a]. So können wir folgende Definition des Disjunktiven-CHASEs trotzdem für Maximum Extended Recoveries verwenden. Wie wir eine Maximum Extended Recovery, die wir mit MaxExtendedRecovery($\mathcal{M}$) (siehe Algorithmus 2) erzeugen, anpassen müssen, damit sie mit dem nun für disjunktive relaxte CHASE-Inversen definierten Disjunktiven-CHASE verarbeitet werden kann, ist Teil des Konzepts im Abschnitt 4.2.

Die Definition des Disjunktiven-CHASEs baut auf dem Disjunktiven-CHASE-Schritt auf. Dieser erzeugt in jedem Durchlauf mehrere Instanzen, die jeweils eine der Disjunktionen in der angewendeten disjunktiven TGD enthalten. Daher ist das Ergebnis des Disjunktiven-CHASEs im Allgemeinen auch eine Menge von Instanzen [FKPT11a].

Definition 3.11. (Disjunktiver-CHASE-Schritt auf Instanzen, nach [FKPT11a]): Sei I eine Instanz und σ eine disjunktive TGD der Form

$$\forall \mathbf{x} : (\phi(\mathbf{x}) \longrightarrow \bigvee_{i=1}^{n} \exists \mathbf{y}_i : \psi_i(\mathbf{x}, \mathbf{y}_i)).$$

Sei $g : \phi(\mathbf{x}) \longrightarrow I$ ein Trigger für σ und I. Der Trigger g ist aktiv, wenn für kein $i \in \{1, ..., n\}$ eine Erweiterung von g zu einem Homomorphismus von Disjunkt $\psi_i(\mathbf{x}, \mathbf{y}_i)$ im Kopf von σ zu I existiert. Für jedes i mit $1 \leq i \leq n$, sei I_i die Vereinigung der Tupel in I mit der Menge an Tupel, die durch die Anwendung des Standard-CHASE-Schrittes mit I und $\phi(\mathbf{x}) \longrightarrow \exists \mathbf{y}_i : \psi_i(\mathbf{x}, \mathbf{y}_i)$ entstehen. Wir nennen die Menge $\{I_1, ..., I_n\}$ das Ergebnis des *Disjunktiven-CHASE-Schrittes* durch die Anwendung von σ auf I und schreiben $I \xrightarrow{\sigma,g} \{I_1, ..., I_n\}$. Wenn σ eine einfache TGD ist, dann reduziert sich die Menge $\{I_1, ..., I_n\}$ zu einer einzelnen Instanz I'. Der CHASE-Schritt wird dann als $I \xrightarrow{\sigma,g} I'$ geschrieben.

Definition 3.12. (Disjunktiver-CHASE auf Instanzen, nach [FKPT11a]): Sei Σ eine endliche Menge disjunktiver TGDs und I eine Instanz. Der *Disjunktive-CHASE* auf I mit Σ ist ein Baum (endlich oder unendlich) mit I als Wurzel. Für jeden Knoten I' gilt, wenn I' die Kinder $\{I_1, ..., I_p\}$ hat, muss $I' \xrightarrow{\sigma,g} \{I_1, ..., I_p\}$ für ein $\sigma \in \Sigma$ und einen Homomorphismus g gelten. Jedes Blatt L im Baum erfüllt die Bedingung, dass es kein $\sigma \in \Sigma$ und keinen Homomorphismus g gibt, sodass σ mit g auf L angewendet werden kann. Wenn der CHASE-Baum endlich ist, nennen wir die Menge der Blätter des CHASE-Baumes $\mathcal{I} = \{L_1, ..., L_m\}$ das Ergebnis des Disjunktiven-CHASE auf I mit Σ und schreiben $CHASE_\Sigma(I) = \mathcal{I}$.

Wenden wir den Disjunktiven-CHASE auf eine Schemaabbildung $\mathcal{M}'$ vom Zielschema T auf das Quellschema S an, die durch eine Menge Σ von disjunktiven TGDs spezifiziert ist, dann ist der CHASE-Baum immer endlich [FKPT11a]. Ein Problem, dass durch den Disjunktiven-CHASE entsteht, ist die Zusicherung von gültigen Ergebnissen bei Anfragen an die Instanz-Ergebnismenge $\mathcal{I}$. Das Problem werden wir gleich mit der Theorie von *Sicheren Antworten* lösen, wobei wir auch ein Beispiel für den Disjunktiven-CHASE angeben werden.

Sichere Antworten Der Disjunktive-CHASE liefert eine Menge von Instanzen $\mathcal{I}$ als Ergebnis. Jede Instanz $L \in \mathcal{I}$ stellt dabei eine mögliche Lösung der Rückabbildung dar. Um nun bei Anfragen an $\mathcal{I}$ gültige Ergebnisse garantieren zu können, wurde in [FKPT11a] die Theorie der *Sicheren Antworten* auf die durch den Disjunktiven-CHASE erzeugte Instanz-Menge $\mathcal{I}$ angewandt.

Definition 3.13. (Sichere Antworten, [FKPT11a]): Die *Sichere Antwort* einer Anfrage q auf I unter der Schemaabbildung $\mathcal{M}$ wird mit $certain_{\mathcal{M}}(q, I) = \bigcap_{(I,J)\in\mathcal{M}} q(J)$ definiert.

Die Anfrage q kann für unsere Zwecke (der Zusicherung von gültigen Ergebnissen) als die Anfrage angesehen werden, die die gesamte Quellinstanz ausgibt. In [FKPT11a] wurde bewiesen, dass für eine durch S-T TGDs spezifizierte Schemaabbildung $\mathcal{M}$, eine durch disjunktive TGDs spezifizierte Maximum Extended Recovery $\mathcal{M}'$ von $\mathcal{M}$ und eine Konjunktive-Anfrage q über dem Quellschema gilt, dass für jede Quellinstanz I

$$certain_{e(\mathcal{M})\circ e(\mathcal{M}')}(q, I) = \left(\bigcap_{L\in\mathcal{I}} q(L)\right)_\downarrow$$

ist, wobei $\mathcal{I} = CHASE_{\mathcal{M}'}(CHASE_{\mathcal{M}}(I))$ und die Notation $J_\downarrow$ eine Instanz J auf die Tupel einschränkt, die keine Nullwerte beinhalten.

Betrachten wir das Beispiel zu den Sicheren Antworten aus [FKPT11a]. Sei eine Schemaabbildung $\mathcal{M}$ durch die S-T TGDs $A(x) \longrightarrow R(x)$, $B(x) \longrightarrow R(x)$ und ihre Maximum Extended Recovery durch $\mathcal{M}'$ durch $\sigma' = R(x) \longrightarrow A(x) \vee B(x)$ spezifiziert. Sei außerdem die Instanz $I = \{(A(a), B(b))\}$ gegeben und unsere Anfrage q die Ausgabe der kompletten Quellinstanz. Für die Berechnung der Sicheren Antworten $certain_{e(\mathcal{M})\circ e(\mathcal{M}')}(q, I)$ ist es nach obiger Aussage ausreichend $\left(\bigcap_{L\in\mathcal{I}} q(L)\right)_\downarrow$ zu berechnen. Die Menge von Instanzen $\mathcal{I}$ entspricht dem Ergebnis des Disjunktiven-CHASEs mit $\mathcal{M}'$ auf dem Ergebnis des Standard-CHASE mit $\mathcal{M}$ auf I. Berechnen wir als erstes $CHASE_{\mathcal{M}}(I)$, dann erhalten wir $J = \{R(a), R(b)\}$.

Für die Berechnung von $CHASE_{\mathcal{M}'}(J)$ müssen wir den Disjunktiven-CHASE verwenden und bauen unseren CHASE-Baum mit der Wurzel J auf. Als Erstes suchen wir nach Abbildungen vom Rumpf von σ' nach J. Dabei finden wir jeweils einen Trigger für das Tupel $R(a)$ $g_1 : R(x) \longrightarrow R(a)$ und einen für das Tupel $R(b)$ $g_2 : R(x) \longrightarrow R(b)$. Beide Trigger sind aktiv, da es keine Erweiterung von g_1 oder g_2 gibt, die auch ein Disjunkt des Kopfes von σ' auf J abbildet (J besteht nur aus Tupeln über R und kann somit nicht um Abbildungen von A oder B auf J erweitert werden).

Da die Kinder eines Knotens im CHASE-Baum jeweils nur für einen Trigger gebildet werden, müssen wir einen Trigger für die Anwendung auf unserer Instanz J auswählen. Für die erste CHASE-Baum-Ebene wählen wir g_1 und führen dementsprechend den Disjunktiven-CHASE-Schritt $J \xrightarrow{\sigma',g_1} \{J_{1.1}, J_{1.2}\}$ aus. Die Instanz $J_{1.1}$ ergibt sich aus der Anwendung des Standard-CHASE-Schrittes mit der TGD $R(x) \longrightarrow A(x)$ und dem Trigger g_1 auf J, während sich die Instanz $J_{1.2}$ aus der Anwendung des Standard-CHASE-Schrittes mit der TGD $R(x) \longrightarrow B(x)$ und dem Trigger g_1 auf J ergibt:

$$J_{1.1} = \{R(a), R(b), A(a)\}, \qquad J_{1.2} = \{R(a), R(b), B(a)\}.$$

Im zweiten Durchlauf des Disjunktiven-CHASEs wählen wir jetzt den Trigger g_2 für die Anwendung auf allen Instanzen in der ersten CHASE-Baum-Ebene. Da der Trigger g_2 für $J_{1.1}$ und $J_{1.2}$ immer noch aktiv ist, führen wir dementsprechend für die zweite CHASE-Baum-Ebene die Disjunktiven-CHASE-Schritte $J_{1.1} \xrightarrow{\sigma',g_2} \{J_{2.1.1}, J_{2.1.2}\}$ und $J_{1.2} \xrightarrow{\sigma',g_2} \{J_{2.2.1}, J_{2.2.2}\}$ aus. Die Instanz $J_{2.1.1}$ ergibt sich aus der Anwendung des Standard-CHASE-Schrittes mit der TGD $R(x) \longrightarrow A(x)$ und dem Trigger g_2 auf $J_{1.1}$, während sich die Instanz $J_{2.1.2}$ aus der Anwendung des Standard-CHASE-Schrittes mit der TGD $R(x) \longrightarrow B(x)$ und dem Trigger g_2 auf $J_{1.1}$ ergibt. Die Instanzen $J_{2.2.1}$ und $J_{2.2.2}$ werden auf ähnliche Weise gebildet, sodass sich für die zweite CHASE-Baum-Ebene insgesamt folgende Instanzen ergeben:

$$J_{2.1.1} = \{R(a), R(b), A(a), A(b)\}, \qquad J_{2.2.1} = \{R(a), R(b), B(a), A(b)\},$$
$$J_{2.1.2} = \{R(a), R(b), A(a), B(b)\}, \qquad J_{2.2.2} = \{R(a), R(b), B(a), B(b)\}.$$

Für keine dieser Instanzen existiert ein aktiver Trigger, sodass sie alle Blätter des CHASE-Baumes sind und in das Ergebnis des Disjunktiven-CHASE mit einfließen. Das Endergebnis erhalten wir, indem die Instanzen auf die Tupel über dem Zielschema einschränken. Die Menge der Instanzen im Ergebnis lautet $\mathcal{I} = \{\{A(a), A(b)\}, \{A(a), B(b)\}, \{B(a), A(b)\}, \{B(a), B(b)\}\}$.

Wollen wir nun den Durchschnitt aller Ergebnisse von q auf den Instanzen in $\mathcal{I}$ bilden, können wir durch unsere Konstruktion von q einfach den Durchschnitt aller Instanzen in $\mathcal{I}$ berechnen. In unserem Beispiel erhalten wir dabei die leere Menge, was bedeutet, dass auch $certain_{e(\mathcal{M})\circ e(\mathcal{M}')}(q, I) = \emptyset$ ist. Da die Schemaabbildung $\mathcal{M}$ die Vereinigung der Werte in A und B darstellt, entspricht das Ergebnis auch unseren Erwartungen, da eine Rückabbildung von der Vereinigung nicht eindeutig aufgelöst werden kann und somit keine sicheren und gültigen Antworten möglich sind.

Ausblick Mit den vorgestellten Theorien der Fagin-Inversen bis hin zur Berechnung von Sicheren Antworten aus dem Ergebnis einer Maximum Extended Recovery haben wir alles definiert, was wir für das im nächsten Kapitel folgende Konzept benötigen. Die Forschung zu inversen Schemaabbildungen hat besten Wissens nach seit der Vorstellung der Maximum Extended Recovery keine für diese Arbeit wichtigen Fortschritte gemacht. Dennoch sollten folgende Arbeiten nicht unerwähnt bleiben:

- In [APR11] beschäftigen sich die Autoren weiter mit dem Problem, dass die Forschung zum Datenaustausch von Datenbanken ohne Nullwerten ausgeht. Sie stellen daher ein Framework für den Austausch von unvollständigen Informationen und Wissensbasen vor, welches sie auf Robustheit testen und wofür sie Ergebnisse zu dessen Aussagekraft, Anfrageverarbeitung und Berechnungskomplexität vorstellen sowie Algorithmen zur Realisierung angeben.

- In [APRR13] wird die Invertierung von Second-Order TGDs untersucht. Die Autoren stellen dafür die sogenannten *einfachen Second-Order TGDs* vor, für welche die Berechnung der Maximum Recovery in jedem Fall möglich ist. Die einfachen Second-Order TGDs teilen viele gute Eigenschaften mit den S-T TGD, sind aber deutlich aussagekräftiger als diese.

- In [GMO15] wird eine neue Sichtweise auf die Invertierung von Schemaabbildungen vorgestellt. Die Autoren versuchen nicht wie üblich eine S-T TGD zu invertieren, sondern nutzen diese S-T TGD um aus einer Zielinstanz die bestmögliche Quellinstanz abzuleiten. Der Ansatz ist grundsätzlich interessant, jedoch durch das Fehlen einer definierten Rückabbildung nicht mit dem GaLVE-Verfahren vereinbar.

Im abschließenden Abschnitt dieses Kapitels werden wir eine weitere Forschungsrichtung für die Invertierung von Schemaabbildungen kennenlernen. Dabei werden wir auch eine neuere Einordnung der Inversen anhand des CHASE vorstellen.

3.3. Invertierung mit Provenance

In diesem Abschnitt werden wir die aktuellen Forschungsergebnisse zur Provenance-Forschung am DBIS-Lehrstuhl anhand des Artikels [AH18b] vorstellen. Der folgende Abschnitt wurde in ähnlicher Form bereits in einem Bericht über die „Aktuelle Forschung des Rostocker DBIS-Lehrstuhls“ [Zim21] veröffentlicht (vorhanden am DBIS-Lehrstuhl der Uni Rostock). Der vorgestellte Artikel beschreibt die ersten Schritte der forschungsdatenreduzierenden Provenance-Forschung, in denen Fragen nach der Existenz und der Art inverser Abbildungen für ausgewählte Basisoperatoren beantwortet wurden. Konkret geht es um die Invertierung einer Auswertungsanfrage mit dem Ziel, eine minimale Teildatenbank zu erzeugen, die eine reproduzierbare Forschung gewährleistet. Für die Bestimmung der inversen Abbildung werden die Theorien des CHASE&BACKCHASE und der inversen Schemaabbildung mit den Theorien des Data Provenance vereinigt.

Die Basis aller folgenden Theorien bildet der CHASE-Algorithmus auf einer um skalare und arithmetische Operationen sowie Gruppierungen und Aggregation erweiterten relationalen Algebra. Mit dem CHASE kann eine als Kombination von Abhängigkeiten dargestellte Auswertungsanfrage q (bestehend aus den ausgewählten Basisoperatoren) in eine Quellinstanz I eingearbeitet werden. Die Ergebnisinstanz K ergibt sich mit $K = chase_{\mathcal{M}}(I)$, wobei q als Schemaabbildung $\mathcal{M} = (S, T, \Sigma)$ aufgefasst wird. Dabei ist S ein Quellschema über der Quellinstanz I, T ein Zielschema über der Zielinstanz K und Σ eine Menge von Abhängigkeiten in q. Die Provenance-Anfrage q_{prov} können wir nun als inverse Schemaabbildung $\mathcal{M}' = (T, S, \Sigma')$ interpretieren und in einem zweiten Schritt mit dem BACKCHASE verarbeiten. Die für eine reproduzierbare Forschung gesuchte minimale Teildatenbank I' von I ergibt sich durch die Berechnung von $I' = chase_{\mathcal{M}'}(K)$. Zusammengefasst kann man die Berechnung der wiederhergestellten Instanz auch mit der Hintereinanderausführung der Schemaabbildung für q und der inversen Schemaabbildung für q_{prov} beschreiben: $I' = chase_{\mathcal{M}'}(chase_{\mathcal{M}}(I))$.

Um den Grad des Informationserhalts zwischen I und I' beschreiben zu können, wurden in [FKPT11b] verschiedene *CHASE-Inversen* vorgestellt. Die CHASE-Inversen bieten die Möglichkeit, die Inversen von Schemaabbildungen mit Hilfe des CHASE-Algorithmus einzuordnen und bilden damit eine Alternative zu den Inversen, die wir im Abschnitt 3.2 vorgestellt haben. Die ursprüngliche Einordnung in die *exakte*, *klassische* und *relaxte CHASE-Inverse* wurde in [AH18b] aufgebrochen und erweitert, sodass eine exaktere Untersuchung auf das Erreichen einer CHASE-Inverse mit oder ohne zusätzliche Provenance-Informationen vorgenommen werden konnte:

- Die stärkste CHASE-Inverse ist die *exakte CHASE-Inverse*. Sie liegt vor, wenn die wiederhergestellte Instanz I' gleich der ursprünglichen Quellinstanz I ist. Formal drücken wir das mit $I = I' = chase_{\mathcal{M}'}(chase_{\mathcal{M}}(I))$ aus.

- Wenn anstatt der Gleichheit eine Äquivalenz zwischen I und I' vorliegt[5], handelt es sich um die *klassische CHASE-Inverse*, formal beschrieben mit $I \equiv I' = chase_{\mathcal{M}'}(chase_{\mathcal{M}}(I))$. Diese wird in der vorgestellten Arbeit nicht weiter betrachtet. Zum Ausgleich wurde eine zweifache Unterteilung der relaxten CHASE-Inversen vorgestellt.

- Die *relaxte CHASE-Inverse* liegt vor, wenn ein Homomorphismus von der wiederhergestellten Instanz I' zu der Quellinstanz I gebildet werden kann und I' zusätzlich ergebnisäquivalent zu I ist. Die Ergebnisäquivalenz zwischen zwei Instanzen I, I' ist mit $chase_{\mathcal{M}}(I) \equiv chase_{\mathcal{M}}(I')$ definiert.

- Die neu definierte *tupelerhaltene relaxte CHASE-Inverse* (tp-relaxt) fordert neben den Bedingungen einer relaxten CHASE-Inverse noch, dass die Anzahl der Tupel in der wiederhergestellten Instanz gleich der Anzahl der Tupel in der Quellinstanz ist ($|I| = |I'|$).

[5] Die Instanzen I und I' sind bis auf homomorphe Abbildungen von markierten Nullwerten auf andere markierte Nullwerte oder Konstanten gleich.

- Die letzte CHASE-Inverse wurde im Artikel für inverse Abbildungen definiert, die nur die Ergebnisäquivalenz erfüllen und somit nicht zu den relaxten CHASE-Inversen eingeordnet werden können. Wir bezeichnen die entsprechende Inverse als *ergebnisäquivalente CHASE-Inverse*.

Die Ordnung, die sich unter den CHASE-Inversen ergibt, lautet:

$$\text{ergebnisäquivalent} \preccurlyeq \text{relaxt} \preccurlyeq \text{tp-relaxt} \preccurlyeq \text{exakt}.$$

Diese Ordnung liefert alle notwendigen Bedingungen für die Einordnung der Basisoperatoren der erweiterten Algebra zu entsprechenden CHASE-Inversen. Sie stellt damit einen Schwerpunkt des vorgestellten Artikels dar.

Um die Existenz einer CHASE-Inversen für einen Basisoperator zu entscheiden bzw. um die Einordnung in die vorgestellte Ordnung der CHASE-Inversen vorzunehmen und anschließend zu bestimmen, ob zusätzliche Provenance-Informationen einen Mehrwert für die inverse Abbildung liefern, mussten die Basisoperatoren zuerst in S-T TGDs umformuliert werden. So können sie mit dem CHASE&BACKCHASE verarbeitet werden. Die S-T TGD-Schreibweise für das Kopieren, die Umbenennung, die Projektion, den natürliche Verknüpfung, die Selektion, den Mengenoperationen und den arithmetischen Operationen konnten aus ihren Datalog-Repräsentationen abgeleitet werden. Die Transformation der Aggregation, Gruppierung und der anderen Operatoren wurden aus [MG12, GM15] übernommen.

Die für die Bestimmung der CHASE-Inversen verwendete Anwendung des CHASE&BACKCHASE läuft wie bereits beschrieben ab: In der CHASE-Phase berechnen wir den CHASE für I und $\mathcal{M}$. Anschließend folgt die BACKCHASE-Phase, in der wir den CHASE für die Ergebnisinstanz der CHASE-Phase K und die inverse Abbildung $\mathcal{M}'$ berechnen. Die so entstandene wiederhergestellte Instanz I' entspricht dem Ergebnis der Provenance-Anfrage q_{prov} auf K. Sie enthält ganze oder auf bestimmte Attribute eingeschränkte (mit markierten Nullwerten aufgefüllte) Tupel aus I.

Antworten auf die Frage der Existenz einer Inversen für die Basisoperatoren sowie der Einordnung in die Ordnung der CHASE-Inversen[6] folgen in großen Teilen den Vorarbeiten [Aug17] und [AH18a]. Die Einordnungen aller Basisoperatoren in die CHASE-Inversen können im Artikel nachgelesen werden. Heraus sticht die Existenz einer exakten CHASE-Inversen ohne das Speichern von zusätzlichen Provenance-Informationen für das Kopieren, die Umbenennung, den natürlichen Verbund ohne Duplikate und die Ein-Variablen-Arithmetik. Im Gegensatz dazu konnten selbst mit zusätzlichen Provenance-Informationen keine CHASE-Inversen für die Selektion auf Ungleichheit und die Mengen-Differenz angegeben werden. Interessant ist ebenfalls, wie die ursprüngliche Einordnung zu der ergebnisäquivalenten CHASE-Inversen von `AVG` (Durchschnittsberechnung per Aggregierung), `SUM` (Summierung per Aggregierung) und der Mengen-Vereinigung mithilfe von zusätzlichen Provenance-Informationen zur Einordnung in die exakte CHASE-Inverse verändert werden konnte.

Diese Änderung der CHASE-Inversen-Einordnung durch den Einsatz von Provenance-Informationen werden wir nun anhand von `SUM` an einem Beispiel erläutern. Für `AVG` ist im Artikel ein äquivalentes Beispiel angegeben. Sei eine Quellinstanz I unter dem Schema $S = \{\text{Modul} = \{\text{Modulnr}, \text{Name}, \text{Punkte}\}\}$[7] wie folgt gegeben:

$$\begin{aligned} I = \{&\text{Modul}(001, \text{DB}, 9),\\ &\text{Modul}(002, \text{Mathe}, 9),\\ &\text{Modul}(002, \text{Mathe}, 6),\\ &\text{Modul}(005, \text{KI}, 3)\}. \end{aligned}$$

[6] In [Aug17] und [AH18a] gab es die tp-relaxte CHASE-Inverse noch nicht.
[7] Das Attribut Modulnr ist kein Schlüssel, daher können gleiche Module verschiedene Punkte haben.

In der CHASE-Phase arbeiten wir die Auswertungsanfrage

$$q : \text{Modul}(x_{\text{Mo}}, x_{\text{Na}}, x_{\text{Pu}}) \rightarrow \text{Ergebnis}(\texttt{SUM}(x_{\text{Pu}}))$$

in die Quellinstanz I ein. Dafür verstehen wir q als Schemaabbildung $\mathcal{M} = (S, T, \{q\})$ mit dem Zielschema $T = \{\text{Ergebnis} = \{\text{sum}_{\text{Punkte}}\}\}$. Die Ergebnisinstanz K über T ergibt sich mit $CHASE_{\mathcal{M}}(I)$ und entspricht dem Ergebnis der Anfrage q auf I:

$$K = \{\text{Ergebnis}(27)\}.$$

Um nun die wiederhergestellte Instanz I' von I unter S zu erhalten, müssen wir die Ergebnisinstanz K zusammen mit der Provenance-Anfrage

$$q_{prov} : \text{Ergebnis}(x_{\text{sP}}) \rightarrow \exists y_{\text{Mo}}, y_{\text{Na}} : \text{Modul}(y_{\text{Mo}}, y_{\text{Na}}, x_{\text{sP}})$$

im BACKCHASE verarbeiten. Dafür werden wir K mit der Schemaabbildung $\mathcal{M}' = (T, S, \{q_{prov}\})$ chasen ($I' = CHASE_{\mathcal{M}'}(K)$). Ohne weitere Provenance-Informationen erhalten wir in der wiederhergestellten Instanz I' nur das Tupel $\text{Modul}(\mu_{\text{Mo}}, \mu_{\text{Na}}, 27)$. Die so entstandene Instanz I' ist zwar ergebnisäquivalent zu I, im Allgemeinen gibt es aber keinen Homomorphismus von I' zu I. Die mit q_{prov} spezifizierte Schemaabbildung ist alleine also nur eine ergebnisäquivalente CHASE-Inverse. Diese Einordnung können wir nun durch eine Erweiterung von K verbessern. Uns stehen dafür zwei Möglichkeiten zur Verfügung:

1. Zum Einen können wir uns die genaue Berechnung der einzelnen Tupel in K mittels eines Provenance-Polynoms merken. So würde würde das Provenance-Polynom für das Ergebnis-Tupel in K wie folgt lauten:

   $$p = 9 \otimes t_1 + 9 \otimes t_2 + 6 \otimes t_3 + 3 \otimes t_4.$$

   Die Tupel-IDs t_1 bis t_4 gehören dabei zu den Tupeln in I. Mithilfe von p können wir in der BACKCHASE-Phase eine Instanz I'_p wiederherstellen, die die Werte des Punkte-Attributs für jedes Tupel in I exakt widerspiegelt:

   $$\begin{aligned} I'_p = \{&\text{Modul}(\mu_{\text{Mo}_1}, \mu_{\text{Na}_1}, 9),\\ &\text{Modul}(\mu_{\text{Mo}_2}, \mu_{\text{Na}_2}, 9),\\ &\text{Modul}(\mu_{\text{Mo}_3}, \mu_{\text{Na}_3}, 6),\\ &\text{Modul}(\mu_{\text{Mo}_4}, \mu_{\text{Na}_4}, 3)\}. \end{aligned}$$

   Da Modulnr- und Name-Attribute mit markierten Nullwerten aufgefüllt wurden, gibt es einen Homomorphismus von I'_p nach I. Zusätzlich ist die Tupelanzahl der beiden Instanzen gleich. Wenn wir ein zusätzliches Provenance-Polynom speichern, existiert mit der durch q_{prov} spezifizierten Schemaabbildung also eine tp-relaxte CHASE-Inverse für q. Im Artikel wird auch von einer exakten CHASE-Inversen für das aggregierte Punkte-Attribut gesprochen.

2. Für die zweite Möglichkeit der Verbesserung der CHASE-Inversen-Einordnung benötigen wir neben der Ergebnisinstanz K und dem Provenance-Polynom p noch eine zusätzliche Instanz K' unter einem Schema T'. Die Instanz K' speichert für jedes Tupel (t_1-t_4) die bisher nicht wiederherstellbaren Attributwerte für das Modulnr- und Name-Attribut. Das Schema T' ergibt sich aus den Attributen der nicht wiederherstellbaren Attributwerte und der Tupel-Id: $T' = \{\text{Modul'}(\text{Modulnr}, \text{Name}, \text{Tupelid}\}$.

In unserem Beispiel ergibt sich die zusätzliche Speicherung von

$$K' = \{\text{Modul}(001, \text{DB}, t_1),\\ \text{Modul}(002, \text{Mathe}, t_2 + t_3),\\ \text{Modul}(005, \text{KI}, t_4)\}.$$

Mit K, K' und p können wir jetzt durch einen BACKCHASE eine Instanz $I'_{p+K'}$ erzeugen, die gleich der Quellinstanz I ist. Somit haben wir eine exakte CHASE-Inverse für q konstruiert. Im Artikel wird darauf hingewiesen, dass die ergänzende Speicherung von K' in der Praxis oft nicht notwendig ist, da die reproduzierbare Forschung alleine mit dem Provenance-Polynom[8] gewährleistet ist. Die Speicherung ist auch nicht immer erlaubt, da z. B. Datenschutz-Aspekte gegen eine Speicherung sprechen können.

Neben der Einordnung der CHASE-Inversen konnten im Artikel auch explizite Inversen einiger Basisoperatoren angegeben werden. Die CHASE-Inversen können dabei die identische Abbildung, die Umbenennung, die Projektion, die Selektion, die Erweiterung um Nullwerte sowie die Null-Tupel-Generierung und die Rekonstruktion verlorener Attributwerte oder Tupel sein. In der Arbeit wurde herausgefunden, dass für die relaxte CHASE-Inverse immer einer der gerade genannten Inversen-Operatoren existiert. Exakt benannt werden konnte die Inverse vom natürlichen Verbund, nämlich die Projektion, die Inversen von den arithmetischen Operationen mit einer Variablen, welche die inversen arithmetischen Operationen sind, sowie die Inverse der Umbenennung, die einer weiteren Umbenennung entspricht. Für eine ergebnisäquivalente CHASE-Inverse hat sich die Identität als inverse Abbildung herausgestellt.

Um von den Basisoperatoren auf die CHASE-Inverse einer komplexeren Auswertung zu schließen, müssen wir diese als Komposition aus Basisoperatoren darstellten. Diese Komposition invertieren wir dann durch die Umkehrung der internen Reihenfolge der Basisoperatoren und deren jeweilige Invertierung. Die Einordnung der Komposition zu einer CHASE-Inversen ergibt sich dann aus der schwächsten Einordnungen aller Basisoperatoren. Sollte es keine CHASE-Inverse für einen Operator geben, können wir auch keine für die komplexe Auswertung angeben. Anwendung findet eine solche Komposition beispielsweise in Machine-Learning-Algorithmen wie dem Hidden-Markov-Modell. Hier können wir die Existenz einer Inversen untersuchen, indem wir die SQL-Darstellung des Hidden-Markov-Modells nutzten. Mit ihr können wir die zu Grunde liegenden Operatoren (Addition und Subtraktion, skalare Multiplikation und Division sowie die Matrix-Vektor- und Matrix-Matrix-Multiplikation) auf ihre CHASE-Inversen prüfen. Für das Hidden-Markov-Modell konnte so eine ergebnisäquivalente CHASE-Inverse bestimmt werden.

Im nächsten Kapitel werden wir das Konzept der neuen GaLVE-Technik vorstellen. Dafür werden wir die ursprüngliche Herangehensweise aus [Str13] (siehe Abschnitt 3.1) verwerfen und einen Ansatz vorstellen, der auf dem Algorithmus 2 für die Berechnung Maximum Extended Recovery einer beliebigen Schemaabbildung aufbaut.

[8] Nach Prof. Andreas Heuer ist hier anzumerken, dass die Speicherung und Auswertung der Provenance-Polynome zu Reproduzierbarkeitszwecken eventuell aufwendiger als die Speicherung von K' ist. Die zusätzlichen Annotationen, die man sich merken muss, sind also entweder polynombasierte Informationen oder "*Side-Tables*" wie K'.

4. Konzept

In diesem Kapitel werden wir unser neues GaLVE-Verfahren unter der Theorie von Schemaabbildungen vorstellen. Das GaLVE-Verfahren wird nach einer Datenintegration (siehe Abschnitt 2.2) ausgeführt werden, um die lokalen Datenbanken mit Informationen aus der globalen Integrationsdatenbank zu erweitern.

Die Erweiterung der lokalen Datenbanken bietet gegenüber der einfachen Datenintegration den Vorteil, dass man die zurück abgebildeten Integrationsdaten aus der globalen Datenbank, zusammen mit den lokalen Daten anfragen kann. Bei der einfachen Datenintegration stehen global nur die Daten zur Verfügung, die man auch ins globale Schema abgebildet hat. Diese Abbildung ist z. B. durch Datenschutzaspekte oder fehlender Kompatibilität des globalen Schemas nicht immer möglich. Ebenso nachteilig an der einfachen Datenintegration ist, dass man für Anfragen an die globale Datenbank das neue globale Schema nutzen muss. So müssen nicht nur neue Anfragen an das neue Schema gestellt werden, sondern auch Legacy-Anwendungen umgeschrieben werden, wenn diese auch auf die Integrationsdaten in der globalen Datenbank zugreifen sollen. In einigen Fällen müssen so Anfragen für das lokale und globale Schema entwickelt und gepflegt werden. In Abbildung 4.1 sind die gleichzeitig zu pflegenden Anfragen mit q_1 an das lokale Schema S_1 und q_1+ an das globale Schema T dargestellt.

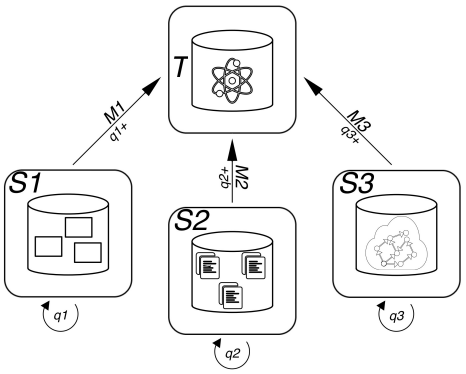

Abbildung 4.1.: Grober Ablauf Datenintegration

Da uns in unserem Ansatz des GaLVE-Verfahrens mit der Theorie von Schemaabbildungen ein großer Pool bekannter und kombinierbarer Techniken und Algorithmen zur Verfügung steht (siehe Kapitel 2 und 3), deren Korrektheit an anderer Stelle bewiesen wurde, hebt sich unser Ansatz auch von dem in [Str13] vorgestelltem GaLVE-Verfahren ab. Dieser vorherige Ansatz (siehe Abschnitt 3.1) wurde ohne die Nutzung von Schemaabbildungen entwickelt und bildet die Daten mit eigenen Regeln in das globale Schema und zurück in das lokale Schema ab. Diese eigenen Regeln erschweren eine Kombination mit weiteren Techniken der Datenintegration. Ausschlaggebend für die Entwicklung eines neuen GaLVE-Verfahrens ist für uns die Tatsache, dass die nach der Rückabbildung entstehenden neuen, um die Integrationsdaten der globalen Datenbank erweiterten lokalen Schemata, nicht zwingend ähnlich zu den originalen lokalen Schemata sind. So müssen neue Anfragen trotzdem für ein neues lokales Schema entwickelt werden und auch die bestehenden Anfragen für das neue lokale Schema transformiert werden. Dabei wurde in der Arbeit offengelassen, wie eine solche Anfragetransformation für das neuerzeugte lokale Schema und bestehende

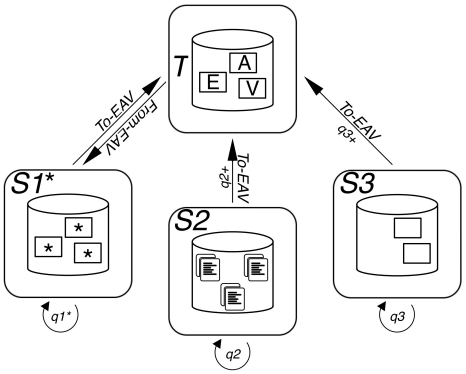

Abbildung 4.2.: Grober Ablauf des GaLVE-Konzepts nach [Str13]

Anfragen ablaufen könnte. Dargestellt ist dieses Verhalten in Abbildung 4.2 mit dem lokalen Schema S_1^* und den Anfragen q_1^* anstatt dem ursprünglichen lokalen Schema S_1 und den ursprünglichen Anfragen q_1, wie sie noch in der Abbildung 4.1 vorlagen.

Der Ansatz unseres GaLVE-Verfahrens und der Unterschied zu der einfachen Datenintegration und dem vorherigen GaLVE-Verfahren ist Abbildung 4.3 zu erkennen. Nach der einfachen Datenintegration mit den Schemaabbildungen M_1, M_2, M_3 invertieren wir für ein ausgewähltes Schema z. B. S_1 zuerst die entsprechende Schemaabbildung M_1 und erhalten eine inverse Schemaabbildung M_1'. Diese erweitern wir dann zur erweiterten inversen Schemaabbildung $M_1'+$. Die Anwendung von $M_1'+$ liefert uns automatisch unser erweitertes lokales Schema S_1+, welches alle für sich wichtigen Informationen aus der globalen Datenbank T enthält. S_1+ wird dabei so aufgebaut, dass bestehende Anfragen q_1 ohne eine weitere Anfragetransformation gleichzeitig auf die ursprünglichen lokalen Daten und auf die neu hinzugefügten Integrationsdaten aus der globalen Datenbank zugreifen können. Neue Anfragen an das neue erweiterte lokale Schema S_1+ können genau so entwickelt werden wie zuvor Anfragen an das originale lokale Schema S_1. Das wird dadurch erreicht, dass S_1 in S_1+ enthalten bleibt, wodurch man immer auch Zugriff auf die ursprüngliche lokale Datenbank unter dem ursprünglichen Schema S_1 hat. Die zusätzlichen Informationen in S_1+, die nicht im originalen Schema S_1 vorliegen, können auf Wunsch angefragt werden, stören aber keine der bestehenden Anfragen.

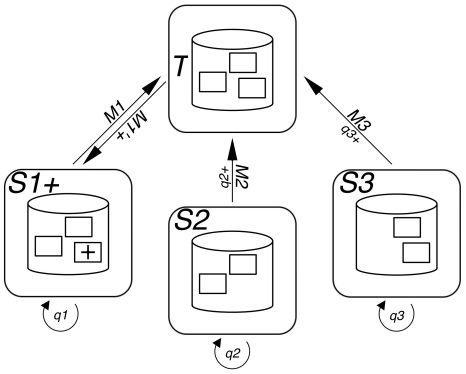

Abbildung 4.3.: Grober Ablauf des neuen GaLVE-Konzepts

Im nächsten Abschnitt werden wir einige Annahmen und Einschränkungen erläutern unter denen wir unser GaLVE-Verfahren entwerfen werden. Im darauf folgenden Abschnitt 4.2 fangen wir mit dem eigentlichen Konzept des GaLVE-Verfahrens an. Zuerst entwickeln wir einen Algorithmus für die Invertierung von beliebigen Schemaabbildungen und passen bestehende Schemaabbildungstheorien für die Verarbeitung des Ergebnisses des Algorithmus an. Darauf aufbauend entwickeln wir im Abschnitt 4.3 Algorithmen für die Erweiterung der inversen Schemaabbildung, sodass die Ausführung der erweiterten inversen Schemaabbildung ein erweitertes lokales Schema liefert, in dem bestehende Anfragen automatisch auf die lokalen und ehemals globalen Daten zugreifen können. Den Abschluss des Konzeptes bildet im Abschnitt 4.4 eine Zusammenfassung der in diesem Kapitel vorgestellten Techniken, ein Überblick über weitere mögliche Umsetzungen des GaLVE-Verfahrens in Zusammenhang materialisierte und virtuelle Ausführungen der Schemaabbildungen und Anfragen sowie die Anwendung des kompletten GaLVE-Verfahrens auf unser allgemeines Beispiel (siehe Abschnitt 1.1).

4.1. Annahmen und Einschränkungen

In diesem Abschnitt werden wir die Annahmen und Einschränkungen erklären und begründen, unter denen wir unser GaLVE-Verfahren entwickeln werden. Dazu zählen zum einen die Annahmen, die direkt für die folgenden GaLVE-Techniken getroffen werden, aber auch Einschränkungen, die sich durch die Verwendung von Schemaabbildungen mit S-T TGDs oder der Umsetzung des GaLVE-Verfahrens als Erweiterung des Datenbankforschungsprototyps ChaTEAU (siehe Abschnitt 5.1) ergeben.

Unser GaLVE-Verfahren wird als ein Aufsatz der herkömmlichen Datenintegration entwickelt. Das bedeutet, dass die Datenintegration bereits vor dem GaLVE-Verfahren durchgeführt wird und das globale

Schema, die Daten der globalen Datenbank sowie die Schemaabbildungen von den lokalen Schemata in das globale Schema als gegeben angesehen werden. Die Schritte, die sich im Ablauf der Datenintegration ergeben sowie die Probleme, die während dessen auftreten, werden wir in dieser Arbeit nicht weiter behandeln. Dazu zählen explizit auch das Matching der lokalen Schemata, die Integration dieser zu einem globalen Schema und die Generierung der Abbildung unter ihnen, wie sie im MIA-Prozess in Abschnitt 2.2 vorgestellt wurden.

Für die Datenintegration werden wir im folgenden Konzept zur Vereinfachung noch zusätzlich fordern, dass alle Schlüssel global, unter den lokalen Schemata eindeutig sind. Somit müssen wir uns nicht um gleiche Daten mit unterschiedlichen Schlüsseln innerhalb verschiedener lokaler Datenbanken kümmern und können dieses Datenintegrationsprobleme ebenfalls aus unserem Konzept auslagern. Diese Annahme vereinfacht in erster Linie die Bereinigung der integrierten Daten im globalen Schema, aber auch die Bereinigung der aus der globalen Datenbank in die Quelldatenbanken zurück integrierten Daten. In Abschnitt 4.3.2 werden wir so z. B. Daten aus unterschiedlichen Datenbanken über ihre Schlüsselattribute gleichsetzen können.

Das in der Vorarbeit genutzten Entity-Attribute-Value-Modell (siehe Abschnitt 3.1 und [Str13]) werden wir in dieser Arbeit nicht weiter untersuchen. Zum einen sind die Ergebnisse des GaLVE-Verfahrens mit dem Entity-Attribute-Value-Modell nicht mit unseren Anforderungen der Weiternutzung bestehender Anfragen vereinbar, zum anderen ist das Entity-Attribute-Value-Modell nicht mit den Schemaabbildungen mittels S-T TGDs kompatibel. In unserem GaLVE-Verfahren werden wir daher nur Daten im Relationenmodell betrachten, auf denen wir die in Kapitel 2 vorgestellten Grundlagen anwenden können.

Durch die Beschränkung auf Daten im Relationenmodell entfallen auch alle Überlegungen zu heterogenen Datenstrukturen. Von den Problemen der Datenintegration mit heterogenen Datenstrukturen grenzen wir uns dadurch ab, dass heterogene Datenquellen für das GaLVE-Verfahren vorher in das Relationenmodell umgewandelt und dann nach dem GaLVE-Verfahren wieder zurück konvertiert werden könnten.

Auch wenn unsere Beschränkung auf Daten im Relationenmodell den Vorteil hat, dass wir die Theorie der Schemaabbildungen mit S-T TGDs nutzen können, bilden gerade diese eine weitere Einschränkung für unser GaLVE-Verfahren. So schließen wir mit der Beschränkung auf Schemaabbildungen mit S-T TGDs alle bekannten Datenintegrationsfunktionalitäten aus, die nicht mittels S-T TGDs umsetzbar sind. Darunter fallen Kleinigkeiten wie die Datentypenanpassung, aber auch wichtige Konzepte wie die Unterstützung von Vergleichsprädikaten und Negationen sowie die Anpassung der Datenwerte mittels Skalarfunktionen. Ebenso sind Aggregatfunktionen wie die Summierung, die Aggregation oder die Gruppierung nicht so mit S-T TGDs umsetzbar, dass sie im GaLVE-Verfahren verwendet werden könnten. Allgemein bildet die Sprache der mit S-T TGDs umsetzbaren Schemaabbildungen nur einen Bruchteil der Funktionalität einer richtigen Datenbanksprache wie z. B. SQL ab. Unser GaLVE-Verfahren bildet daher nur die Basis, um in folgenden Arbeiten die Unterstützung solcher Techniken wie z. B. Skalarfunktionen oder Aggregatfunktionen hinzuzufügen.

Da wir in dieser Arbeit auch Anfragen als S-T TGDs darstellen, treffen für sie erst einmal dieselben Beschränkungen wie für die Schemaabbildungen zu. Der Unterschied zu den Schemaabbildungen ist, dass wir Anfragen an die erweiterte lokale Datenbank oder die globale Datenbank nach dem Ablauf des GaLVE-Verfahrens stellen und diese in der Praxis somit nicht auf S-T TGDs limitiert werden müssen. Nach der Erweiterung einer Quelldatenbank mit dem GaLVE-Verfahren steht es dem Anwender frei, richtige Datenbanksprachen zu verwenden. Unser GaLVE-Verfahren bereitet nur die Daten in der erweiterten Quelldatenbank für folgende Anfragen vor, die Anfragen stellen an sich aber keinen Teil des GaLVE-Verfahrens dar. Sollte man bei der S-T TGD-Darstellung von Anfragen bleiben, muss der Funktionsumfang von S-T TGDs in folgenden Arbeiten neben den angesprochenen Techniken (die schon für

die Schemaabbildungen notwendig waren) noch um weitere Techniken, wie die Möglichkeit für eine statistische Auswertung und des maschinellen Lernens erweitert werden.

Die in diesem Kapitel folgenden Techniken für das GaLVE-Verfahren werden in ihren Abschnitten einzeln entwickelt und dann erst im Abschnitt 4.4.1 zum gesamten GaLVE-Verfahren zusammengesetzt. Das bedeutet, dass wir auch im GaLVE-Verfahren die einzelnen Techniken nacheinander ausführen werden und unter ihnen keine Kompositionen durchführen werden. Dadurch werden wir in dieser Arbeit auch keine Komposition von Schemaabbildungen oder sonstigen Integritätsbedingungen und Anfragen betrachten. Ein Grund für diese Umsetzung des GaLVE-Verfahrens ist die Implementierung unseres Konzepts als Erweiterung von ChaTEAU, wodurch wir uns auf S-T TGDs in Logik 1. Ordnung limitieren müssen.

Die Implementierung des GaLVE-Verfahrens als Erweiterung von ChaTEAU werden wir im 5. Kapitel vorstellen. Die angesprochene Limitierung auf S-T TGDs in Logik 1. Ordnung rührt daher, dass wir den in ChaTEAU implementierten CHASE-Algorithmus als Grundlage der Ausführung der Schemaabbildungen von und zu den Quelldatenbanken verwenden werden. Somit müssen wir die Theorie unseres GaLVE-Verfahrens auch an dessen Funktionalität und Einschränkungen ausrichten. In ChaTEAU gibt es unter anderem keine Unterstützung für Skolemisierung, Disjunktionen oder sonstigen Funktionen, die für Formeln in Logik 2. Ordnung benötigt werden. Aufgrund dieser Einschränkungen können wir z. B. auch nicht den Algorithmus 2 ohne Weiteres zur Invertierung von Schemaabbildungen nutzen, da weder die skolemisierten S-T TGDs noch die Homomorphismen innerhalb der disjunktiven TGDs, geschweige denn die disjunktiven TGDs an sich unterstützt werden. In Abschnitt 4.2.1 stellen wir Algorithmen vor, die uns eine inverse Schemaabbildung ohne diese Konzepte liefern werden.

Eine weitere Einschränkung von ChaTEAU ist, dass die Attribute pro Relationenschema eindeutig sein müssen. Das bedeutet, dass Attribute mit dem gleichen Namen auch die gleiche Bedeutung für die Datenbank haben müssen. Im CHASE-Algorithmus von ChaTEAU werden Nullwerte und Variablen z. B. mit dem Attribut indiziert, über dem sie definiert sind und gleich indizierte Nullwerte und Variablen somit auch gleichgesetzt oder mit denselben Werten ersetzt. Diese Einschränkung von ChaTEAU werden wir als Annahme für unser GaLVE-Verfahren übernehmen. So werden wir die folgenden GaLVE-Techniken ebenfalls unter der Annahme entwickeln, dass gleich benannte Attribute auch auf Daten mit der gleichen Bedeutung verweisen. Diese Annahme wird für uns für alle Algorithmen und Theorien wichtig sein, in denen wir über Attribute in einem Relationenschema oder Variablen über den Attributen in den Schemaabbildungen iterieren und aufgrund von Gleichheiten unter ihnen Entscheidungen für das GaLVE-Verfahren treffen. Wir gehen für das GaLVE-Verfahren so weit, dass wir gleiche Bedeutung für gleiche Attribute sogar über allen lokalen Schemata und dem globalen Schema fordern. Diese Annahme zählen wir zu einem der Probleme, die wir mit den Problemen der Datenintegration auslagern können, da auch dort die Probleme von unterschiedlich benannten Attributen mit der gleichen Bedeutung untersucht werden.

Keine Besonderheit in der Datenintegration, aber eine weitere Annahme, die wir für unser Konzept treffen, ist, dass die Quelldatenbanken nach der Datenintegration erhalten bleiben. Das ermöglicht uns im GaLVE-Verfahren die bestehenden Daten in den Quelldatenbanken als korrekt zu betrachten und Konflikte zu ihren Gunsten aufzulösen. Eventuell auftretende Inkonsistenzen bei der Rückintegration der Daten aus der globalen Datenbank können so ohne weitere Probleme bereinigt werden. Wir könnten für das GaLVE-Verfahren somit auch auf ein Data Cleaning in der globalen Datenbank verzichten. Sollten die Daten in der globalen Datenbank trotzdem bereinigt werden, dann stellt es für die Rückintegration kein Problem mehr dar, wenn Konflikte global zugunsten einer anderen Quelle aufgelöst wurden, da wir diese lokal wieder zugunsten der vorhandenen Daten auflösen können. Diese Auflösung der Datenkonflikte zugunsten der originalen Quelldaten werden wir im Abschnitt 4.3.2 mittels einer speziellen Ausführung von TGDs und EGDs erreichen.

Diese simple Annahme, dass die lokalen Datenbanken erhalten bleiben, hat sogar noch mehr Konsequenzen für unser GaLVE-Verfahren. So müssen wir, im Gegensatz zum vorherigen GaLVE-Verfahren in [Str13] (siehe Abschnitt 3.1), unsere Schemaabbildungen von den lokalen Schemata in das globale Schema nicht auf die Fagin-invertierbaren Schemaabbildungen (siehe Abschnitt 3.2.1) einschränken und können stattdessen beliebige durch S-T TGDs spezifizierte Schemaabbildungen erlauben. Da unsere Quelldatenbank nach der Datenintegration bestehen bleibt, ist es auch nicht notwendig, dass alle Daten in die globale Datenbank übertragen werden, um sie dann bei der Rückabbildung wieder rekonstruieren zu können. In unseren GaLVE-Verfahren werden wir die bestehende Quelldatenbank einfach mit den Informationen, die sich aus der globalen Datenbank ergeben, anreichern. Das entkräftet auch das Argument in [Str13] (siehe Abschnitt 3.1), dass ein globales Schema im Entity-Attribute-Value-Modell für die Speicherung der lokalen Daten notwendig ist, um die vielen Nullwerte für die Attribute ohne Gegenpart in den anderen Schemata zu vermeiden. Dadurch, dass wir solche Attribute nicht zwangsweise in das globale Schema integrieren müssen, kann die globale Datenbank ohne diese Attribute ohne Gegenpart in anderen Schemata und somit ohne die vielen Nullwerte aufgebaut werden.

Die in diesem Abschnitt vorgestellten Annahmen und Einschränkungen des folgenden GaLVE-Verfahrens dienen dazu, uns bei der Beschreibung der einzelnen Techniken auf das Wichtigste zu konzentrieren. Eine Aufweichung der einzelnen Annahmen und Einschränkungen bezüglich der Datenintegration sollte kein großes Problem darstellen, wenn man deren Theorie mit dem GaLVE-Verfahren kombiniert. Die Einschränkungen, die sich aus der Implementierung als Erweiterung von ChaTEAU ergeben, sollten ebenfalls kein Problem darstellen, wobei manche Annahmen wie die Eindeutigkeit von Attributen tief in den folgenden Theorien verwurzelt sind. In den folgenden Abschnitten dieses Kapitels folgen nun die einzelnen Techniken für unser GaLVE-Verfahren unter den Annahmen und Einschränkungen dieses Abschnitts. Starten werden wir mit der Invertierung der Schemaabbildungen.

4.2. Invertierung von Schemaabbildungen

Die Invertierung von Schemaabbildungen ist einer der wichtigsten Schritte im GaLVE-Verfahren. Im Kapitel 3 haben wir bereits die aktuelle Forschung für geeignete Invertierungstechniken untersucht. Dabei hat sich herausgestellt, dass nur die in Abschnitt 3.2.4 vorgestellte Maximum Extended Recovery Nullwerte in den Quellen einer Schemaabbildung erlaubt. Da unsere globale Datenbank aus der Integration von verschiedenen Quelldatenbanken entsteht, können in ihr natürlich Nullwerte enthalten sein. Somit ist die Maximum Extended Recovery die einzige Invertierungstechnik, die sinnvoll für das GaLVE-Verfahren genutzt werden kann.

Leider hat die Wahl der Maximum Extended Recovery auch Nachteile. So ist ihre Existenz in Logik erster Ordnung nicht beweisbar. Gleichzeitig gibt der Algorithmus 2 für ihre Berechnung eine Schemaabbildung in Logik zweiter Ordnung zurück. Da wir die GaLVE-Technik als Erweiterung von ChaTEAU entwickeln, müssen wir uns in diesem Konzept auch an die Einschränkungen des in ChaTEAU implementierten Standard-CHASE halten. Für den Algorithmus 2 bedeutet das, dass wir ihn so anpassen müssen, dass er eine Schemaabbildung in Logik erster Ordnung liefert. Diese Anpassungen werden wir im Abschnitt 4.2.1 beschreiben. Im darauf folgenden Abschnitt 4.2.2 passen wir den Disjunktiven-CHASE an die Verarbeitung der Schemaabbildung aus dem angepassten Algorithmus an. Zusätzlich werden wir untersuchen, wie man mit der Theorie der Sicheren Antworten eine möglichst große Instanz aus dem Durchschnitt der Ergebnisinstanzen des Disjunktiven-CHASEs garantieren kann.

4.2.1. Algorithmus Invertierung

Die problematischen Strukturen, die im Ergebnis von Algorithmus 2 zur Logik zweiter Ordnung führen, sind die existenzquantifizierten Skolemfunktionen und Homomorphismen. Auch die disjunktiven TGDs können nicht ohne Weiteres im ChaTEAU-CHASE verarbeitet werden. Als Lösung dieser Probleme stellen wir nun den angepassten Algorithmus AdaptedMaxExtendedRecovery($\mathcal{M}$) (siehe Algorithmus 3) vor. Dieser liefert eine mit ChaTEAU kompatible inverse Schemaabbildung, indem er den originalen Algorithmus 2 an den richtigen Stellen anpasst, kürzt oder erweitert.

Mehr zu Eingabe / Ausgabe Als Eingabe erhält der Algorithmus 3 genau wie der Algorithmus 2 eine Schemaabbildung $\mathcal{M} = (S, T, \Sigma)$. Diese stellt eine der Schemaabbildungen für die vorherige Datenintegration von einer Quelldatenbank unter dem Schema S zur globalen Datenbank unter dem Schema T dar. Die endliche Menge an Integritätsbedingungen Σ besteht ausschließlich aus S-T TGDs.

So wie das Ergebnis des Algorithmus 2 noch eine Strong Maximum Extended Recovery $\mathcal{M}' = (T, S, \sigma')$ von $\mathcal{M}$ war, ist das Ergebnis unseres angepassten Algorithmus nun eine *Adapted Strong Maximum Extended Recovery* $\mathcal{M}' = (T, S, \Omega')$ von $\mathcal{M}$. Der Unterschied zwischen der Strong Maximum Extended Recovery und der Adapted Strong Maximum Extended Recovery liegt in der Darstellung der Integritätsbedingungen. Während σ' noch eine Formel in existenzieller Logik zweiter Ordnung war, ist Ω' eine Menge von Mengen von S-T TGDs in Logik erster Ordnung. Diese Mengen von S-T TGDs werden wir nun als *Disjunktive-Mengen* definieren, sodass Ω' eine Menge von Disjunktiven-Mengen ist.

Definition 4.1. (Disjunktive-Menge): Eine *Disjunktive-Menge* ist eine Menge von S-T TGDs, deren Rümpfe aus einem einzelnen Ausdruck bestehen, wobei jeder dieser S-T TGD-Rumpf-Ausdrücke über demselben Relationenschema definiert ist. Das Relationenschema der Ausdrücke in den S-T TGD-Rümpfen nennen wir auch *Rumpf-Relationenschema* der Disjunktiven-Menge. Die Variablen in dem Rümpfen der einzelnen S-T TGDs müssen dabei nicht miteinander übereinstimmen.

Algorithmus 3 AdaptedMaxExtendedRecovery($\mathcal{M}$)

Input: Schemaabbildung $\mathcal{M} = (S, T, \Sigma)$, wobei Σ eine endliche Menge von S-T TGDs ist.
Output: *Adapted Strong Maximum Extended Recovery* $\mathcal{M}' = (T, S, \Omega')$ von $\mathcal{M}$, wobei Ω' eine Menge von Mengen von S-T TGDs (Menge von *Disjunktiven-Mengen*) in Logik erster Ordnung ist.

1. (Normalisiere die Konjunktionen in den Köpfen der S-T TGDs.) Erstelle Σ' aus Σ, indem jede S-T TGD $\alpha \longrightarrow (\beta_1 \wedge ... \wedge \beta_r)$ in Σ durch S-T TGDs der Form $\alpha \longrightarrow \beta_1, ..., \alpha \longrightarrow \beta_r$ ersetzt wird, bei denen β_i im Kopf atomar ist.

2. (Erstelle disjunktive TGDs über einem Homomorphismus.) Für jedes Relationenschema $R \in T$ (wobei R k-stellig ist) seien $z_{R_1}, ..., z_{R_k}$ neue, eindeutige Variablen und S_R eine neue leere Menge. Für jede Formel $\alpha(\mathbf{x}) \longrightarrow R(t_1, ..., t_k)$ in Σ' (wobei der Ausdruck im Kopf über dem Relationenschema R definiert ist), füge zu S_R die Formel $\exists \mathbf{x}(\alpha(\mathbf{x}) \wedge (h(z_{R_1}) = t_1) \wedge ... \wedge (h(z_{R_k}) = t_k))$ hinzu. Sei τ_R die disjunktive TGD $R(z_{R_1}, ..., z_{R_k}) \longrightarrow \bigvee\{\Phi | \Phi \in S_R\}$. Bestehe Σ'' aus den disjunktiven TGDs τ_R für jedes Relationenschema R in T.

3. (Normalisiere die Disjunktionen in den Köpfen der disjunktiven TGDs.) Sei $\Omega = \{\omega_1, ..., \omega_q\}$ eine Menge von Mengen von S-T TGDs (Menge von *Disjunktiven-Mengen*), mit $q = |\Sigma''|$. Jedes $\omega_i \in \Omega$ entsteht aus einer disjunktiven TGD in Σ''. Für eine disjunktive TGD $R(z_{R_1}, ..., z_{R_k}) \longrightarrow \bigvee\{\Phi | \Phi \in S_R\}$, wird für jedes $\Phi_j \in S_R$ eine S-T TGD der Form $R(z_{R_1}, ..., z_{R_k}) \longrightarrow \Phi_j$ zu ω_i hinzugefügt.

4. (Wende den Homomorphismus an.) Erstelle Ω' aus Ω, indem die Abbildungen in den einzelnen S-T TGDs in jedem $\omega_i \in \Omega$ angewendet werden.

Return $\mathcal{M}' = (T, S, \Omega')$.

Mit den Disjunktiven-Mengen können wir nun die Adapted Strong Maximum Extended Recovery definieren. Deren Nutzung ist ausschließlich auf eine Verwendung in ChaTEAU beschränkt. Das liegt daran, dass wir für ihre Definition den Algorithmus AdaptedMaxExtendedRecovery($\mathcal{M}$) nutzen werden, für dessen Entwicklung wir die Eigenschaften von ChaTEAU genutzt haben. Welche Eigenschaften das sind, folgt in den nächsten Paragraphen.

Definition 4.2. (Adapted Strong Maximum Extended Recovery): Sei $\mathcal{M}$ eine Schemaabbildung von einem Quellschema S in ein Zielschema T, für die die Quell- und Zielinstanzen Nullwerte beinhalten können. Eine Schemaabbildung $\mathcal{M}'$ ist eine *Adapted Strong Maximum Extended Recovery* von $\mathcal{M}$, wenn $\mathcal{M}' = (T, S, \Omega')$ das Ergebnis von AdaptedMaxExtendedRecovery($\mathcal{M}$) ist. Ω' ist dabei eine Menge von Disjunktiven-Mengen.

Mehr zu Schritt 1 Im ersten Schritt des Algorithmus 3 erstellen wir eine normalisierte Form von den S-T TGDs in Σ. Dafür wird für jedes Konjunkt, sprich für jeden Ausdruck in den Köpfen der S-T TGDs, eine eigene S-T TGD mit einem atomaren Kopf erstellt.

Wichtig ist, dass wir im Vergleich zum ersten Schritt im Algorithmus 2 die existenzquantifizierten Variablen in den Köpfen der S-T TGDs nicht skolemisieren. Da wir die Schemaabbildung aus dem Ergebnis des Algorithmus direkt anwenden werden, können wir die Skolemisierung vernachlässigen. Diese wären nur dann wichtig gewesen, wenn wir unsere inverse Schemaabbildung noch mit anderen Schemaabbildungen komponieren wollten. Die verbleibenden existenzquantifizierten Variablen aus den S-T TGD-Köpfen spielen für die weitere Invertierung keine Sonderrolle mehr. Sie werden im nächsten Schritt in die Rümpfe von disjunktiven TGDs überführt und somit als allquantifizierte Variablen behandelt. Diese Vereinfachung des ersten Schrittes führt dazu, dass wir ChaTEAU nicht um Skolemfunktionen erweitern müssen.

Mehr zu Schritt 2 In Schritt 2 erstellen wir disjunktive TGDs, die bereits von dem globalen in das lokale Schema abbilden. Dabei wird für jedes Relationenschema im globalen Schema eine disjunktive TGD erstellt, solange das Relationenschema in einem S-T TGD-Kopf der Eingabe-Schemaabbildung vorkommt[1]. Die Abbildungen unter einem Homomorphismus, die zu jedem Disjunkt im Kopf einer disjunktiven TGD hinzugefügt werden, gewährleisten, dass die Beziehungen der Attribute innerhalb des S-T TGD-Kopfs der Eingabe-Schemaabbildung untereinander erhalten bleiben. So werden z. B. zwei neue Variablen mit den Abbildungen auf die gleiche alte Variable abgebildet, wenn die Variablen im ursprünglichen Ausdruck des S-T TGD-Kopfs an diesen Stellen auch gleich waren. Zu bemerken ist, dass alle Variablen, die im Rumpf der entstehenden disjunktiven TGDs auftauchen, allquantifiziert sind. Explizit sind auch die im vorherigen Schritt bewahrten existenzquantifizierten y-Variablen nun allquantifiziert. Alle Variablen, die nur im Kopf der disjunktiven TGDs auftauchen, bleiben existenzquantifiziert.

Diese Abbildungen innerhalb der disjunktiven TGDs wurden im Algorithmus 2 in Schritt 3 hinzugefügt. Der Unterschied zu unserem Schritt 2 ist die Bezeichnung der neuen Variablen mit z, um sie von den existenzquantifizierten y-Variablen (die im Schritt 1 nicht entfallen) unterscheiden zu können. Dazu kommt natürlich, dass der 3. Schritt in Algorithmus 2 die disjunktiven TGDs innerhalb einer Schemaabbildung als Ergebnis zurückgibt und wir sie noch in zwei weiteren Schritten verarbeiten.

Ein weiterer Unterschied zwischen Algorithmus 2 und 3 ist der in Algorithmus 3 fehlende 2. Schritt des Algorithmus 2. Im 2. Schritt des Algorithmus 2 werden Regeln definiert, die dafür sorgen, dass der erzeugte Homomorphismus jede Konstante auf sich selbst abbildet, der Wertebereich jeder Skolemfunktion nur aus Nullwerten besteht, dass die Skolemfunktionen jeweils disjunkte Nullwerte erzeugen und dass diese Werte alle neu sind. Die Regeln zu den Skolemfunktionen entfallen in unserem Algorithmus zum einen dadurch, dass wir die Skolemfunktionen im ersten Schritt weglassen. Zum anderen können wir generell von all diesen Regeln innerhalb einer Schemaabbildung absehen, da ChaTEAU bereits alle implementiert. So werden Nullwerte sowieso immer mit einem entsprechenden Attributnamen und einem Index versehen, mit denen dafür gesorgt wird, dass neu hinzugefügte Nullwerte innerhalb der gesamten Instanz neu sind. ChaTEAU überprüft bei der Ausführung von Integritätsbedingungen auch die Homomorphismusregeln des CHASE-Algorithmus (siehe Definition 2.9), sodass ganz automatisch dafür gesorgt wird, dass Konstanten nur auf sich selbst abgebildet werden. Wir können die Erstellung von Formeln für einen Homomorphismus und Skolemfunktionen somit komplett weglassen und brauchen die Integritätsbedingungen in ChaTEAU nicht damit erweitern.

Mehr zu Schritt 3 Der 3. Schritt normalisiert jetzt die Disjunktionen in den Köpfen der disjunktiven TGDs. Die Normalisierung der Disjunktionen läuft dabei ähnlich wie die Normalisierung der Konjunktionen im ersten Schritt ab. Der Unterschied ist, dass wir die durch die Normalisierung erzeugten S-T TGDs nicht in einer einzelnen Menge von Integritätsbedingungen vereinigen, sondern jeweils eine (Disjunktive-)Menge für die S-T TGDs erstellen, die aus einer disjunktiven TGD erstellt werden. Das ist notwendig, da die S-T TGD in so einer Disjunktiven-Menge so verarbeitet werden müssen wie die Disjunktionen in den disjunktiven TGDs[2]. Eine entsprechende Anpassung des Disjunktiven-CHASEs folgt im Abschnitt 4.2.2 (siehe Definition 4.3 und 4.4). Die Abbildungen innerhalb der Disjunkte in den Köpfen der disjunktiven TGDs bleibt noch bis zum nächsten Schritt in den Köpfen der S-T TGDs bestehen.

[1] Der Algorithmus 3 erzeugt auch für die Relationenschemata eine disjunktive TGD, die vorher nicht durch die Eingabe-Schemaabbildung gefüllt wurden. Diese disjunktiven TGDs sind allerdings trivial, da sie einen leeren Kopf haben und werden von uns nicht weiter erwähnt.

[2] Gemeint ist der Aufbau eines CHASE-Baumes durch den Disjunktiven-CHASE.

Mehr zu Schritt 4 Im letzten Schritt des Algorithmus wenden wir den durch die Abbildungen in den S-T TGDs definierten Homomorphismus an. Durch die Anwendung des Homomorphismus innerhalb des Algorithmus entfällt eine Erweiterung des ChaTEAU-CHASE um Homomorphismen innerhalb von Integritätsbedingungen. Die entstehende Menge Ω' besteht nun aus Disjunktiven-Mengen, in denen die Ausdrücke über dem Rumpf-Relationenschema der Disjunktiven-Menge unterschiedliche Variablen für die einzelnen Attribute enthalten können. Das verdeutlicht die Notwendigkeit der Disjunktiven-Menge als eine Menge von S-T TGDs gegenüber einer S-T TGD mit einer Menge von S-T TGD-Köpfen.

Beispiel Invertierung Um die Ähnlichkeit, aber auch die Unterschiede zum Algorithmus 2 zu zeigen, werden wir nun das Beispiel zum Algorithmus 2 aus Abschnitt 3.2.4 mit dem Algorithmus 3 durchrechnen. Sei dazu wieder die Schemaabbildung $\mathcal{M} = (S, T, \Sigma)$ mit $S = \{A = \{a_1, a_2\}, B = \{b\}, C = \{c_1, c_2, c_3, c_4\}\}$ und $T = \{Q = \{q_1, q_2, q_3, q_4\}, R = \{r_1, r_2\}\}$ gegeben und bestehe Σ aus den zwei S-T TGDs:

$$A(x_1, x_2) \wedge B(x_1) \longrightarrow Q(x_1, x_2, x_1, x_1),$$
$$C(x_1, x_2, x_2, x_3) \longrightarrow \exists y : Q(x_1, x_2, x_2, y) \wedge R(y, x_2).$$

Der erste Schritt des Algorithmus startet diesmal nicht mit der Skolemisierung der Integritätsbedingungen in Σ, sondern direkt mit deren Normalisierung. Dafür erzeugen wir wie im Algorithmus 2 für jedes Konjunkt in den Köpfen der S-T TGDs eine eigene S-T TGD mit dem ursprünglichen Rumpf und den jeweiligen Konjunkt im Kopf. Wieder müssen wir nur die zweite S-T TGD behandeln, da die erste nur aus einem Konjunkt im Kopf besteht. Dass wir bei der Normalisierung den einen Existenzquantor für die gemeinsame y-Variable in zwei Existenzquantoren aufspalten, stellt für uns kein Problem dar. Das liegt daran, dass die Köpfe der so entstehenden S-T TGDs in unserem Endergebnis die Rümpfe der S-T TGDs in den Disjunktiven-Mengen bilden werden und die y-Variablen somit zum Schluss allquantifiziert sind, wodurch die entstehenden Existenzquantoren wieder entfallen. Nach der Normalisierung ergibt sich Σ' mit den S-T TGDs:

$$A(x_1, x_2) \wedge B(x_1) \longrightarrow Q(x_1, x_2, x_1, x_1),$$
$$C(x_1, x_2, x_2, x_3) \longrightarrow \exists y : Q(x_1, x_2, x_2, y),$$
$$C(x_1, x_2, x_2, x_3) \longrightarrow \exists y : R(y, x_2).$$

Im zweiten Schritt des Algorithmus 3 sammeln wir, wie im dritten Schritt des Algorithmus 2, für jedes Relationenschema R in T die Rümpfe der S-T TGDs, in denen R im Kopf vorkommt, in einer Menge S_R auf. Auch hier konjugieren wir die Rümpfe mit Abbildungen von neuen eindeutigen Variablen auf die Variablen in R, nur dass wir dieses Mal z-Variablen verwenden, um uns von den nicht skolemisierten y-Variablen abzugrenzen. In unserem Beispiel ergibt sich die Menge S_Q für die Relation Q und die Menge S_R für die Relation R wie folgt:

$$\begin{aligned} S_Q = \{&(A(x_1, x_2) \wedge B(x_1) \wedge (h(z_{Q_1}) = x_1) \wedge (h(z_{Q_2}) = x_2) \wedge (h(z_{Q_3}) = x_1) \wedge (h(z_{Q_4}) = x_1)), \\ &(C(x_1, x_2, x_2, x_3) \wedge (h(z_{Q_1}) = x_1) \wedge (h(z_{Q_2}) = x_2) \wedge (h(z_{Q_3}) = x_2) \wedge (h(z_{Q_4}) = y))\}, \\ S_R = \{&(C(x_1, x_2, x_2, x_3) \wedge (h(z_{R_1}) = y) \wedge (h(z_{R_2}) = x_2))\}. \end{aligned}$$

Die eben erzeugten Mengen werden wir nun wieder dafür nutzen, um disjunktive TGDs zu generieren. Dies läuft wie zuvor im Algorithmus 2 ab, indem wir jedes Element aus einer Menge S_R als Disjunkt im Kopf einer disjunktiven TGD hinzufügen. Der Rumpf der disjunktiven TGD besteht aus dem R-Ausdruck über den neuen eindeutigen z-Variablen. Aus S_Q entsteht so eine disjunktive TGD mit zwei Disjunkten, während die einelementige Menge S_R eine einfache TGD liefert.

Das so erzeugte Σ'' besteht demnach aus den folgenden disjunktiven TGDs:

$$\begin{aligned}
Q(z_{Q_1}, z_{Q_2}, z_{Q_3}, z_{Q_4}) \longrightarrow & \exists x_1, x_2 : (A(x_1, x_2) \wedge B(x_1) \\
& \wedge (h(z_{Q_1}) = x_1) \wedge (h(z_{Q_2}) = x_2) \wedge (h(z_{Q_3}) = x_1) \wedge (h(z_{Q_4}) = x_1)) \\
& \vee \exists x_1, x_2, x_3, y : (C(x_1, x_2, x_2, x_3) \\
& \wedge (h(z_{Q_1}) = x_1) \wedge (h(z_{Q_2}) = x_2) \wedge (h(z_{Q_3}) = x_2) \wedge (h(z_{Q_4}) = y)), \\
R(z_{R_1}, z_{R_2}) \longrightarrow & \exists x_1, x_2, x_3, y : (C(x_1, x_2, x_2, x_3) \wedge (h(z_{R_1}) = y) \wedge (h(z_{R_2}) = x_2)).
\end{aligned}$$

Die disjunktiven TGDs wurden im Algorithmus 2 als Nächstes mit den Formeln des dortigen zweiten Schrittes vereinigt und dann um Existenzquantifizierung für einen Homomorphismus und Skolemfunktionen erweitert, um sie in einer Schemaabbildung als Ergebnis auszugeben. Im Algorithmus 3 entfällt das alles. Stattdessen sorgen wir im dritten Schritt unseres Algorithmus dafür, dass wir den Homomorphismus aus den Schemaabbildungen herausziehen können. Um die Skolemfunktionen müssen wir uns in unseren angepassten Algorithmus bereits keine Gedanken mehr machen. Damit wir die Abbildungen im Homomorphismus anwenden können, normalisieren wir die Disjunktionen in den Köpfen der disjunktiven TGD. Die einzelnen S-T TGDs speichern wir in einer Disjunktiven-Menge pro normalisierter disjunktiver TGD, welche in der Menge Ω zusammengefasst werden. Nach dem dritten Schritt enthält Ω die Disjunktiven-Mengen:

$$\begin{aligned}
\omega_1 = \{ & Q(z_{Q_1}, z_{Q_2}, z_{Q_3}, z_{Q_4}) \longrightarrow \exists x_1, x_2 (A(x_1, x_2) \wedge B(x_1) \\
& \wedge (h(z_{Q_1}) = x_1) \wedge (h(z_{Q_2}) = x_2) \wedge (h(z_{Q_3}) = x_1) \wedge (h(z_{Q_4}) = x_1)), \\
& Q(z_{Q_1}, z_{Q_2}, z_{Q_3}, z_{Q_4}) \longrightarrow \exists x_1, x_2, x_3, y : (C(x_1, x_2, x_2, x_3) \\
& \wedge (h(z_{Q_1}) = x_1) \wedge (h(z_{Q_2}) = x_2) \wedge (h(z_{Q_3}) = x_2) \wedge (h(z_{Q_4}) = y))\}, \\
\omega_2 = \{ & R(z_{R_1}, z_{R_2}) \longrightarrow \exists x_1, x_2, x_3, y : (C(x_1, x_2, x_2, x_3) \\
& \wedge (h(z_{R_1}) = y) \wedge (h(z_{R_2}) = x_2))\}.
\end{aligned}$$

Jetzt haben wir in jeder Disjunktiven-Menge einzelne S-T TGDs ohne Disjunktionen im Kopf und somit innerhalb einer S-T TGD nur noch Abbildungen, die keine neue z-Variable auf unterschiedliche Variablen aus den ursprünglichen S-T TGD in Σ abbilden. Das war in den disjunktiven TGDs in Σ'' noch nicht der Fall, da dort z. B. in der ersten disjunktiven TGD mit $h(z_{Q_3}) = x_1$ und $h(z_{Q_3}) = x_2$ die neue Variable z_{Q_3} auf unterschiedliche Variablen aus Σ abgebildet wurde. Mit unserer Vorbereitung können wir nun im Schritt 4 unseres Algorithmus die Abbildungen des Homomorphismus in jeder einzelnen S-T TGD in den Disjunktiven-Mengen anwenden und so die einzelnen Rümpfe der S-T TGD anpassen. Nach der Anwendung des Homomorphismus entsteht so eine Menge Ω' mit Disjunktiven-Mengen ohne Abbildungen unter einem Homomorphismus in den S-T TGDs:

$$\begin{aligned}
\omega'_1 = \{ & Q(x_1, x_2, x_1, x_1) \longrightarrow A(x_1, x_2) \wedge B(x_1), \\
& Q(x_1, x_2, x_2, y) \longrightarrow \exists x_3 : C(x_1, x_2, x_2, x_3)\}, \\
\omega'_2 = \{ & R(y, x_2) \longrightarrow \exists x_1, x_3 : C(x_1, x_2, x_2, x_3)\}.
\end{aligned}$$

Die Menge Ω' ist das eigentliche Ergebnis des Algorithmus 3, welche wir nun in der inversen Schemaabbildung $\mathcal{M}' = (T, S, \Omega')$ ausgeben können. Zur Erinnerung, das Ergebnis des Algorithmus 2 liefert beim gleichen Beispiel eine Formel in existenzieller Logik 2. Ordnung über einem Homomorphismus und einer Menge von Skolemfunktionen. Unsere Disjunktiven-Mengen bieten im Vergleich dazu deutlich einfachere Weiterverwendungsmöglichkeiten.

Algorithmus direkte Invertierung Basierend auf dem Algorithmus 3, welchen wir nach der Vorlage vom Algorithmus 2 entworfen haben, können wir nun eine direkte Berechnung der Adapted Maximum Extended Recovery angeben. Da wir geklärt haben, welche Form die Ausgabe-Schemaabbildung im Algorithmus 3 für eine Eingabe-Schemaabbildung haben soll, können wir uns nun den Umweg über die disjunktiven TGDs sparen und die Disjunktiven-Mengen direkt aus den S-T TGDs der Eingabe-Schemaabbildung berechnen. Der Algorithmus 4 macht genau das. Im ersten Schritt normalisiert er die S-T TGDs der Eingabe-Schemaabbildung genau wie im Algorithmus 3. Im zweiten Schritt werden die einzelnen S-T TGDs invertiert. Bei der Invertierung werden einfach der Rumpf und der Kopf einer S-T TGD vertauscht und die Allquantoren und Existenzquantoren der Variablen entsprechend angepasst. Nach der Invertierung müssen wir die einzelnen S-T TGDs im 3. Schritt nur noch nach den Relationenschemata ihrer Rumpf-Ausdrücke sortieren und erzeugen so die Disjunktiven-Mengen der Ausgabe-Schemaabbildung. Ein Beispiel für den Algorithmus 4 folgt im nächsten Paragraphen mit dem gleichen Beispiel wie zuvor zu Algorithmus 3.

Der kompakte Algorithmus 4 erzeugt die gleiche Adapted Maximum Extended Recovery wie der komplexere Algorithmus 3. Daher werden wir ihn auch im 5. Kapitel für die Implementierung der Invertierung einer Schemaabbildung nutzen. Damit wir bei der Berechnung der Adapted Max Extended Recovery nicht immer beide Algorithmen erwähnen müssen, beschränken wir uns im weiteren Verlauf dieser Arbeit auf den Algorithmus 4. Ein Beispiel für den Algorithmus 4 ist nach dem folgenden Beispiel für den Algorithmus 3 aufgeführt.

Algorithmus 4 DirectAdaptedMaxExtendedRecovery($\mathcal{M}$)

Input: Schemaabbildung $\mathcal{M} = (S, T, \Sigma)$, wobei Σ eine endliche Menge von S-T TGDs ist.
Output: *Adapted Strong Maximum Extended Recovery* $\mathcal{M}' = (T, S, \Omega)$ von $\mathcal{M}$, wobei Ω eine Menge von Mengen von S-T TGDs (Menge von *Disjunktiven-Mengen*) in Logik erster Ordnung ist.

1. (Normalisiere die Konjunktionen in den Köpfen der S-T TGDs.) Erstelle Σ' aus Σ, indem jede S-T TGD $\alpha \longrightarrow (\beta_1 \wedge ... \wedge \beta_r)$ in Σ durch S-T TGDs der Form $\alpha \longrightarrow \beta_1, ..., \alpha \longrightarrow \beta_r$ ersetzt wird, bei denen β_i im Kopf atomar ist.

2. (Invertiere die S-T TGDs.) Füge für jede normalisierte S-T TGD $\alpha \longrightarrow \beta$ in Σ' die S-T TGD $\beta \longrightarrow \alpha$ zur Menge Σ'' hinzu. In Σ'' wird jede Variable in β zu einer allquantifizierten Variable, während jede Variable in α, die nicht in β vorkommt, zu einer existenzquantifizierten Variable wird.

3. (Sammle die S-T TGDs in Disjunktive-Mengen.) Sei $\Omega = \{\omega_1, ..., \omega_q\}$ eine Menge von Mengen von S-T TGDs (Menge von *Disjunktiven-Mengen*), mit q = Anzahl der Relationenschemata R, über denen Ausdrücke in den Rümpfen der S-T TGDs in Σ'' existieren. Für jedes R in den Ausdrücken der Rümpfe der S-T TGDs entsteht ein $\omega_i \in \Omega$, indem die S-T TGDs in Σ' mit R im Rumpf-Ausdruck zusammengefasst werden.

Return $\mathcal{M}' = (T, S, \Omega)$.

Beispiel direkte Invertierung Um die Vereinfachung des Algorithmus 4 im Gegensatz zum Algorithmus 3 zu zeigen, werden wir das vorherige Beispiel noch einmal mit den Schritten des Algorithmus 4 durchrechnen. Sei wieder die Schemaabbildung $\mathcal{M} = (S, T, \Sigma)$ mit $S = \{A = \{a_1, a_2\}, B = \{b\}, C = \{c_1, c_2, c_3, c_4\}\}$ und $T = \{Q = \{q_1, q_2, q_3, q_4\}, R = \{r_1, r_2\}\}$ gegeben und bestehe Σ aus den zwei S-T TGDs:

$$A(x_1, x_2) \wedge B(x_1) \longrightarrow Q(x_1, x_2, x_1, x_1),$$
$$C(x_1, x_2, x_2, x_3) \longrightarrow \exists y : Q(x_1, x_2, x_2, y) \wedge R(y, x_2).$$

Der erste Schritt des Algorithmus startet wieder mit der Normalisierung der S-T TGDs. Die Menge Σ' der normalisierten S-T TGDs unterschiedet sich nicht von der aus dem vorherigen Beispiel:

$$A(x_1, x_2) \wedge B(x_1) \longrightarrow Q(x_1, x_2, x_1, x_1),$$
$$C(x_1, x_2, x_2, x_3) \longrightarrow \exists y : Q(x_1, x_2, x_2, y),$$
$$C(x_1, x_2, x_2, x_3) \longrightarrow \exists y : R(y, x_2).$$

Im zweiten Schritt des Algorithmus 4 erzeugen wir Σ'' aus Σ', indem wir die S-T TGDs invertieren. Dafür vertauschen wir die Rümpfe und Köpfe aller S-T TGDs. Die Variablen in den ehemaligen Köpfen der S-T TGDs werden im Rumpf der invertierten S-T TGD zu allquantifizierten Variablen. Die Variablen aus den ehemaligen Rümpfen bleiben allquantifiziert, solange die Variablen auch im neuen Rumpf auftauchen. Alle anderen Variablen werden zu existenzquantifizierten Variablen, sodass sich Σ'' wie folgt ergibt:

$$Q(x_1, x_2, x_1, x_1) \longrightarrow A(x_1, x_2) \wedge B(x_1),$$
$$Q(x_1, x_2, x_2, y) \longrightarrow \exists x_3 : C(x_1, x_2, x_2, x_3),$$
$$R(y, x_2) \longrightarrow \exists x_1, x_3 : C(x_1, x_2, x_2, x_3).$$

Die invertierten S-T TGDs müssen jetzt nur noch nach ihrem Relationenschema des Rumpf-Ausdrucks sortiert werden. Diese Sortierung ergibt automatisch die Disjunktiven-Mengen in Ω:

$$\omega_1 = \{Q(x_1, x_2, x_1, x_1) \longrightarrow A(x_1, x_2) \wedge B(x_1),$$
$$\quad Q(x_1, x_2, x_2, y) \longrightarrow \exists x_3 : C(x_1, x_2, x_2, x_3)\},$$
$$\omega_2 = \{R(y, x_2) \longrightarrow \exists x_1, x_3 : C(x_1, x_2, x_2, x_3)\}.$$

Die Menge Ω ist auch hier das eigentliche Ergebnis des Algorithmus 4, welche wir nun in der inversen Schemaabbildung $\mathcal{M}' = (T, S, \Omega)$ ausgeben können. Da Algorithmus 4 für die gleiche Eingabe die gleiche Ausgabe wie der Algorithmus 3 liefert, entspricht Ω genau den Disjunktiven-Mengen aus dem vorherigen Beispiel zu Algorithmus 3.

Im nächsten Abschnitt werden wir den Disjunktiven-CHASE für die Anwendung auf einer Schemaabbildung mit einer Menge von Disjunktiven-Mengen wie $\mathcal{M}' = (T, S, \Omega)$ anpassen. Daran anschließend passen wir auch die Theorie der Sicheren Antworten für inverse Schemaabbildungen an, sodass wir einen *Durchschnitt unter einem Homomorphismus* bilden können und nicht mehr alle Tupel mit Nullwerten aus dem Durchschnitt der Ergebnisinstanzen des Disjunktiven-CHASEs entfernen müssen.

4.2.2. Verarbeitung der Inversen

Um nun auch mit Schemaabbildungen, in denen die Abbildungen in Disjunktiven-Mengen gespeichert sind, weiter arbeiten zu können, müssen wir die bestehende Theorie zum Disjunktiven-CHASE leicht anpassen. In diesem Abschnitt werden wir ebenfalls die Theorie der Sicheren Antworten so erweitern, dass diese besser zur Anwendung im GaLVE-Verfahren geeignet ist.

Disjunktiver-CHASE Im Abschnitt 3.2.4 haben wir beschrieben, wie mit dem originalen Disjunktiven-CHASEs (siehe Definition 3.11 und 3.12) eine disjunktive TGD in eine Instanz eingechaset wird. Das Ergebnis des Disjunktiven-CHASEs ergibt sich dabei aus der Menge der Blätter $\mathcal{I} = \{L_1, ..., L_m\}$ des durch die Disjunktionen aufgespannten CHASE-Baumes. Da die Schemaabbildung im Ergebnis unseres

angepassten Algorithmus 4 nun nicht mehr aus disjunktiven TGDs, sondern aus Disjunktiven-Mengen besteht, müssen wir einen *angepassten Disjunktiven-CHASE-Schritt* und *angepassten Disjunktiven-CHASE* definieren, welche die Disjunktiven-Mengen genauso verarbeiten, wie der normale Disjunktive-CHASE-Schritt und Disjunktive-CHASE die disjunktiven TGDs.

Definition 4.3. (Angepasster Disjunktiver-CHASE-Schritt auf Instanzen): Sei I eine Instanz und ω eine Disjunktive-Menge der Form

$$\{(\phi(x_1) \longrightarrow \exists y_1 : \psi_1(x_1, y_1)), ..., (\phi(x_n) \longrightarrow \exists y_n : \psi_n(x_n, y_n))\}.$$

Sei $G = \{g_1, ..., g_n\}$ die Menge der Homomorphismen $g_i : \phi(x_i) \longrightarrow I$ mit $i \in \{1, ..., n\}$, wobei jedes $g_i \in G$ ein aktiver Trigger für die S-T TGD $\phi(x_i) \longrightarrow \exists y_i : \psi_i(x_i, y_i)$ und I ist. Für jedes i mit $1 \leq i \leq n$, sei I_i die Vereinigung der Tupel in I mit der Menge an Tupel, die durch die Anwendung des Standard-CHASE-Schrittes mit I und $\phi(x_i) \longrightarrow \exists y_i : \psi_i(x_i, y_i)$ entstehen. Wir nennen die Menge $\{I_1, ..., I_n\}$ das Ergebnis des *angepassten Disjunktiven-CHASE-Schrittes* durch die Anwendung von ω auf I und schreiben $I \xrightarrow{\omega,G} \{I_1, ..., I_n\}$. Wenn ω einelementig ist, dann reduziert sich die Menge $\{I_1, ..., I_n\}$ zu einer einzelnen Instanz I'. Der CHASE-Schritt wird dann als $I \xrightarrow{\omega,g} I'$ geschrieben.

Definition 4.4. (Angepasster Disjunktiver-CHASE auf Instanzen): Sei Ω eine endliche Menge Disjunktiver-Mengen und I eine Instanz. Der *angepasste Disjunktive-CHASE* auf I mit Ω ist ein endlicher Baum mit I als Wurzel. Für jeden Knoten I' gilt, wenn I' die Kinder $\{I_1, ..., I_p\}$ hat, muss $I' \xrightarrow{\omega,G} \{I_1, ..., I_p\}$ für ein $\omega \in \Omega$ und die Homomorphismen in G gelten. Jedes Blatt L im Baum erfüllt die Bedingung, dass es kein $\omega \in \Omega$ und keine Menge von Homomorphismen G gibt, sodass ω mit G auf L angewendet werden kann. Die Menge der Blätter des CHASE-Baumes $\mathcal{I} = \{L_1, ..., L_m\}$ nennen wir das Ergebnis des Disjunktiven-CHASE auf I mit Ω.

Im Gegensatz zum normalen Disjunktiven-CHASE übergeben wir dem angepassten Disjunktiven-CHASE mit den Disjunktiven-Mengen nur S-T TGDs. Daraus folgt, dass der angepasste Disjunktive-CHASE immer terminiert und somit immer einen endlichen CHASE-Baum generiert. Ein Beispiel für den Aufbau des CHASE-Baumes im angepassten Disjunktiven-CHASE folgt im nächsten Paragraphen.

Da wir weder in der originalen, noch in der angepassten Disjunktiven-CHASE-Definition EGDs beachten, kann der Disjunktive-CHASE auch nicht fehlschlagen. Das vereinfacht nun die Definition des *Ergebnisses für Schemaabbildungen mit Hilfe des angepassten Disjunktiven-CHASEs* im Vergleich zur äquivalenten Definition mit dem Standard-CHASE (siehe Definition 2.19). Mit der folgenden Definition schränken wir nun eine Menge von Instanzen im Ergebnis einer Schemaabbildung auf die Relationen ein, die aus dem Ziel der Schemaabbildung stammen.

Definition 4.5. (Ergebnis für Schemaabbildungen mit Hilfe des angepassten Disjunktiven-CHASEs): Sei $\mathcal{M} = (S, T, \Omega)$ eine Schemaabbildung mit Quellschema $S = \{R_{S1}, ..., R_{Sp}\}$, Zielschema $T = \{R_{T1}, ..., R_{Tq}\}$ mit $S \cap T = \emptyset$ und einer Menge Ω von Disjunktiven-Mengen mit S-T TGDs vom Quellschema zum Zielschema. Sei außerdem $\mathcal{I}_{S \cup T} = \{L_{1_{S \cup T}}, ..., L_{m_{S \cup T}}\}$ die Menge an Ergebnisinstanzen (unter $S \cup T$) des angepassten Disjunktiven-CHASEs nach Definition 4.4 auf einer Quellinstanz I_S unter Quellschema S mit den Disjunktiver-Mengen in Ω. Die Menge von Ergebnisinstanzen $\mathcal{I}_T = \{L_{1_T}, ..., L_{m_T}\}$ von $\mathcal{M}$ ergibt sich durch das Einschränken von den Instanzen in $\mathcal{I}_{S \cup T}$ auf Relationen über dem Ziel-Schema T:

$$L_{j_T} = r_i(R_i) \in L_{j_{S \cup T}} : R_i \in T, i \in \{1, ..., p + q\}, j \in \{1, ..., m\}.$$

Beispiel Disjunktiver-CHASE Betrachten wir nun erneut das Beispiel zum Algorithmus 2 aus Abschnitt 3.2.4, welches wir auch im letzten Abschnitt für den Algorithmus 4 verwendet haben. Sei wieder die Schemaabbildung $\mathcal{M} = (S, T, \Sigma)$ mit den Schemata $S = \{A = \{a_1, a_2\}, B = \{b\}, C = \{c_1, c_2, c_3, c_4\}\}$ und $T = \{Q = \{q_1, q_2, q_3, q_4\}, R = \{r_1, r_2\}\}$ gegeben und sei $\mathcal{M}' = (T, S, \Omega)$ mit $\Omega = \{\omega_1, \omega_2\}$, das Ergebnis von DirectAdaptedMaxExtendedRecovery($\mathcal{M}$), wie wir es im letzten Abschnitt berechnet haben. Sei außerdem die Instanz I gegeben und sei J die Ergebnisinstanz aus der Anwendung des CHASEs mit $\mathcal{M}$ und I. Die S-T TGDs in Σ, die Instanzen I, J und die Disjunktiven-Mengen $\omega_1, \omega_2 \in \Omega$ sind wie folgt definiert:

$$\begin{aligned}\Sigma = \{&A(x_1, x_2) \wedge B(x_1) \longrightarrow Q(x_1, x_2, x_1, x_1),\\ &C(x_1, x_2, x_2, x_3) \longrightarrow \exists y : Q(x_1, x_2, x_2, y) \wedge R(y, x_2)\},\\ I = \{&A(1,2), B(1), C(1,2,2,3)\},\end{aligned}$$

$$\begin{aligned}CHASE_{\mathcal{M}}(I) = J = \{&Q(1,2,1,1), Q(1,2,2,\eta_1), R(\eta_1, 2)\},\\ \omega_1 = \{&Q(x_1, x_2, x_1, x_1) \longrightarrow A(x_1, x_2) \wedge B(x_1),\\ &Q(x_1, x_2, x_2, y) \longrightarrow \exists x_3 : C(x_1, x_2, x_2, x_3)\},\\ \omega_2 = \{&R(y, x_2) \longrightarrow \exists x_1, x_3 : C(x_1, x_2, x_2, x_3)\}.\end{aligned}$$

Für die Berechnung von $CHASE_{\mathcal{M}'}(J)$ müssen wir nun unseren angepassten Disjunktiven-CHASE verwenden und fangen damit an, einen CHASE-Baum mit der Wurzel J aufzubauen. Als Nächstes suchen wir nach Abbildungen von den Rümpfen der S-T TGD in ω_1 und ω_2 nach J. Betrachten wir ω_1, so finden wir für jede S-T TGD in ω_1 jeweils einen Trigger: Zum einen für die erste S-T TGD und das Tupel $Q(1,2,1,1)$ den Trigger $g_{1.1} : Q(x_1, x_2, x_1, x_1) \longrightarrow Q(1,2,1,1)$ und zum anderen für die zweite S-T TGD und das Tupel $Q(1,2,2,\eta_1)$ den Trigger $g_{1.2} : Q(x_1, x_2, x_2, y) \longrightarrow Q(1,2,2,\eta_1)$. Für die Disjunktive-Menge ω_2 finden wir für ihre eine S-T TGD ebenfalls einen Trigger, und zwar für das Tupel $R(\eta_1, 2)$ mit $g_2 : R(y, x_2) \longrightarrow R(\eta_1, 2)$. Alle drei Trigger sind aktiv, da es keine Erweiterung von $g_{1.1}$, $g_{1.2}$ oder g_2 gibt, die auch den Kopf ihrer jeweiligen S-T TGD auf J abbildet (J besteht nur aus Tupel über Q und R und kann somit nicht um Abbildungen von A, B oder C auf J erweitert werden).

Da die Kinder eines Knotens im CHASE-Baum jeweils nur für eine Disjunktive-Menge und ihre zugehörigen Trigger gebildet werden, müssen wir eine Disjunktive-Menge mit ihren Triggern für die Anwendung auf unserer Instanz J auswählen. Zur Berechnung der ersten CHASE-Baum-Ebene wählen wir die Disjunktive-Menge ω_1 mit den Triggern $g_{1.1}$ und $g_{1.2}$ und führen den angepassten Disjunktiven-CHASE-Schritt $J \xrightarrow{\omega_1, G_1} \{J_{1.1}, J_{1.2}\}$ mit $G_1 = \{g_{1.1}, g_{1.2}\}$ aus. Die Instanz $J_{1.1}$ ergibt sich aus der Anwendung des Standard-CHASE-Schrittes mit der S-T TGD $Q(x_1, x_2, x_1, x_1) \longrightarrow A(x_1, x_2) \wedge B(x_1)$ und dem Trigger $g_{1.1}$ auf J und die Instanz $J_{1.2}$ ergibt sich aus der Anwendung des Standard-CHASE-Schrittes mit der S-T TGD $Q(x_1, x_2, x_2, y) \longrightarrow \exists x_3 : C(x_1, x_2, x_2, x_3)$ und dem Trigger $g_{1.2}$ auf J:

$$\begin{aligned}J_{1.1} =&\{Q(1,2,1,1), Q(1,2,2,\eta_1), R(\eta_1, 2), A(1,2), B(1)\},\\ J_{1.2} =&\{Q(1,2,1,1), Q(1,2,2,\eta_1), R(\eta_1, 2), C(1,2,2,\eta_2)\}.\end{aligned}$$

Im zweiten Durchlauf des Disjunktiven-CHASEs wählen wir jetzt den Trigger g_2 der Disjunktiven-Menge ω_2 und wenden ihn auf alle Instanzen in der ersten CHASE-Baum-Ebene an. Da der Trigger g_2 allerdings nur noch für die Instanz $J_{1.1}$ aktiv[3] ist, führen wir dementsprechend für die zweite CHASE-Baum-Ebene nur den Disjunktiven-CHASE-Schritt $J_{1.1} \xrightarrow{\omega_2, G_2} \{J_{2.1}\}$ mit $G_2 = \{g_2\}$ aus.

[3] Für ω_2 und $J_{1.2}$ ist g_2 kein aktiven Trigger, da mit dem Tupel $C(1,2,2,\eta_2)$ eine Erweiterung von g_2 von dem Kopf der S-T TGD in ω_2 auf die Instanz $J_{1.2}$ möglich ist.

Die Instanz $J_{2.1}$ ergibt sich dementsprechend aus der Anwendung des Standard-CHASE-Schrittes mit der S-T TGD $R(y, x_2) \longrightarrow \exists x_1, x_3 : C(x_1, x_2, x_2, x_3)$ und dem Trigger g_2 auf $J_{1.1}$:

$$J_{2.1} = \{Q(1,2,1,1), Q(1,2,2,\eta_1), R(\eta_1,2), A(1,2), B(1), C(\eta_2,2,2,\eta_3)\}.$$

Für die Instanzen $J_{1.2}$ und $J_{2.1}$ existierten nun keine aktiven Trigger mehr[4], sodass wir alle Blätter des CHASE-Baumes gefunden haben. Das Endergebnis erhalten wir, indem die Instanzen $J_{2.1}$ und $J_{1.2}$ auf die Tupel über dem Zielschema einschränken. Die Menge der Instanzen im Ergebnis lautet:

$$\mathcal{I} = \{\{A(1,2), B(1), C(\eta_2,2,2,\eta_3)\}, \{C(1,2,2,\eta_2)\}\}.$$

Im Beispiel zum Disjunktiven-CHASE im Abschnitt 3.2.4 haben wir nach der Berechnung der Menge der Instanzen direkt die Sichere Antwort für eine Anfrage q, die die ganze Instanz zurückgibt, berechnet. Bilden wir hier den Durchschnitt der Instanzen in $\mathcal{I}$, bleiben in der Schnittmenge höchstens die Tupel der C-Relation übrig und dass auch nur dann, wenn wir die Nullwerte mit Konstanten gleichsetzen könnten. Selbst wenn wir dies zulassen würden, entfernen wir in der bisherigen Theorie der Sicheren Antworten nach der Durchschnittsbildung alle Tupel mit Nullwerten aus dem Ergebnis. Für die Sicheren Antworten von q und $\mathcal{I}$ ergibt sich somit $certain_{e(\mathcal{M})\circ e(\mathcal{M}')}(q, I) = \emptyset$.

Im nächsten Paragraphen werden wir das Problem mit den Nullwerten im Ergebnis des Disjunktiven-CHASEs und den Sicheren Antworten genauer betrachten. Wir werden eine Lösung vorstellen, die mit einem *Durchschnitt unter einem Homomorphismus* eine *Sichere Instanz* berechnen kann, welche auch Tupel mit Nullwerten enthalten kann.

Sichere Instanz Der angepasste Disjunktive-CHASE liefert genauso wie der Disjunktive-CHASE eine Menge von Instanzen $\mathcal{I} = \{L_1, ..., L_m\}$ als Ergebnis. Jede Instanz $L \in \mathcal{I}$ stellt eine mögliche Lösung für eine Schemaabbildung dar. In unserem Anwendungsfall für das GaLVE-Verfahren enthält $\mathcal{I}$ die möglichen Lösungen einer inversen Schemaabbildung, angewendet auf eine von mehreren Schemaabbildungen aufgebaute globale Datenbank zur Datenintegration. Um nun bei Anfragen an $\mathcal{I}$ gültige Ergebnisse zu garantieren, haben wir gezeigt, wie mit den Sicheren Antworten (siehe Definition 3.13) eine Art Durchschnitt auf den Instanzen in $\mathcal{I}$ gebildet werden kann. In unserer Verwendung der Sicheren Antworten haben wir die Anfrage in der Definition der Sicheren Antworten so spezifiziert, dass sie die gesamte Instanz zurückgibt. Die so erzeugte Instanz nennen wir nun die *Sichere Instanz* von $\mathcal{I}$.

Leider hat die bisherige Definition der Sicheren Antworten eine Eigenschaft, die nicht verträglich mit unseren Erwartungen an das GaLVE-Verfahren ist. So besagt die in [FKPT11a] an die Verwendung mit einer Maximum Extended Recovery angepasste Berechnung der Sicheren Antwort, dass wir das Ergebnis der Durchschnittsbildung von den Instanzen auf die Tupel ohne Nullwerte einschränken müssen. Dieser Wegfall von Tupeln mit Nullwerten ist gerade bei Schemaabbildungen für die Datenintegration ein großer Nachteil, da wir hier durchaus Datensätze erwarten können, in denen jedes Tupel in mindestens einem Attribut einen Nullwert enthält. Unser Ziel muss es also sein, Sichere Antworten bzw. Sichere Instanzen zu berechnen, in denen Tupel mit Nullwerten enthalten bleiben.

Für das Beispiel des angepassten Disjunktiven-CHASE aus dem vorherigen Paragraphen haben wir bereits festgestellt, dass wir aufgrund der Nullwerte in den Tupeln die leere Menge als Ergebnis der Sicheren Antwort erhalten. Unser Problem ist jetzt, dass alle Instanzen ein Tupel der C-Relation enthalten, welches wir auf das originale Tupel der Ausgangsinstanz abbilden können. Diese Tupel der C-Relation liefern uns

[4] Wir müssen für jede S-T TGD in einer Disjunktiven-Menge einen aktiven Trigger finden, um einen Disjunktiven-CHASE-Schritt ausführen zu können.

also Informationen, die wir schon vor den Schemaabbildungen hatten und sollten daher in gewisser Weise in den Sicheren Antworten auftauchen:

$$I = \{A(1,2), B(1), C(1,2,2,3)\},$$
$$\mathcal{I} = \{\{A(1,2), B(1), C(\eta_1,2,2,\eta_2)\},$$
$$\{C(1,2,2,\eta_1)\}\},$$

$$\begin{aligned} certain_{e(\mathcal{M})\circ e(\mathcal{M}')}(q,I) = \emptyset & \quad (\text{ 1. Ergebnis nach Definition ohne Nullwerte}) \\ \vee = \{C(\eta_1,2,2,\eta_2)\} & \quad (\text{ 2. Ergebnis mit den wenigsten Informationen}) \\ \vee = \{C(1,2,2,\eta_1)\} & \quad (\text{ 3. Ergebnis mit den meisten Informationen}). \end{aligned}$$

Wenden wir die originale Definition der Sicheren Antworten an (siehe Definition 3.13) erhalten wir nur die leere Menge im Ergebnis. Nun können wir aber auch sagen, dass wir von dem C-Tupel im 2. Ergebnis wissen, dass es in gewisser Form in allen Instanzen in $\mathcal{I}$ auftaucht und auch als Sichere Antwort gelten sollte. Das C-Tupel im 3. Ergebnis enthält dabei sogar noch mehr Informationen als das C-Tupel im 2. Ergebnis. Dabei gilt, dass alle C-Tupel in den Instanzen in $\mathcal{I}$ auf das C-Tupel im 3. Ergebnis abgebildet werden können.

In der folgenden Definition werden wir die Berechnung des *Durchschnitts unter einem Homomorphismus* vorstellen. Mit ihr werden wir Ergebnisse wie das C-Tupel im 3. Ergebnis in den Sicheren Antworten bzw. der Sicheren Instanzen ausgeben können.

Definition 4.6. (Durchschnitt unter einem Homomorphismus): Der *Durchschnitt unter einem Homomorphismus* h (geschrieben: $\widetilde{\bigcap}^h$) übernimmt bei der Durchschnittsbildung über einer Menge von Instanzen auch die Tupel mit ins Ergebnis, die sich mit einem Homomorphismus (siehe Definition 2.9) aufeinander abbilden lassen. Dabei muss ein Homomorphismus für alle Tupel im Ergebnis des Durchschnitts gelten. Sei die Menge von Instanzen $\mathcal{I} = \{L_1, ..., L_m\}$ mit disjunkten Nullwerten unter den Instanzen geben. Seien außerdem die Tupel $[t_{1_1}, ..., t_{1_n}] \in L_1, ..., [t_{m_1}, ..., t_{m_n}] \in L_m$ gegeben. Dann besteht die Ergebnisinstanz $I_{\cap^h}$ des Durchschnitts unter dem Homomorphismus h aus den Tupeln $t_1, ..., t_l$, wenn $\exists h \forall o, j \exists k : h(t_{j_k}) = t_o, o \in \{1, ..., l\}, j \in \{1, ..., m\}, k \in \{1, ..., j_n\}$ gilt[5] und h minimal[6] ist.

Mit dem Durchschnitt unter einem Homomorphismus können wir nun die *angepassten Sicheren Antworten* und die *Sicheren Instanzen* definieren. Diese werden wir gleich an eine Verwendung im GaLVE-Verfahren anpassen, bei der die globale Datenbank durch mehrere Schemaabbildungen aufgebaut wird.

Definition 4.7. (Angepasste Sichere Antworten, Sichere Instanz): Seien die durch S-T TGDs spezifizierte Schemaabbildungen $\mathcal{M}_1 = (S_1, T, \Sigma_1), ..., \mathcal{M}_n = (S_n, T, \Sigma_n)$ von möglicherweise verschiedenen Quellschemata $S_1, ..., S_n$ auf ein gemeinsames globales Schema T gegeben. Seien $I_1, ..., I_n$ jeweils Instanzen unter den einzelnen Schemata $S_1, ..., S_n$ und J die globale Instanz unter dem Schema T, wobei J aus der Anwendung des CHASEs mit den Schemaabbildungen $\mathcal{M}_1, ..., \mathcal{M}_n$ auf den entsprechenden Instanzen $I_1, ..., I_n$ entstanden ist[7]. Sei $\mathcal{M}'_i = (T, S_i, \Omega_i)$ die Adapted Strong Maximum Extended Recovery einer Schemaabbildung $\mathcal{M}_i$ mit $i \in \{1, ..., n\}$ und die Menge an Instanzen $\mathcal{I} = \{L_1, ..., L_m\}$ das Ergebnis des angepassten Disjunktiven-CHASEs mit Ω_i und der globalen Instanz[8] J. Dann ist die *angepasste Sichere*

[5] Es existiert ein Homomorphismus, sodass für jedes Tupel der Ergebnisinstanz in jeder an der Durchschnittsbildung beteiligten Instanz ein Tupel existiert, das auf das Tupel der Ergebnisinstanz abgebildet werden kann

[6] Der Homomorphismus enthält nur die Abbildungen, die nötig sind, um die Tupel aufeinander abzubilden.

[7] Da $\mathcal{M}_1, ..., \mathcal{M}_n$ jeweils auf demselben Zielschema T definiert sind, können wir $J = CHASE_{\mathcal{M}_1,...,\mathcal{M}_n}(I_1, ..., I_n)$ schreiben.

[8] Da $\mathcal{M}_1, ..., \mathcal{M}_n$ jeweils auf demselben Zielschema T definiert sind und T das Quellschema von $\mathcal{M}'_i$ ist, können wir $\mathcal{I} = CHASE_{\mathcal{M}'_i}(CHASE_{\mathcal{M}_1,...,\mathcal{M}_n}(I_1, ..., I_n))$ mit $i \in \{1, ..., n\}$ schreiben.

Antwort einer Konjunktiven-Anfrage q über dem Schema S_i auf $\mathcal{I}$

$$certain_{e(\mathcal{M}_1,...,\mathcal{M}_n)\circ e(\mathcal{M}'_i)}(q, I_1, ..., I_n) = \left(\widetilde{\bigcap}^h_{L\in\mathcal{I}} q(L)\right)$$

mit $i \in \{1, ..., n\}$, wobei $\widetilde{\bigcap}^h$ den Durchschnitt unter einem Homomorphismus bildet.
Liefert die Anfrage q die gesamte Instanz, sprechen wir im Kontext des GaLVE-Verfahrens anstatt von der angepassten Sicheren Antwort auch von der *Sicheren Instanz* der inversen Schemaabbildung $\mathcal{M}'_i$. Wollen wir ausdrücken, dass wir eine Sichere Instanz berechnen, lassen wir die Anfrage q in der Durchschnittsbildung weg und schreiben

$$certain_{e(\mathcal{M}_1,...,\mathcal{M}_n)\circ e(\mathcal{M}'_i)}(q, I_1, ..., I_n) = \left(\widetilde{\bigcap}^h_{L\in\mathcal{I}} L\right)$$

mit $i \in \{1, ..., n\}$, wobei $\widetilde{\bigcap}^h$ den Durchschnitt unter einem Homomorphismus bildet.
Betrachten wir nur eine Schemaabbildung $\mathcal{M}$ mit ihrer Adapted Strong Maximum Extended Recovery $\mathcal{M}'$, dann schreiben wir vereinfacht

$$certain_{e(\mathcal{M})\circ e(\mathcal{M}')}(q, I) = \left(\widetilde{\bigcap}^h_{L\in\mathcal{I}} q(L)\right),$$

wobei die Instanz I unter dem Quellschema von $\mathcal{M}$ definiert ist und $\widetilde{\bigcap}^h$ den Durchschnitt unter einem Homomorphismus bildet.

Betrachten wir erneut das Anfangsbeispiel aus diesem Paragraphen (diesmal mit disjunkten Nullwerten unter den Instanzen, dargestellt mit der Menge $\mathcal{I}_1$), dann berechnet der Durchschnitt unter einem Homomorphismus genau das C-Tupel, welche wir als das mit den meisten Informationen identifiziert haben. Das liegt wie schon zuvor vermutet daran, dass in jeder Instanz in $\mathcal{I}_1$ ist ein Tupel existiert, das auf $C(1, 2, 2, \eta_3)$ abgebildet werden kann:

$$\mathcal{I}_1 = \{\{A(1,2), B(1), C(\eta_1, 2, 2, \eta_2)\}, \{C(1,2,2,\eta_3)\}\}, \qquad \widetilde{\bigcap}^h_{L\in\mathcal{I}_1} L = \{C(1,2,2,\eta_3)\}.$$

Für die Menge von Instanzen $\mathcal{I}_2$ ist der Durchschnitt unter einem Homomorphismus leer, obwohl in jeder Instanz ein C-Tupel existiert. Die C-Tupel der ersten und der dritten Instanz können sogar jeweils auf das C-Tupel der zweiten Instanz abgebildet werden. Das Problem ist, dass die C-Tupel der ersten und der dritten Instanz nicht aufeinander abgebildet werden können:

$$\mathcal{I}_2 = \{\{A(1,2), B(1), C(\eta_1, 2, 2, \eta_2)\}, \{C(\eta_3, 2, \eta_4, \eta_3)\}, \{C(1, 2, \eta_5, \eta_6)\}\}, \qquad \widetilde{\bigcap}^h_{L\in\mathcal{I}_2} L = \emptyset.$$

In $\mathcal{I}_3$ können wir bei der Bildung des Durchschnitts unter einem Homomorphismus zwar das A-Tupel und das B-Tupel der zweiten Instanz auf die entsprechenden A- und B-Tupel der ersten Instanz abbilden, es existiert aber kein Homomorphismus, der η_1 sowohl auf die Konstante 2 als auch auf die Konstante 1 abbilden kann. Da es keinen gemeinsamen Homomorphismus für die Abbildungen von den A- und den B-Tupeln existiert, ist auch hier der Durchschnitt der Instanzen leer:

$$\mathcal{I}_3 = \{\{A(1,2), B(1)\}, \{A(1,\eta_1), B(\eta_1)\}\}, \qquad \widetilde{\bigcap}^h_{L\in\mathcal{I}_3} L = \emptyset.$$

Algorithmus Sichere Instanz Da wir den Durchschnitt unter einem Homomorphismus ausschließlich für die Berechnung der Sicheren Instanzen verwenden werden, werden wir die Sichere Instanz als Synonym für Durchschnitt unter einem Homomorphismus verwenden. So berechnet der Algorithmus 5 die Sichere Instanz für eine Menge von Instanzen $\mathcal{I}$, indem der Durchschnitt unter einem Homomorphismus von $\mathcal{I}$ nach Definition 4.6 berechnet wird.

Algorithmus 5 Sichere Instanz($\mathcal{I}$)

Input: Menge von Instanzen $\mathcal{I} = \{L_1, ..., L_m\}$ mit disjunkten Nullwerten untereinander.
Output: Sichere Instanz = Ergebnisinstanz $I_{\cap^h}$ des Durchschnitts unter einem Homomorphismus.

```
1: Homomorphismus h, h_∅ = ∅
2: Minimal erweiterter Homomorphismus h_+ = ∅, mit h ⊆ h_+
3: Boolean found
4: for all Tupel t_j ∈ L_1 [loop-1] do
5:    for all Instanzen L_i ∈ {L_2, ..., L_m} [loop-2] do
6:       found = false
7:       for all Tupel t_k ∈ L_i [loop-3] do
8:          if ∃h_∅ : h_∅(t_k) = t_j ∨ h_∅(t_j) = t_k und in Konflikt stehende Nullwerte ∈ t_j ∨ ∈ t_k then
9:             Lösche Tupel mit in Konflikt stehenden Nullwerten (t_j aus L_1 ∨ t_k aus L_i)
10:            h += h_∅
11:            I_∩h, h_∅ = ∅
12:            Starte loop-1 erneut mit geänderten Instanzen und bereits erzeugtem h
13:         if ∃h_+ : h_+(t_k) = t_j ∨ h_+(t_j) = t_k then
14:            h = h_+
15:            found = true
16:         else if ∃h_∅ : h_∅(t_k) = t_j ∨ h_∅(t_j) = t_k then
17:            Lösche Tupel mit in Konflikt stehenden Nullwerten (t_j aus L_1 ∨ t_k aus L_i)
18:            h += h_∅
19:            I_∩h, h_∅ = ∅
20:            Starte loop-1 erneut mit geänderten Instanzen und bereits erzeugten h
21:      if found = false then
22:         continue loop-1
23:   I_∩h += h(t_j)
24: I_∩h = h(I_∩h)
25: Return I_∩h
```

[Eingabe/Ausgabe] Als Eingabe erhält der Algorithmus eine Menge von Instanzen. In unseren Anwendungsfällen werden das die Instanzen sein, die der angepasste Disjunktive-CHASE für eine Menge von Disjunktiven-Mengen einer inversen Schemaabbildung und der globalen Datenbank berechnet. Die Ausgabe des Algorithmus ist die Instanz, die den Durchschnitt aller Instanzen unter einem Homomorphismus darstellt, sprich die Sichere Instanz aus unseren Eingabeinstanzen.

[Zeile 1-3] In Zeile 1 des Algorithmus definieren wir zwei Homomorphismen. Zum einen den Homomorphismus h, der die Abbildungen der Tupel in den Eingabeinstanzen auf die Tupel in der Ergebnisinstanz speichert und zum anderen den Homomorphismen $h_\emptyset$, der vor jeder Verwendung auf die leere Menge von Abbildungen gesetzt wird. In der 2. Zeile definieren wir einen weiteren Homomorphismus h_+. Dieser hat die Eigenschaft, dass er immer als minimale Erweiterung von h fungiert und daher immer eine Obermenge von h ist. Die Minimalität der Erweiterung bedeutet, dass zu h_+ nur die absolut nötigen Abbildungen hinzugefügt werden, mit denen zwei Tupel aufeinander abgebildet werden können. Die letzte Variable für den Algorithmus ist das Boolean found in Zeile 3, welches angibt, ob ein passendes Tupel in zwei Instanzen gefunden wurde.

[Zeile 4-7] Die erste Schleife, die wir durchlaufen, iteriert über alle Tupel einer ausgewählten Instanz der Eingabeinstanzen. Wir bezeichnen diese Schleife in Zeile 4 des Algorithmus als loop-1, um sie besser von den kommenden Schleifen unterscheiden zu können. Da der Durchschnitt unter einem Homomorphismus letztendlich über allen Instanzen gebildet wird, ist die Wahl der Instanz aus den Eingabeinstanzen in

loop-1 nicht ausschlaggebend für das Ergebnis des Algorithmus. In praktischen Anwendungen könnte man die Iterationen in loop-1 dadurch minimieren, dass man die Instanz mit den wenigsten Tupeln wählt. Im Ergebnis des Algorithmus können unabhängig von der Wahl der Instanz in loop-1 sowieso maximal so viele Tupel enthalten sein wie in der Instanz mit den wenigsten Tupeln. Um uns im Algorithmus auf das Wichtige zu konzentrieren, wählen wir eine beliebige Instanz aus den Eingabeinstanzen und iterieren dann im nächsten Schleifendurchlauf über die restlichen Instanzen aus den Eingabeinstanzen. Diese Schleife in der Zeile 5 benennen wir mit loop-2. Da wir mit loop-2 die Suche nach einem Tupel in einer neuen Instanz beginnen, müssen wir in Zeile 6 found auf false setzen. In der darauf folgenden innersten Schleife loop-3 in Zeile 7 iterieren wir dann über alle Tupel der Instanz aus loop-2.

[Zeile 8-12] In der 8. Zeile prüfen wir zuerst, ob unsere Tupel aus loop-1 und loop-3 aufeinander abgebildet werden könnten und ob eines von ihnen einen Nullwert enthält, welcher im Homomorphismus bereits auf unterschiedliche Konstanten abgebildet wird. Ist das der Fall, bedeutet das, dass wir zuvor bereits Tupel gefunden und gelöscht haben, die widersprüchliche Abbildungen von denselben Nullwerten erzeugt haben. Die aktuell betrachteten Tupel würden nun den für die gleichen Nullwerte einen Widerspruch erzeugen. Wir können daher in Zeile 9 das Tupel mit den widersprüchlichen Nullwerten aus seiner Instanz löschen. Es reicht dabei, das Tupel nur aus der einen Instanz zu löschen, da es so nicht mehr im Durchschnitt vorkommen kann. Damit wir in der weiteren Berechnung wirklich alle Widersprüche identifizieren können, müssen wir den Homomorphismus h in Zeile 10 auch mit den Abbildungen aus $h_\emptyset$ erweitern. In Zeile 11 kann der Homomorphismus $h_\emptyset$ wieder auf die leere Menge von Abbildungen gesetzt werden. Da wir einen Widerspruch gefunden haben, müssen wir die Berechnung der Sicheren Instanz neu starten. Dafür entfernen wir in Zeile 11 auch alle bereits gefundenen Tupel aus der Ergebnisinstanz und starten in Zeile 12 erneut mit loop-1.

[Zeile 13-15] In Zeile 13 prüfen wir nun, ob wir unseren Homomorphismus h so erweitern können (ob ein Homomorphismus h_+ existiert), dass dieser entweder das Tupel aus loop-1 auf das Tupel aus loop-3 abbildet oder umgekehrt. Ist das der Fall, erweitern wir in Zeile 14 den Homomorphismus h um die entsprechenden Abbildungen. Da wir somit mindestens ein Tupel in der Instanz aus loop-2 gefunden haben, das auf das Tupel aus loop-1 abgebildet werden kann (oder umgekehrt), können wir die Variable found in Zeile 15 auf true setzen. Die Suche nach passenden Tupeln in loop-3 brechen wir mit dem Finden eines passenden Tupels nicht ab. In der Instanz aus loop-2 können noch weitere Tupel existieren, die auf das Tupel in loop-1 abgebildet werden können (oder umgekehrt). In diesen Fällen müssen wir unseren Homomorphismus h auch um die Abbildungen für diese Tupel erweitern, um dann auch wirklich alle möglichen Informationen in die Ergebnisinstanz übernehmen zu können (z. B. erzeugt das erste Tupel Abbildung von einem Nullwert auf einen anderen Nullwert und das nächste gefundene Tupel Abbildung von dem Nullwert auf eine Konstante).

[Zeile 16-20] In der 16. Zeile prüfen wir, ob h in Zeile 13 nicht erweitert werden konnte, weil in h bereits Abbildungen enthalten sind, die zu Widersprüchen führen würden. Dafür versuchen wir mit $h_\emptyset$ einen frischen Homomorphismus zwischen den beiden untersuchten Tupel aufzubauen. So ein $h_\emptyset$ finden wir nur, wenn die Tupel eigentlich aufeinander abgebildet werden könnten, aber Abbildungen in h diese neuen Abbildungen stören. Im Detail tritt das Problem auf, wenn ein Nullwert aus einer der untersuchten Instanzen auf zwei verschiedene Konstanten abgebildet werden soll. In h wird der Nullwert dann bereits auf eine Konstante abgebildet und in $h_\emptyset$ soll der Nullwert auf eine andere Konstante abgebildet werden. Diesen Fall werden wir auch noch im nachfolgenden Beispiel behandeln. Wir identifizieren die widersprüchlichen Nullwerte, indem wir alle Nullwerte (optimalerweise die Nullwerte, die auch in beiden Homomorphismen abgebildet werden) darauf untersuchen, ob wir einen Homomorphismus zwischen ihren jeweiligen Abbildungen aus h und $h_\emptyset$ finden können[9]. Ist das nicht der Fall, dann werden die Nullwerte mit h auf eine andere Konstante als mit $h_\emptyset$ abgebildet.

[9] Widersprüchliche Abbildung der Nullwerte $\eta_1, ..., \eta_n$ mit $\forall o \in \{1, ..., n\} \nexists g : g(h(\eta_o)) = h_\emptyset(\eta_o) \vee g(h_\emptyset(\eta_o)) = h(\eta_o)$.

Sollte der beschriebene Sonderfall auftreten, müssen wir das störende Tupel mit den Nullwerten aus der entsprechenden Instanz löschen. Es reicht dabei in Zeile 17 das Tupel aus der einen Instanz zu löschen, da es so nicht mehr im Durchschnitt vorkommen kann. Weitere Tupel mit den widersprüchlichen Nullwerten löschen wir nicht, da diese auch noch andere Nullwerte enthalten können, die an späterer Stelle zu Widersprüchen führen könnten. Das Tupel, das die bereits vorliegende Abbildung des Widerspruches hinzugefügt hat, wird in einem späteren Durchlauf entfernt. Damit wir diese Tupel auch identifizieren können, müssen wir den Homomorphismus h in Zeile 18 mit den Abbildungen aus $h_\emptyset$ erweitern. In Zeile 19 müssen wir den Homomorphismus $h_\emptyset$ wieder auf die leere Menge von Abbildungen setzen. Da wir einen Widerspruch gefunden haben, müssen wir die Berechnung der Sicheren Instanz neu starten. Dafür entfernen wir in Zeile 19 auch alle bereits gefundenen Tupel aus der Ergebnisinstanz und starten in Zeile 20 erneut mit loop-1. Es würde hier nicht reichen, das widersprüchliche Tupel aus der Ergebnisinstanz zu löschen und dann mit loop-1 an der aktuellen Stelle fortzufahren, da bei einer möglicher Duplikateliminierung in der Ergebnisinstanz nicht mehr entschieden werden kann, ob Tupel nur aus einer widersprüchlichen Abbildung oder auch aus erlaubten Abbildungen entstanden sind.

[Zeile 21-25] Zur Zeile 21 gelangen wir nur, wenn wir die Schleife loop-3 erfolgreich durchlaufen haben. Hier prüfen wir, ob found immer noch false ist, was dann bedeutet, dass wir in der untersuchten Instanz kein passendes Tupel gefunden haben. Sollte das der Fall sein, brechen wir im darauf folgenden Schritt die weitere Suche nach dem Tupel aus loop-1 in allen Instanzen ab und starten in loop-1 die nächste Iteration mit dem nächsten Tupel. Zur Zeile 23 gelangen wir nur, wenn wir die Schleife loop-2 erfolgreich durchlaufen haben. Das ist nur der Fall, wenn wir in jeder Instanz ein passendes Tupel für den Durchschnitt unter dem Homomorphismus h gefunden haben. Mit der Anwendung von h auf das Tupel aus loop-1 erzeugen wir dann das Tupel, welches wir zu unserer Ergebnisinstanz hinzufügen können. Indem wir nach dem Durchlauf von loop-1 den Homomorphismus h noch einmal auf alle Tupel der Ergebnisinstanz anwenden, erzeugen wir in Zeile 24 das Endergebnis des Algorithmus. Dieser Schritt ist notwendig, um sicherzugehen, dass wirklich alle Nullwerte in den Tupeln mit allen Abbildungen in h abgebildet werden. So bilden wir auch die Nullwerte in den Tupeln ab, die vor der Aufnahme einer entsprechenden Abbildung des Nullwertes in h zu der Ergebnisinstanz hinzugefügt wurden. In der 25. Zeile können wir dann unsere Ergebnisinstanz des Durchschnitts unter einem Homomorphismus, sprich die Sichere Instanz aus den Eingabeinstanzen, zurückgeben.

Das in der Tabelle 4.1 angegebene Beispiel für den Algorithmus 5 deckt einige interessante Fälle ab. Als Eingabeinstanzen sei $\mathcal{I} = \{I_1, I_2\}$ gegeben, mit:

$$
\begin{aligned}
I_1 = \{&A(1, \eta_1, \eta_2),\\
&B(\eta_1),\\
&C(\eta_1),\\
&C(\eta_2)\},
\end{aligned}
\qquad\qquad
\begin{aligned}
I_2 = \{&A(1, 1, \eta_3),\\
&B(2),\\
&C(\eta_4),\\
&C(2)\}.
\end{aligned}
$$

Eine Besonderheit an diesem Beispiel ist die widersprüchliche Abbildung vom Nullwert η_1 aus I_1. Betrachtet man I_1 und I_2, könnte man zuerst erwarten, dass in ihrer Sicheren Instanz ein A-, ein B- und mindestens ein C-Tupel enthalten ist.

Bei der Suche nach Abbildungen zwischen den Tupeln der beiden Instanzen bilden wir zuerst die beiden A-Tupel aufeinander ab und erzeugen so eine Abbildung von η_1 auf die Konstante 1 aus dem A-Tupel der Instanz I_2. Das resultierende A-Tupel fügen wir dann zur Ergebnisinstanz hinzu. Wenn wir nun im weiteren Verlauf versuchen, die beiden B-Tupel aufeinander abzubilden, müssten wir η_1 auf die Konstante 2 aus dem B-Tupel der Instanz I_2 abbilden. Dabei stört uns aber die bestehende Abbildung von η_1 auf die Konstante 1. Aufgrund dieses Widerspruchs löschen wir das B-Tupel mit den in Konflikt stehenden Nullwert η_1 aus der Instanz I_1.

Tupel loop-1	Tupel loop-3	Abb. in h	Abb. in h_+	found	Abb. in $h_\emptyset$	Lösche Tupel	Tupel in $I_{\cap^h}$
$A(1, \eta_1, \eta_2)$	$A(1, 1, \eta_3)$	-	$\eta_1 \longrightarrow 1$ $\eta_2 \longrightarrow \eta_3$	true	-	-	-
$A(1, \eta_1, \eta_2)$	$B(2)$	$\eta_1 \longrightarrow 1$ $\eta_2 \longrightarrow \eta_3$	-	true	-	-	-
$A(1, \eta_1, \eta_2)$	$C(\eta_4)$	$\eta_1 \longrightarrow 1$ $\eta_2 \longrightarrow \eta_3$	-	true	-	-	-
$A(1, \eta_1, \eta_2)$	$C(2)$	$\eta_1 \longrightarrow 1$ $\eta_2 \longrightarrow \eta_3$	-	true	-	-	-
$B(\eta_1)$	$A(1, 1, \eta_3)$	$\eta_1 \longrightarrow 1$ $\eta_2 \longrightarrow \eta_3$	-	false	-	-	$A(1, 1, \eta_3)$
$B(\eta_1)$	$B(2)$	$\eta_1 \longrightarrow 1$ $\eta_2 \longrightarrow \eta_3$	-	false	$\eta_1 \longrightarrow 2$	$B(\eta_1) \in I_1$	-
$A(1, \eta_1, \eta_2)$	$A(1, 1, \eta_3)$	$\eta_1 \longrightarrow 1$ $\eta_2 \longrightarrow \eta_3$ $\eta_1 \longrightarrow 2$	-	false	$\eta_1 \longrightarrow 1$ $\eta_2 \longrightarrow \eta_3$	$A(1, \eta_1, \eta_2) \in I_1$	-
$C(\eta_1)$	$A(1, 1, \eta_3)$	$\eta_1 \longrightarrow 1$ $\eta_2 \longrightarrow \eta_3$ $\eta_1 \longrightarrow 2$	-	false	-	-	-
$C(\eta_1)$	$B(2)$	$\eta_1 \longrightarrow 1$ $\eta_2 \longrightarrow \eta_3$ $\eta_1 \longrightarrow 2$	-	false	-	-	-
$C(\eta_1)$	$C(\eta_4)$	$\eta_1 \longrightarrow 1$ $\eta_2 \longrightarrow \eta_3$ $\eta_1 \longrightarrow 2$	-	false	$\eta_1 \longrightarrow \eta_4$	$C(\eta_1) \in I_1$	-
$C(\eta_2)$	$A(1, 1, \eta_3)$	$\eta_1 \longrightarrow 1$ $\eta_2 \longrightarrow \eta_3$ $\eta_1 \longrightarrow 2$ $\eta_1 \longrightarrow \eta_4$	-	false	-	-	-
$C(\eta_2)$	$B(2)$	$\eta_1 \longrightarrow 1$ $\eta_2 \longrightarrow \eta_3$ $\eta_1 \longrightarrow 2$ $\eta_1 \longrightarrow \eta_4$	-	false	-	-	-
$C(\eta_2)$	$C(\eta_4)$	$\eta_1 \longrightarrow 1$ $\eta_2 \longrightarrow \eta_3$ $\eta_1 \longrightarrow 2$ $\eta_1 \longrightarrow \eta_4$	$\eta_1 \longrightarrow 1$ $\eta_2 \longrightarrow \eta_3$ $\eta_1 \longrightarrow 2$ $\eta_1 \longrightarrow \eta_4$ $\eta_3 \longrightarrow \eta_4$	true	-	-	-
$C(\eta_2)$	$C(2)$	$\eta_1 \longrightarrow 1$ $\eta_2 \longrightarrow \eta_3$ $\eta_1 \longrightarrow 2$ $\eta_1 \longrightarrow \eta_4$ $\eta_3 \longrightarrow \eta_4$	$\eta_1 \longrightarrow 1$ $\eta_2 \longrightarrow \eta_3$ $\eta_1 \longrightarrow 2$ $\eta_1 \longrightarrow \eta_4$ $\eta_3 \longrightarrow \eta_4$ $\eta_4 \longrightarrow 2$	true	-	-	-
-	-	$\eta_1 \longrightarrow 1$ $\eta_2 \longrightarrow \eta_3$ $\eta_1 \longrightarrow 2$ $\eta_1 \longrightarrow \eta_4$ $\eta_3 \longrightarrow \eta_4$ $\eta_4 \longrightarrow 2$	-	-	-	-	$\underline{C(2)}$

Tabelle 4.1.: Beispiel: Durchschnitt unter einem Homomorphismus

Nach der widersprüchlichen Abbildung von η_1 starten wir die Berechnung der Sicheren Instanz mit der aktualisierten Instanz, dem erweiterten Homomorphismus und der geleerten Ergebnisinstanz erneut und suchen wieder nach Abbildungen für die beiden A-Tupel. Da das A-Tupel aus I_1 mit η_1 einen Nullwert enthält, der im Homomorphismus auf verschiedene Konstanten abgebildet wird, löschen wir auch das A-Tupel aus I_1 und starten erneut mit der Berechnung der Sicheren Instanz.

Dass wir das A-Tupel entfernen müssen, ist erst mal logisch, da dieses ja auch die widersprüchliche Abbildung zum Homomorphismus hinzugefügt hat. Eine weitere Besonderheit des Beispiels finden wir im nächsten Tupel. $C(\eta_1)$ enthält nämlich auch η_1 und muss daher ebenfalls aus I_1 entfernt werden. Das liegt daran, dass wir η_1 bereits widersprüchlich abbilden und so keine Abbildung mehr für $C(\eta_1)$ erstellen können.

Im Beispiel bleibt uns in der Instanz I_1 danach nur noch das Tupel $C(\eta_2)$. Im nächsten Durchlauf der Berechnung der Sicheren Instanz vergleichen wir dieses zuerst mit dem Tupel $C(\eta_4)$ aus I_2 und übernehmen daher $C(\eta_4)$ in die Ergebnisinstanz. Die nächste Besonderheit ist, dass wir das Tupel im folgenden Schritt dann noch mit dem Tupel $C(2)$ aus I_2 gleichsetzen, wodurch wir den Homomorphismus um eine Abbildung von η_4 auf die Konstante 2 erweitern. Das führt letztendlich zu dem Tupel $C(2)$ in der Ergebnisinstanz, welches auch das einzige Tupel in der Sicheren Instanz von I_1 und I_2 ist.

Einschub - Optimierung Disjunktiver-Mengen Der im angepassten Disjunktiven-CHASE-Schritt aufgebaute CHASE-Baum wird je nach der Anzahl der S-T TGDs in einer Disjunktiven-Menge aufgespalten. Jede neue Abzweigung im CHASE-Baum erzeugt damit zusätzliche Instanzen im Ergebnis des angepassten Disjunktiven-CHASEs. Kombinieren wir nun die Anwendung des angepassten Disjunktiven-CHASEs mit der Bildung der Sicheren Instanz, fällt auf, dass es Kombinationen von Disjunktiven-Mengen gibt, deren im angepassten Disjunktiven-CHASE erzeugte Tupel nie in einer Sicheren Instanz auftauchen. Wissen wir, dass für das Ergebnis des angepassten Disjunktiven-CHASEs die Sichere Instanz berechnen werden soll, können wir die Disjunktiven-Mengen so noch vor der Anwendung des angepassten Disjunktiven-CHASEs optimieren.

Interessant sind für uns dabei die Disjunktiven-Mengen, für die der Durchschnitt der Relationen in den Kopf-Ausdrücken der einzelnen S-T TGDs leer ist. Es muss außerdem gelten, dass es keine andere Disjunktive-Menge gibt, bei der eine von den Relationen in den Kopf-Ausdrücken auch im eigenen Durchschnitt der Relationen in den Kopf-Ausdrücken der einzelnen S-T TGDs vorkommt. Tritt der beschriebene Fall ein, können wir die Disjunktive-Menge löschen und erhalten für das Ergebnis des angepassten Disjunktiven-CHASE mit der optimierten Menge von Disjunktiven-Mengen die gleiche Sichere Instanz wie mit den originalen Disjunktiven-Mengen. Da der Durchschnitt der Relationen in den Kopf-Ausdrücken der einzelnen S-T TGDs bei einelementigen Disjunktiven-Mengen nicht leer sein kann, können einelementige Disjunktiven-Mengen nie gelöscht werden. Das ist auch damit zu erklären, dass mindestens zwei S-T TGDs notwendig sind, um den CHASE-Baum aufzuspalten und somit mehrere Instanzen im Ergebnis des angepassten Disjunktiven-CHASEs zu erzeugen.

Seien ω_1 und ω_2 wie folgt innerhalb einer Menge von Disjunktiven-Mengen Ω gegeben:

$$\begin{aligned}\omega_1 = \{&Q(x_1, x_2, x_1, x_1) \longrightarrow A(x_1, x_2) \wedge B(x_1),\\ &Q(x_1, x_2, x_2, y) \longrightarrow \exists x_3 : C(x_1, x_2, x_2, x_3)\},\\ \omega_2 = \{&R(y, x_2) \longrightarrow \exists x_1, x_3 : C(x_1, x_2, x_2, x_3)\}.\end{aligned}$$

In diesem Beispiel müssen wir sowohl ω_1 als auch ω_2 in Ω behalten. Die Disjunktive-Menge ω_2 bleibt erhalten, da sie nur aus einer S-T TGD besteht und somit immer Tupel erzeugt, die in der Sicheren Instanz auftauchen werden. Das liegt daran, dass einzelne S-T TGDs in jeder Abzweigung des CHASE-Baumes die gleichen Tupel hinzufügen und diese dann bei der Durchschnittsberechnung in jeder Instanz aufeinander abgebildet werden können. Die Disjunktive-Menge ω_1 behalten wir ebenfalls, da sie Ausdrücke über Relationen im Kopf der S-T TGDs hat, über denen auch in ω_2 Kopf-Ausdrücke existieren. Solche Disjunktiven-Mengen spalten zwar den CHASE-Baum auf, erzeugen aber Tupel über Relationen in einem Teil des CHASE-Baumes, für die durch eine andere Disjunktive-Menge im gesamten CHASE-Baum Tupel erzeugt werden. Die Tupel, die durch die Disjunktive-Menge ω_1 im angepassten Disjunktiven-CHASE erzeugt werden, müssen somit in der Berechnung des Durchschnitts beachtet werden.

Seien ω_1 und ω_2 nun wie folgt innerhalb einer Menge von Disjunktiven-Mengen Ω gegeben:

$$\begin{aligned}\omega_1 = \{&Q(x_1, x_2, x_1, x_1) \longrightarrow A(x_1, x_2) \wedge B(x_1),\\ &Q(x_1, x_2, x_2, y) \longrightarrow \exists x_3 : C(x_1, x_2, x_2, x_3)\},\\ \omega_2 = \{&R(y, x_2) \longrightarrow \exists x_1, x_3 : C(x_1, x_2, x_2, x_3) \wedge D(x_1, x_2, x_2, x_3),\\ &R(y, x_2) \longrightarrow \exists x_1, x_3 : C(x_1, x_2, x_2, x_3) \wedge E(x_1, x_2, x_2, x_3)\}.\end{aligned}$$

Da der Durchschnitt der Relationen in den Kopf-Ausdrücken der einzelnen S-T TGDs in ω_2 nicht leer ist (in jedem Kopf ist ein Ausdruck über C enthalten), behalten wir auch in diesem Beispiel die Disjunktive-Menge ω_2. Die Disjunktive-Menge ω_1 müssen wir hier ebenfalls behalten, da sie mit $C(x_1, x_2, x_2, x_3)$ einen Ausdruck über einer Relation im Kopf der S-T TGDs hat, die auch im Durchschnitt der Relationen in den Kopf-Ausdrücken der einzelnen S-T TGDs von ω_2 enthalten ist.

Seien ω_1 und ω_2 nun wie folgt innerhalb einer Menge von Disjunktiven-Mengen Ω gegeben:

$$\begin{aligned}\omega_1 = \{&Q(x_1, x_2, x_1, x_1) \longrightarrow A(x_1, x_2) \wedge B(x_1),\\ &Q(x_1, x_2, x_2, y) \longrightarrow \exists x_3 : C(x_1, x_2, x_2, x_3)\},\\ \omega_2 = \{&R(y, x_2) \longrightarrow \exists x_1, x_3 : D(x_1, x_2, x_2, x_3)\}.\end{aligned}$$

In diesem Fall können wir ω_1 aus Ω entfernen. Mit ω_1 werden im angepassten Disjunktiven-CHASE unter anderem Instanzen geschaffen, die nur Tupel über A und B und keine Tupel über C enthalten und Instanzen, die nur Tupel über C und keine Tupel über A und B enthalten. Das Hinzufügen von Tupeln über D durch die S-T TGD in ω_2 kann nicht verhindern, dass zwischen den Tupeln über A und B und über den Tupeln über C eine leere Schnittmenge entsteht.

Betrachten wir Ω nur mit ω_1, können wir ω_1 aus denselben Gründen wie im letzten Beispiel entfernen. Die Ausführung des angepassten Disjunktiven-CHASEs können wir uns dann sparen, da die Sichere Instanz für ihr Ergebnis immer leer sein wird. Die folgende Definition fasst die beschriebene Optimierung formal zusammen.

Definition 4.8. (Optimierung der Disjunktiven-Mengen im GaLVE-Verfahren): Sei Ω eine Menge von Disjunktiven-Mengen, für deren Ergebnis mit dem angepassten Disjunktiven-CHASE die Sichere Instanz berechnet werden soll. Entstehe Ω' aus Ω, indem wir jede Disjunktive-Menge ω_i aus Ω entfernen, für die $\cap_{RK}^{\omega_i} = \emptyset$ und $\nexists \omega_j \in \Omega : R \in \cap_{RK}^{\omega_j} \wedge R \in \cup_{RK}^{\omega_i}$ gilt, wobei $\cap_{RK}^{\omega_{i|j}}$ der Durchschnitt der Relationen in den Kopf-Ausdrücken der einzelnen S-T TGDs von ω_i bzw. ω_j ist und $\cup_{RK}^{\omega_i}$ die Vereinigung der Relationen in den Kopf-Ausdrücken der einzelnen S-T TGDs von ω_i ist. Die Sichere Instanz des Ergebnisses des angepassten Disjunktiven-CHASEs mit Ω' entspricht der Sicheren Instanz, die auf dem Ergebnis des angepassten Disjunktiven-CHASEs mit Ω berechnet werden würde.

Im nächsten Abschnitt werden wir vorstellen, wie die inverse Schemaabbildung erweitert werden kann, um neue Tupel und Attribute so zur lokalen Datenbank hinzufügen, dass bestehende Anfragen ohne Transformationen gleichzeitig auf die lokalen und globalen Informationen zugreifen können. Wie wir ChaTEAU anpassen müssen, damit die Disjunktiven-Mengen unterstützt werden und wie der angepasste Disjunktive-CHASE sowie die Berechnung der Sicheren Instanz implementiert werden, wird im Kapitel 5 beschrieben.

4.3. Erweiterung der Quelle

Die Erweiterung der Quelle ist neben der Invertierung von Schemaabbildungen eine der Hauptaufgaben dieser Arbeit. Um die Herausforderungen besser beschreiben zu können, unterteilen wir die Aufgabe in zwei Unteranforderungen. **1.)** Zum einen benötigen wir eine automatisierbare Erweiterungsvorschrift für die Transformation einer inversen Schemaabbildung $\mathcal{M}' = (T, S, \Omega)$, wie sie durch den Algorithmus DirectAdaptedMaxExtendedRecovery($\mathcal{M}$) entstehen, in eine erweiterte inverse Schemaabbildung $\mathcal{M}'_+ = (T, S_+, \Omega_+)$ unter einem erweiterten Quellschema S_+. **2.)** Zum anderen müssen wir die mit $\mathcal{M}'_+$ zu S_+ hinzugefügten Daten so darstellen, dass unter S definierte Anfragen ohne Weiteres auf sie zugreifen können. Beides werden wir in den nächsten zwei Unterabschnitten 4.3.1 und 4.3.2 genauer betrachten. Der Unterabschnitt 4.3.3 gibt ein kleines Beispiel für die Erweiterung einer Schemaabbildung und der Nachbereitung deren Ausführung. Ein großes Beispiel, dass die Erweiterung innerhalb des kompletten GaLVE-Verfahrens zeigt, ist in Abschnitt 4.4 gegeben.

4.3.1. Erweiterung der Inversen

Ziel unserer gesuchten erweiterten inversen Schemaabbildung $\mathcal{M}'_+ = (T, S_+, \Omega_+)$ wird es sein, die Quelldatenbank unter S um Tupel und Attribute der globalen Datenbank unter T zu erweitern. Die Erweiterung um Tupel sehen wir als eine Art horizontale Erweiterung der original vorhandenen Tupel an. Abgeleitet von der horizontalen Fragmentierung im verteilten Datenbankentwurf, sollen die originalen Tupel neben den integrierten neuen Tupeln erhalten bleiben und nur auf Wunsch miteinander vereinigt werden. Um die originalen Tupel einer Relation R nach der Ausführung von $\mathcal{M}'_+$ zu bewahren, werden wir alle neuen Tupel für R in eine zusätzliche Relation R_+ speichern und R nicht direkt erweitern. Das hat den Vorteil, dass wir eventuell auftretende Inkonsistenzen zwischen den originalen Tupeln in R und den neuen Tupeln in R_+ in einer extra Standard-CHASE-Phase bereinigen können (siehe Abschnitt 4.3.2) und nicht die Disjunktiven-CHASE-Phase um Konzepte des Data Cleanings (EGDs) erweitern müssen. Mit dieser extra Standard-CHASE-Phase werden wir auch gewährleisten, dass Anfragen an R direkt auf die in R_+ integrierten Informationen zugreifen können.

Die Erweiterung um zusätzliche Attribute kann als vertikale Erweiterung beschrieben werden. Abgeleitet von der vertikalen Fragmentierung im verteilten Datenbankentwurf bleiben auch hier die originalen Attribute in einer Relation R bestehen und werden nur auf Wunsch mit den neuen Attributen einer zusätzlichen R_+-Relation verbunden. Die zusätzliche Relation R_+ müsste für die Darstellung der vertikalen Fragmentierung mindestens den Schlüssel der originalen Relation R speichern. Eine direkte Erweiterung der originalen Relation um die zusätzlichen Attribute wäre nur mit zusätzlichen Sichten und Umbenennungen möglich. Das liegt daran, dass Anfragen (als S-T TGDs) auf die korrekte Anzahl an Attributen für eine Relation (Variablen in den Ausdrücken) angewiesen sind und nicht einfach auf Relationen mit mehr Attributen zugreifen können.

Für unser Konzept der Erweiterung werden wir nur eine Relation $R_+ \in S_+$ pro Relation $R \in S$ benutzen und in ihr die neuen Tupel zusammen mit den neuen Attributen speichern. Eine abstrakte Darstellung des erweiterten Quellschemas S_+ zusammen mit der R_+-Relation für eine R-Relation ist in Abbildung 4.4 gegeben. Eine Aufteilung von R_+ in eine $R_{+\text{Tupel}}$-Relation für neue Tupel und eine $R_{+\text{Attribute}}$-Relation für zusätzliche Attribute ist nicht vorgesehen, da beide Relationen per TGDs aus R_+ heraus projiziert werden können. Die Nachbereitung von R_+ für die Anfragebeantwortung werden wir in Abschnitt 4.3.2 vorstellen.

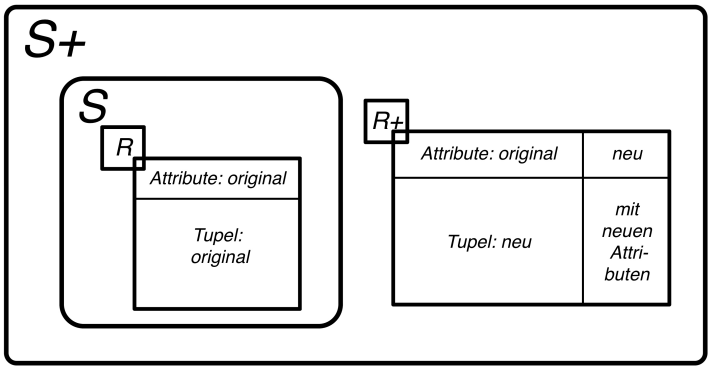

Abbildung 4.4.: Grober Aufbau erweitertes Quellschema S_+

Erweiterung um neue Tupel im Detail Um die Erweiterung einer Schemaabbildung[10] $\mathcal{M} = (T, S, \Omega)$ wie beschrieben umzusetzen, müssen wir als erstes die Disjunktiven-Mengen (siehe Definition 4.1) in Ω so abändern, dass ihre S-T TGDs keine Tupel mehr direkt zu den Relationen R des Quellschemas S, sondern zu neuen Relationen R_+ eines erweiterten Quellschemas S_+ hinzufügen. Dafür müssen wir jeden Relationennamen R der Ausdrücke über den Relationen R in den Köpfen der S-T TGDs durch einen neuen Relationennamen R_+ ersetzen, mit $R_+ \notin S$. Die Relationennamen R von Ausdrücken über der gleichen Relation R werden dabei auch durch den gleichen Relationennamen R_+ ersetzt. Das erweiterte Quellschema S_+ entsteht dann dadurch, dass wie für jede neu erzeugte R_+-Relation ihr entsprechendes Relationenschema zu S hinzufügen. Die Erweiterung um die neuen Tupel ist mit diesem Schritt bereits abgeschlossen.

Erweiterung um neue Attribute im Detail Für die Erweiterung der R_+-Relationen um neue Attribute aus dem globalen Schema T müssen wir noch die Attribute identifizieren, die für das lokale Schema S bzw. S_+ einen Mehrwert an Informationen liefern. Diese neuen Attribute entsprechen denen, die in einer mit $\mathcal{M}$ abgebildeten Relationen $R_T \in T$ enthalten sind, selbst aber nicht nach S bzw. S_+ abgebildet werden. Im Detail handelt es sich um die Attribute, die in einem Ausdruck über dem Rumpf-Relationenschema R_T einer Disjunktiven-Menge durch eine Variable repräsentiert werden, dann aber in keinem S-T TGD-Kopf-Ausdruck in derselben oder einer anderen Disjunktiven-Menge durch eine allquantifizierte Variable repräsentiert werden. Das Attribut darf demnach nicht im Rumpf und Kopf einer S-T TGD durch dieselbe Variable repräsentiert werden.

Mehr zur Erweiterung um neue Attribute Um die Abgrenzung von für das lokale Schema interessanten zu nicht interessanten Attributen besser zu verstehen, werden wir dies nun anhand von Beispielen erklären. So sind Attribute, die in einem anderen S-T TGD-Kopf derselben Disjunktiven-Menge durch eine allquantifizierte Variable repräsentiert werden, nicht interessant für uns. Sie werden innerhalb der Disjunktiven-Menge sowieso zur Quelle hinzugefügt und ihre Übernahme würde nur zu Redundanzen im Datenbankentwurf führen.

Definieren wir beispielsweise im Rostocker Uni-Datenbankschema S_R für unser allgemeines Beispiel (siehe Abschnitt 1.1) noch eine R-ABSCHLUSS-Relation, welche die Matrikelnummer eines Studierenden zusammen mit seinem angestrebten Abschluss speichert. Sei dazu passend eine Disjunktive-Menge ω_1 mit den S-T TGDs σ_1 und σ_2 (mit bereits angepassten Kopf) in einer inversen Schemaabbildung[11]

[10] Im Weiteren schreiben wir nur noch von Schemaabbildungen $\mathcal{M} = (T, S, \Omega)$, auch wenn wir die Erweiterung für inverse Schemaabbildungen $\mathcal{M}' = (T, S, \Omega)$ entwerfen, wie sie durch den Algorithmus DirectAdaptedMaxExtendedRecovery($\mathcal{M}$) entstehen. Dabei bezeichnen wie T weiterhin als globales Schema und S als Quellschema.

[11] Ergebnis vom Algorithmus DirectAdaptedMaxExtendedRecovery($\mathcal{M}$).

$\mathcal{M}' = (T, S_R, \Omega)$ enthalten. Die S-T TGDs σ_1 und σ_2 sind wie folgt gegeben:

$$\begin{aligned}\sigma_1 =& \text{T-STUDIERENDE}(x_{\text{Ma}}, x_{\text{Na}}, x_{\text{Vo}}, x_{\text{St}}, y_{\text{Ab}}) \longrightarrow \\ & \exists x_{\text{Be}} : \text{R-STUDIERENDE}_+(x_{\text{Ma}}, x_{\text{Na}}, x_{\text{Vo}}, x_{\text{St}}, x_{\text{Be}}), \\ \sigma_2 =& \text{T-STUDIERENDE}(x_{\text{Ma}}, x_{\text{Na}}, x_{\text{Vo}}, x_{\text{St}}, x_{\text{Ab}}) \longrightarrow \text{R-ABSCHLUSS}_+(x_{\text{Ma}}, x_{\text{Ab}}).\end{aligned}$$

Die S-T TGD σ_1 erweitern wir in diesem Fall nicht mit der Variable y_{Ab} für das Abschluss-Attribut, da dieses bereits mit σ_2 abgedeckt wird. Im Detail finden wir mit x_{Ab} eine Variable für den Abschluss, die im Rumpf und im Kopf von σ_2 auftaucht. Würden wir das Abschluss-Attribut zu R-STUDIERENDE$_+$ hinzufügen, heben wir die im Datenbankentwurf beabsichtigte Trennung des Abschlusses vom Studierenden auf und erzeugen Redundanzen in der Speicherung.

Ebenfalls nicht interessant sind die Attribute, die in einem S-T TGD-Kopf einer anderen Disjunktiven-Menge durch eine allquantifizierte Variable repräsentiert werden. Nehmen wir beispielsweise an, dass der Abschluss eines Studierenden sowohl im globalen als auch im lokalen Schema als zusätzliches Attribut in den Noten enthalten ist. Sei unsere Disjunktive-Menge ω_1 dieses Mal nur mit der S-T TGD σ_1 definiert und bestehe eine zweite Disjunktive-Menge ω_2 in $\mathcal{M}'$ nur aus der S-T TGD σ_3 (mit bereits angepasstem Kopf). Die S-T TGD σ_3 ist wie folgt gegeben:

$$\begin{aligned}\sigma_3 =& \text{T-NOTEN}(x_{\text{Mo}}, x_{\text{Ma}}, x_{\text{Se}}, x_{\text{No}}, x_{\text{Ab}}) \longrightarrow \\ & \exists x_{\text{Te}} : \text{R-TEILNAHME}_+(x_{\text{Te}}, x_{\text{Mo}}, x_{\text{Ma}}) \wedge \text{R-NOTEN}_+(x_{\text{Te}}, x_{\text{Se}}, x_{\text{No}}, x_{\text{Ab}}).\end{aligned}$$

Wir fügen auch in diesem Fall die Variable y_{Ab} für das Abschluss-Attribut nicht zu R-STUDIERENDE$_+$ hinzu. Die Begründung ist hierbei allerdings nicht so trivial wie im letzten Beispiel. Das Problem ist, dass wir nicht davon ausgehen können, dass in der globalen Datenbank T-NOTEN-Tupel existieren, die dann auch in die Quelldatenbank abgebildet werden. So kann es sein, dass kein Tupel in T-NOTEN existiert, sodass der Abschluss keines Studierenden nach R-NOTEN$_+$ abgebildet werden kann. In diesem Fall scheint eine Übernahme des Abschluss-Attributs zu R-STUDIERENDE$_+$ sinnvoll, würde im Allgemeinen aber auch zu Redundanzen führen. Das liegt daran, dass auch hier der Datenbankentwurf der Quelle vorsieht, dass der Abschluss in R-NOTEN gespeichert wird. Im Allgemeinen sollte also auch die Rückabbildung aus dem globalen Schema diese Entwurfsentscheidung berücksichtigen. Sollte der Fall eintreten, dass die lokalen und globalen Datenbankentwürfe sowie die Schemaabbildung von der lokalen in die globale Datenbank so vorliegen, dass Attribute wie beispielsweise der Abschluss für die Studierenden in der Rückabbildung nicht integriert werden können, so liegt der Fehler in der Datenbankintegration vor dem GaLVE-Verfahren und wird in der Erweiterung nicht aufgelöst werden. Wir werden den Datenbankentwurf also nicht verändern und lassen den Abschluss ausschließlich in R-NOTEN bzw. R-NOTEN$_+$.

Ein Beispiel, in dem der Abschluss ein interessantes Attribute darstellt, erhalten wir, wenn wir nur ω_1 mit der S-T TGD σ_1 in $\mathcal{M}'$ betrachten. In dem Fall würden wir R-STUDIERENDE$_+$ mit dem Abschluss-Attribut erweitern. Dafür fügen wir die Variable y_{Ab}, die den Abschluss im S-T TGD-Rumpf repräsentiert, zu dem R-STUDIERENDE$_+$-Ausdruck im S-T TGD-Kopf hinzu. Die so entstehende S-T TGD lautet:

$$\begin{aligned}\sigma_{1+} =& \text{T-STUDIERENDE}(x_{\text{Ma}}, x_{\text{Na}}, x_{\text{Vo}}, x_{\text{St}}, y_{\text{Ab}}) \longrightarrow \\ & \exists x_{\text{Be}} : \text{R-STUDIERENDE}_+(x_{\text{Ma}}, x_{\text{Na}}, x_{\text{Vo}}, x_{\text{St}}, x_{\text{Be}}, y_{\text{Ab}}).\end{aligned}$$

Entsteht eine Schemaabbildung $\mathcal{M}'$ als Ergebnis von DirectAdaptedMaxExtendedRecovery($\mathcal{M}$), sind für uns genau die Attribute von Interesse, die in jedem S-T TGD-Rumpf durch eine y-Variable repräsentiert werden. Das sind genau die Attribute im globalen Schema, die von $\mathcal{M}$ nicht mit Werten

gefüllt werden. Der Fall ist in unseren Beispielen nur mit der einzelnen S-T TGD σ_1 aufgetreten. In den Beispielen mit σ_2 und σ_3 gab es immer eine entsprechende x-Variable, die das Abschluss-Attribut repräsentiert hat.

Im Allgemeinen identifizieren wir für die Erweiterung um Attribute zuerst alle infrage kommenden Attribute einer Disjunktiven-Menge und erweitern dann jeden R_+-Ausdruck aller S-T TGD-Köpfe der Disjunktiven-Menge. Zu den R_+-Ausdrücken fügen wir die Variablen hinzu, die das neue Attribut in dem entsprechenden S-T TGD-Rumpf (zu dem S-T TGD-Kopf des R_+-Ausdrucks) repräsentieren. Wichtig dabei ist, dass wir beim Erweitern eines R_+-Ausdrucks in einer Disjunktiven-Menge auch alle weiteren Vorkommen von R_+ in anderen Disjunktiven-Mengen sowie die Definition des Relationenschemas von R_+ in S_+ anpassen müssen. Dieser Schritt ist notwendig, um die korrekte Anzahl an Variablen in den R_+-Ausdrücken gewährleisten zu können und reduziert zugleich den Aufwand, falls ein Attribut in mehreren Disjunktiven-Mengen zu den gleichen Relationen hinzugefügt werden würde. Falls das neue Attribut im Rumpf-Relationenschema der anderen Disjunktiven-Mengen enthalten ist, müssen wir darauf achten, die entsprechende Variable, die das Attribut im entsprechenden S-T TGD-Rumpf repräsentiert, zu verwenden. Falls das Rumpf-Relationenschema der anderen Disjunktiven-Mengen das neue Attribut nicht enthält, wählen wir eine neue Variable für die entsprechende S-T TGD.

Einen Sonderfall der Erweiterung stellen die Relationen im globalen Schema dar, von denen keine Abbildungen in das Quellschema ausgehen. Solche Relationen könnte man mit Hilfe einer neuen S-T TGD in einer zusätzlichen Disjunktiven-Menge in Ω_+ zu der Quelldatenbank als neue Relation hinzufügen. Dabei könnte man alle Attribute, von denen man durch bestehende S-T TGDs weiß, auf welche Attribute sie im lokalen Schema abgebildet werden, auch in der neuen S-T TGD richtig abbilden. Solche komplett neuen Relationen haben allerdings den Nachteil, dass sie die Quelle je nach Größe des globalen Schemas überproportional vergrößern könnten. Da die in ihnen vorhandenen Informationen nur bedingt einen Mehrwert für die Quelle liefern (vorherige Anwendungen haben die Relationen nicht vorgesehen), sollten solche Relationen im globalen Schema verbleiben und bei Bedarf direkt von dort angefragt werden. Im Konzept dieser Arbeit werden wir solche neuen Relationen daher nicht hinzufügen, auch wenn dies widersprüchlich zu der ursprünglichen Idee des GaLVE-Verfahrens ist, bei der die gesamte globale Datenbank als Erweiterung der lokalen Datenbank zur Verfügung stehen sollte (mehr dazu im Ausblick, siehe Kapitel 6).

Algorithmus Erweiterung Der Algorithmus Erweiterung($\mathcal{M}$) stellt eine automatisierbare Erweiterungsvorschrift für das beschriebene Vorgehen der Erweiterung dar. Als Eingabe erhält der Algorithmus eine Schemaabbildung $\mathcal{M} = (T, S, \Omega)$, wobei Ω eine Menge von Disjunktiven-Mengen ist. Als Eingabe kann also direkt die Schemaabbildung im Ergebnis vom Algorithmus 4 verwendet werden. Die Ausgabe vom Algorithmus ist dann die erweiterte Schemaabbildung $\mathcal{M}_+ = (T, S_+, \Omega_+)$, wobei S_+ ein aus S entstandenes erweitertes Quellschema und Ω_+ eine erweiterte Form von Ω ist. In der ersten Zeile von Erweiterung($\mathcal{M}$) wird das erweiterte Quellschema S_+ mit dem Quellschema S initialisiert. Die zweite Zeile initialisiert die zwei Mengen $\mathcal{A}$ und $\mathcal{D}$ für die Speicherung von neuen Attributen als leere Mengen. Dabei wird $\mathcal{A}$ als Menge aller interessanten Attribute und $\mathcal{D}$ als Menge der interessanten Attribute für eine Disjunktive-Menge dienen. Die beschriebene Änderung der Relationennamen in den Ausdrücken der S-T TGD-Köpfe von R zu R_+ wird in Zeile 3 behandelt, wobei die Menge von Disjunktiven-Mengen Ω_+ erzeugt wird. In der nächsten Zeile wir dann dafür gesorgt, dass S_+ um die Relationenschemata der neuen R_+-Relationen erweitert wird. Die Erweiterung um die neuen Attribute beginnt in Zeile 5 damit, dass wir über alle Attribute a der Rumpf-Relationenschema R_T der Disjunktiven-Mengen[12] in Ω_+ iterieren. Der darauf folgende Schritt in Zeile 6 prüft, ob keine S-T TGD in den Disjunktiven-Mengen in Ω_+ existiert, die eine Variable

[12] Es gibt nur ein Rumpf-Relationenschema R_T pro Disjunktiver-Menge (siehe Definition 4.1). Aus R_T wird mit $\mathcal{M}$ von T nach S abgebildet.

x_a für a sowohl im Rumpf als auch im Kopf hat. Sollte das der Fall sein, wird in Zeile 7 das Attribut a zur Menge aller interessanten Attribute $\mathcal{A}$ hinzugefügt. Nachdem wir in $\mathcal{A}$ alle interessanten Attribute gesammelt haben, können wir in Zeile 8 damit beginnen, über alle Disjunktiven-Mengen in Ω_+ zu iterieren, um sie mit den Attributen zu erweitern, die für sie interessant sind. Diese interessanten Attribute einer Disjunktiven-Menge werden in Zeile 9 dadurch gebildet, dass wir alle interessanten Attribute in $\mathcal{A}$ mit den Attributen des Rumpf-Relationenschemas der Disjunktiven-Menge schneiden und das Ergebnis in $\mathcal{D}$ speichern. Nun können wir jeden Ausdruck in den S-T TGD-Köpfen der Disjunktiven-Menge einzeln (siehe Iteration in Zeile 10) um Variablen für die neuen Attribute in $\mathcal{D}$ erweitern. Dabei stellt der Vektor $\mathbf{x}_a$ in Zeile 11 die Variablen dar, die im S-T TGD-Rumpf (zum Ausdruck aus dem S-T TGD-Kopf) die Attribute in $\mathcal{D}$ repräsentieren. Wichtig ist auch hier, dass wir nicht nur den Ausdruck über R_+ der gerade in der Iteration auftaucht, um $\mathbf{x}_a$ erweitern, sondern auch alle weiteren Ausdrücke über demselben R_+ sowie das Relationenschema von R_+ in S_+ anpassen. Sind wir mit diesem Schritt fertig, können wir unsere erweiterte Schemaabbildung als Ergebnis zurückgeben. Ein Beispiel für den Algorithmus werden wir im Abschnitt 4.3.3 zusammen mit der im nächsten Abschnitt vorgestellten Nachbereitung der erweiterten Schemaabbildung durchgehen.

Algorithmus 6 Erweiterung($\mathcal{M}$)

Input: Schemaabbildung $\mathcal{M} = (T, S, \Omega)$, mit Ω Menge von Disjunktiven-Mengen.
Output: Erweiterte Schemaabbildung $\mathcal{M}_+ = (T, S_+, \Omega_+)$.

1: Erweitertes lokales Schema $S_+ = S$
2: Mengen von neuen Attributen $\mathcal{A}, \mathcal{D} = \emptyset$
3: Menge von Disjunktiven-Mengen $\Omega_+ = f(\Omega)$, mit Funktion f ändert
S-T TGD-Kopf-Ausdrucksbezeichner R zu R_+, $R_+ \notin S$
4: S_+ += alle R_+
5: **for all** Attribute a in Rumpf-Relationenschema R_T der Disjunktiven-Mengen in Ω_+ **do**
6: **if** $\nexists$ S-T TGD $\in \Omega_+$ mit Variable x_a für a im Rumpf und Kopf **then**
7: $\mathcal{A}$ += a
8: **for all** $\omega \in \Omega_+$ **do**
9: $\mathcal{D} = \mathcal{A} \cup$ Attribute in ω-Rumpf-Relationenschema
10: **for all** S-T TGD-Kopf-Ausdruck $R_+(x) \in \omega$ **do**
11: $R_+(\mathbf{x}) = R_+(\mathbf{x}, \mathbf{x}_a)$, mit Vektor $\mathbf{x}_a$ aus Variablen für alle $a \in \mathcal{D}$ im S-T TGD-Rumpf
12: **Return** $\mathcal{M}_+ = (T, S_+, \Omega_+)$.

4.3.2. Nachbereitung Erweiterung

Im vorherigen Abschnitt haben wir beschrieben, wie eine inverse Schemaabbildung so erweitert werden kann, dass sie die originalen Relationen einer Quelldatenbank unverändert lässt und alle neuen Tupel zusammen mit neuen Attributen aus der globalen Datenbank in zusätzlichen Relationen speichert. Die Ausführung einer solchen erweiterten inversen Schemaabbildung ist natürlich noch nicht ausreichend, um mit Anfragen an die originalen Relationen (ohne weitere Anfragetransformationen) auf die neuen integrierten Daten zuzugreifen. Um die integrierten Daten nun zusammen mit den originalen Daten zur Verfügung zu stellen, gibt es grob zusammengefasst zwei Möglichkeiten:

Die erste Variante erweitert die originale Relation R. Soll R unverändert bleiben, können folgende Schritte auch virtuell vor der Anfrageausführung ausgeführt werden. Für die Nachbereitung der erweiterten inversen Schemaabbildung müssen wir als erstes die neuen Tupel aus R_+ (eingeschränkt auf die Attribute in R) mit einer TGD zu R hinzufügen. Um eventuell auftretende Inkonsistenzen zugunsten der originalen Daten auflösen zu können, müssen wir uns dabei merken, welche Tupel ursprünglich aus R stammen und welche aus R_+ neu hinzugefügt wurden. Der zweite Schritt besteht nun darin, die originalen und neuen Tupel in R mit einer EGD zu bereinigen. Dabei sehen wir (wie in Abschnitt 4.1 besprochen) die originalen Informationen immer als korrekt an und ersetzen in Konflikt stehende Informationen, wenn möglich mit

Werten, die ursprünglich aus der originalen Relation stammen. Neue Informationen können die originalen Tupel also höchstens erweitern, sprich Nullwerte durch Werte ersetzen oder als neue Tupel vorliegen. Nach der Bereinigung enthält R alle konsistenten neuen Informationen aus den neuen Tupeln in R_+. Der Aufbau von R und R_+ in dieser Variante ist in Abbildung 4.5 angegeben. Anfragen an R können in dieser Variante direkt auf die neuen Tupel zugreifen. Da Anfragen sowieso angepasst werden müssen, wenn sie die neuen Attribute nutzen wollen, stellt es keinen Widerspruch dar, dass die neuen Attribute in R_+ verbleiben. Für Anfragen an die bereinigten Tupel zusammen mit den neuen Attributen können wir eine Sicht definieren, die R über den Schlüssel von R mit den neuen Attributen in R_+ verbindet. Auch direkte Anfragen an die neuen Attribute in R_+ sind über den Schlüssel von R möglich, zumindest wenn man nicht auch auf die restlichen originalen Attribute[13] zugreifen möchte.

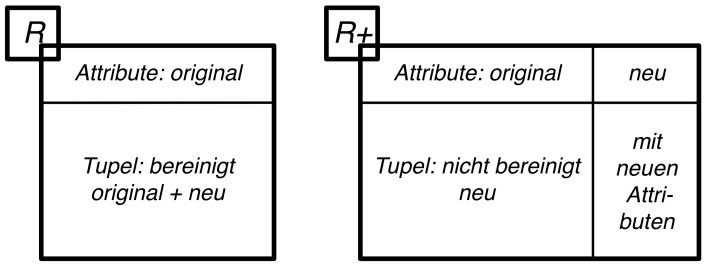

Abbildung 4.5.: Grober Aufbau Nachbereitung mit der ersten Variante

Die zweite Variante ist die Erweiterung von R_+ um die originalen Tupel aus R. Auch hier müssen wir uns beim Hinzufügen der originalen Tupel zu R_+ merken, welche Tupel ursprünglich aus R stammen, um die originalen und neuen Tupel wie in der ersten Variante zu bereinigen. Dabei müssen wir neue Tupel, die in R_+ das Hinzufügen von Tupel aus R verhindern (da sie gleich- oder höherwertige Informationen[14] beinhalten) auch als originale Tupel ansehen. Der Aufbau von R und R_+ in dieser Variante ist in Abbildung 4.5 angegeben. Um die integrierten Informationen nun den Anfragen an R ohne weitere Transformationen zur Verfügung zu stellen, müssen wir eine Sicht auf R_+ definieren, die R_+ auf die originalen Attribute von R projiziert (die neuen Attribute stören Anfragen als S-T TGDs). Dabei muss beachtet werden, dass die Sicht genau so wie R zugreifbar sein soll, was bedeutet, dass das originale R umbenannt werden muss. Da so eine Sicht selbst alle originalen Tupel enthält, kann die originale Relation R als Alternative auch gelöscht werden. Das Löschen sowie das Umbenennen kann allerdings zu Problemen bei bestehenden Beziehungen (z. B. Fremdschlüsselbeziehungen) unter den Relationen führen. Der Vorteil der zweiten Variante ist, dass alle integrierten Informationen zusammen mit den originalen Tupeln in nur einer Relation vorliegen. Durch das Löschen von R, in Verbindung mit einer Sicht auf R_+ können wir auch Speicherplatz und Redundanzen einsparen.

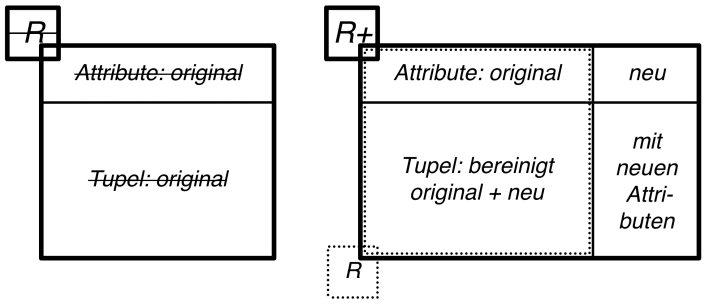

Abbildung 4.6.: Grober Aufbau Nachbereitung mit der zweiten Variante

[13] Da die originalen Attribute in R_+ nicht bereinigt wurden, darf auf sie nicht direkt zugegriffen werden.
[14] Gemeint sind neue Tupel, die in den originalen Attributen mit den originalen Tupeln übereinstimmen oder Werte haben, wo in den originalen Tupeln Nullwerte gespeichert sind.

Die beiden vorgestellten Varianten eignen sich für verschiedene Anwendungszwecke. Die erste Variante ist besser geeignet, wenn es darum geht Legacy-Anwendungen zu unterstützen. Die direkte Erweiterung der originalen Relationen erzeugt keinen Mehraufwand bei der Ausführung von bestehenden Anfragen und erfordert keine Umstellung auf neue Strukturen für spätere Anwendungen. Nachteilig bei der ersten Variante ist die externe Speicherung von neuen Attributen in einer zusätzlichen Relation, was bei Anfragen an diese einen extra Verbund erfordert. Zudem bleiben die nicht bereinigten neuen Werte der originalen Attribute in der zusätzlichen Relation bestehen und dürfen nicht für Anfragen genutzt werden. Die zweite Variante kann mit Hilfe von materialisierten Sichten bei der Unterstützung von Legacy-Anwendungen mithalten, benötigt allerdings Mehraufwand für deren Bereitstellung und Pflege. Auch die Umstellung auf die neue Struktur von R_+ für kommende Anwendungen bedeutet Mehraufwand gegenüber der ersten Variante.

In weiteren Verlauf dieser Arbeit werden wir die erste Variante für die Erweiterung der Quellen nutzen. Eine Umsetzung der zweiten Variante sollte allerdings ähnlich ablaufen. Entscheidend für die Auswahl der ersten Variante ist die direkte Unterstützung von Legacy-Anwendungen, was einer der Hauptzielsetzungen der GaLVE-Technik entspricht. Der Nachteil der redundanten Speicherung von nicht bereinigten neuen Werten in R_+ kann zudem dadurch gelöst werden, dass man nach der Nachbereitung die nicht zum Schlüssel von R gehörenden originalen Attribute aus dem Relationenschema von R_+ entfernt. Der zusätzlich notwendige Verbund könnte zumindest für den Anwender mit einer Sicht verschleiert werden.

Theorie der Nachbereitung Bevor wir nun die Erstellung der TGDs und EGDs für die beschriebene Nachbereitung mit der ersten Variante vorstellen, müssen wir zunächst klären, wie wir uns beim Hinzufügen eines Tupels mittels TGD merken können, dass dieses nicht original ist und wie wir diese Information nutzen können, um in einer EGD die Werte eines originalen Tupels bevorzugt zu behandeln. Fügen wir ein neues Tupel aus R_+ mittels TGD zu R hinzu, haben wir die Möglichkeit, ein zusätzlich definiertes Attribut in R mit einem konstanten Wert für neue Tupel aus R_+ zu füllen. Dieses zusätzliche Attribut muss in R initial mit einem anderen konstanten Wert für die originalen Tupel besetzt werden. Nachdem die Attributwerte für die Bereinigung mittels angepasster EGD genutzt wurden, kann das zusätzliche Attribut wieder aus R entfernt werden. In dieser Arbeit werden wir der Einfachheit halber Provenance-Annotationen an den Tupeln nutzen, um zwischen originalen und neuen Tupeln zu unterscheiden. Für ein originales Tupel in der Relation R schreiben wir $R(...)|_{original}$, für ein neues Tupel schreiben wir $R(...)|_{neu}$. Die Provenance-Annotationen können bei der Ausführung einer TGD automatisch durch einen angepassten Standard-CHASE hinzugefügt werden. Diesen angepassten Standard-CHASE werden wir hier allerdings nicht definieren, da die Variante mit dem zusätzlichen Attribut mit dem Standard-CHASE nach Definition 2.18 umgesetzt werden könnte.

Wichtiger für diese Arbeit sind die angepassten EGDs, die nun die Provenance-Annotationen nutzen, um Werte aus originalen Tupeln bevorzugt zu behandeln. So eine *biased EGD*[15] (kurz: bEGD) entspricht exakt der Definition von herkömmlichen EGDs (siehe Definition 2.8), erwartet aber zusätzliche Provenance-Annotationen an den Tupeln.

Definition 4.9. (bEGD): *Biased equality-generating dependencies* sind prädikatenlogische Ausdrücke erster Ordnung der Form

$$\forall \mathbf{x} : (\phi(\mathbf{x}) \longrightarrow (x_1 = x_2), ..., (x_n = x_m)),$$

wobei $\phi(\mathbf{x})$ eine Konjunktion von atomaren Ausdrücken der Form $R_i(a_1, ..., a_p)$ ist und jedes a_j (mit $j \in \{1, ..., p\}$) eine allquantifizierte Variable aus $\mathbf{x}$ ist. Der Bezeichner R_i steht für den Bezeichner eines Relationenschemas. Alle Variablen in den Gleichheitsatomen auf der rechten Seite kommen auch in

[15] Eingedeutscht auch „*befangene EGD*“ genannt.

$\phi(\mathbf{x})$ vor. Eine bEGD erwartet Provenance-Annotationen für originale und neue Tupel an den Tupeln der Relationen.

Der Unterschied zu den herkömmlichen EGDs wird erst in der Verarbeitung durch einen *befangenen Standard-CHASE-Schritt*[16] ersichtlich. In ihm können wir die Provenance-Annotationen nutzen, um bei zwei verglichenen Tupeln auch Konstanten eines neuen Tupels mit Konstanten eines originalen Tupels zu ersetzen. Zusätzlich können wir mit ihnen Inkonsistenzen in den neuen Tupeln entfernen, ohne den CHASE scheitern zu lassen. Dafür müssen wir nur die in Konflikt stehenden Konstanten in den verglichenen neuen Tupeln mit einem neuen Nullwert ersetzen. Da die in Konflikt stehenden Konstanten aus der globalen Datenbank stammen, gehen durch ihre Ersetzung mit einem Nullwert keine wichtigen Informationen für das Quellschema verloren. Auch hier wäre die Umsetzung mit einem zusätzlichen Attribut anstatt der Provenance-Annotation möglich, welche wir in dieser Arbeit aber nicht weiter ausführen werden. Wir beschränken uns auf die folgende Definition mit Provenance-Annotationen.

Definition 4.10. (Befangener CHASE-Schritt auf Instanzen für bEGDs): Sei $g : \phi(\mathbf{x}) \longrightarrow I_i$ ein aktiver Trigger für eine bEGD $b : \phi(\mathbf{x}) \longrightarrow (x_1 = x_2), ..., (x_q = x_p)$ und einer Instanz I_i. Für den *befangenen CHASE-Schritt* $I_i \xrightarrow{b,g} I_{i+1}$ gilt:

- Falls die Abbildungen $g(x_n)$ und $g(x_m)$ für ein Gleichheitsatom $(x_n = x_m)$ unterschiedliche Konstanten ergeben, wobei die Konstante $g(x_n)$ aus einem Tupel mit der Provenance-Annotation für ein originales Tupel und die Konstante $g(x_m)$ aus einem Tupel mit der Provenance-Annotation für ein neues Tupel stammt (z. B. $g(x_n) = 1|_{\text{original}}$, $g(x_m) = 2|_{\text{neu}}$), dann wird die Konstante $g(x_m)$ durch die Konstante $g(x_n)$ ersetzt ($h(2) = 1$ mit $h : I_i \longrightarrow I_{i+1}$). Wichtig ist, dass der Homomorphismus h nur die einzelne Konstante für das entsprechende Attribut in dem verglichenen neuen Tupel ersetzt und nicht jede Konstante in der Instanz, die zufälliger Weise den gleichen Wert hat.

- Falls die Abbildungen $g(x_n)$ und $g(x_m)$ für ein Gleichheitsatom $(x_n = x_m)$ unterschiedliche Konstanten ergeben, wobei die Konstanten $g(x_n)$ und $g(x_m)$ aus Tupeln mit der Provenance-Annotation für ein neues Tupel stammen (z. B. $g(x_n) = 1|_{\text{neu}}$, $g(x_m) = 2|_{\text{neu}}$), dann werden beide Konstanten durch einen neuen Nullwert ersetzt ($h(2) = \eta_j, h(1) = \eta_j$ mit $h : I_i \longrightarrow I_{i+1}$). Wichtig ist, dass der Homomorphismus h nur die beiden Konstanten für die entsprechenden Attribute in den verglichenen neuen Tupeln ersetzt und nicht jede Konstante in der Instanz, die zufälliger Weise den gleichen Wert hat.

- Falls die Abbildungen $g(x_n)$ und $g(x_m)$ für ein Gleichheitsatom $(x_n = x_m)$ unterschiedliche Konstanten ergeben, wobei die beiden Konstanten $g(x_n)$ und $g(x_m)$ aus Tupeln mit der Provenance-Annotation für ein originales Tupel stammen (z. B. $g(x_n) = 1|_{\text{original}}$, $g(x_m) = 2|_{\text{original}}$), dann schlägt der CHASE fehl ($I_{i+1} = \perp$).

- Falls eine Variable des Gleichheitsatoms $(x_m = x_m)$ auf eine Konstante abgebildet wird und die andere auf einen benannten Nullwert (z. B. $g(x_m) = 1|_{[\text{original}|\text{neu}]}$, $g(x_n) = \eta_{\text{Ma}}|_{[\text{original}|\text{neu}]}$), dann wird der Nullwert im Allgemeinen ungeachtet der Provenance-Annotation überall durch die Konstante ersetzt ($h(\eta_{\text{Ma}}) = 1$ mit $h : I_i \longrightarrow I_{i+1}$). Hierbei müssen allerdings einige Sonderfälle (*) beachtet werden.

- Falls beide Variablen des Gleichheitsatoms $(x_m = x_n)$ auf unterschiedlich benannte Nullwerte abgebildet werden (z. B. $g(x_m) = \eta_{\text{No}_1}|_{[\text{original}|\text{neu}]}$, $g(x_n) = \eta_{\text{No}_2}|_{[\text{original}|\text{neu}]}$), dann wird ein Nullwert ungeachtet der Provenance-Annotation überall durch den anderen ersetzt. Die Auswahl erfolgt über den durch die Namenskonvention vorgeschriebenen Index der Nullwerte. Die Konvention besagt, dass verschiedene Nullwerte für ein Attribut durchnummeriert werden. So kann man den Nullwert

[16] Angelehnt an die biased EGD auf Englisch auch „*biased CHASE step*“ genannt.

mit dem größeren Index durch den Nullwert mit dem kleineren Index ersetzen ($h(\eta_{\text{No}_2}) = \eta_{\text{No}_1}$ mit $h : I_i \longrightarrow I_{i+1}$).

Diese Regeln werden für jedes Gleichheitsatom aus b auf I_i angewendet. Bei der Anwendung wird ein Homomorphismus h zwischen den Instanzen I_i und I_{i+1} gebildet, der die Werte, die ersetzt werden (z. B. $g(x_n) = \eta_{\text{Ma}}|_{[\text{original|neu}]}$), auf die ersetzenden Werte (z. B. $g(x_m) = 1|_{[\text{original|neu}]}$) abbildet ($(h(g(x_n)) = g(x_m)) \longrightarrow h(\eta_{\text{Ma}}) = 1$ mit $h : I_i \longrightarrow I_{i+1}$). Falls zwei Tupel nach der Gleichsetzung in jedem Attribut außer der Provenance-Annotation übereinstimmen, ersetzt das originale Tupel das neue.

Die Anwendung einer bEGD mit dem befangenen CHASE-Schritt überführt die Instanz I_i in eine modifizierte Instanz I_{i+1}. Schlägt ein befangener CHASE-Schritt fehl, wird anstatt einer Instanz das Symbol für Falschheit $\perp$ zurückgegeben.

(*) Die Sonderfälle treten auf, wenn wir im selben CHASE-Schritt einen Nullwert mit zwei verschiedenen Konstanten vergleichen. Im normalen CHASE-Schritt würden dann auch die beiden verschiedenen Konstanten verglichen werden und $\perp$ zurückgegeben werden, wodurch der CHASE in so einer Konstellation abbrechen würde. Im befangenen CHASE-Schritt werden zwar auch die Konstanten verglichen, es kann aber passieren, dass sich dann die Konstante aus einem originalen Tupel gegenüber der aus einem neuen Tupel durchsetzt oder beide Konstanten aus neuen Tupeln stammen und durch einen neuen Nullwert ersetzt werden. Der CHASE würde dann nicht abbrechen. In diesen Fällen müssen wir den Homomorphismus h, der die Instanz I_i am Ende eines CHASE-Schrittes in eine modifizierte Instanz I_{i+1} überführt, anders als gewohnt anpassen.

- Besitzt h bereits eine Abbildung von einem Nullwert η_i auf eine Konstante c_{original_i} (aus einem originalen Tupel), fügen wir keine weitere Abbildung zu h hinzu, die auch η_i abbilden will.
  - Soll eine Abbildung von η_i auf eine andere Konstante c_{original_j} (aus einem originalen Tupel) hinzugefügt werden, bricht der CHASE nach diesem CHASE-Schritt sowieso ab, da auch die beiden Konstanten c_{original_i} und c_{original_j} noch verglichen werden.
  - Soll eine Abbildung von η_i auf eine andere Konstante c_{neu_i} (aus einem neuen Tupel) hinzugefügt werden, wird c_{neu_i} im gleichen CHASE-Schritt sowieso noch auf die Konstante c_{original_i} abgebildet.
- Besitzt h bereits eine Abbildung von einem Nullwert η_i auf eine Konstante c_{neu_i} (aus einem neuen Tupel), müssen wir folgende Fälle unterscheiden:
  - Soll eine Abbildung von η_i auf eine andere Konstante c_{original_i} (aus einem originalen Tupel) hinzugefügt werden, ersetzt diese die vorherige Abbildung von η_i auf c_{neu_i}, da c_{neu_i} im gleichen CHASE-Schritt sowieso noch auf die Konstante c_{original_i} abgebildet wird.
  - Soll eine Abbildung von η_i auf eine andere Konstante c_{neu_j} (aus einem neuen Tupel) hinzugefügt werden, löschen wir die vorherige Abbildung von η_i auf c_{neu_i} ersatzlos und fügen die neue Abbildung nicht hinzu, da c_{neu_i} im gleichen CHASE-Schritt sowieso noch mit der Konstante c_{neu_j} verglichen wird und dann beide auf einen neuen Nullwert η_j abgebildet werden. Dieser kann dann im nächsten CHASE-Schritt mit η_i verglichen werden.

Im Weiteren bezeichnen wir den mit dem befangenen Standard-CHASE-Schritt erweiterten *befangenen Standard-CHASE* nur noch als Standard-CHASE. Sobald eine bEGD anstatt einer TGD oder EGD im Standard-CHASE verarbeitet wird, ist klar, dass wir den eben definierten befangenen Standard-CHASE-Schritt benutzen müssen.

Algorithmus Nachbereitung Jetzt können wir klären, wie die TGDs und bEGDs für die Nachbereitung im Detail aufgebaut werden müssen. Die TGD wird dafür genutzt, die neuen Tupel aus R_+ in die originale Relation R zu kopieren. Die bEGD ist dann dafür zuständig, die originalen und neuen Tupel in R zu bereinigen. Wir benötigen jeweils eine TGD und eine bEGD für jede neue R_+-Relation. Der Aufbau der TGD und bEGD für ein R_+ ist immer derselbe. Die TGD hat für ein R_+ und das dazugehörige R (für das R_+ die neuen Tupel und Attribute speichert) die Form

$$R_+(\mathbf{x}, \mathbf{x}_a) \longrightarrow R(\mathbf{x}),$$

wobei $\mathbf{x}$ ein Vektor aus Variablen ist, die die originalen Attribute in R repräsentieren und $\mathbf{x}_a$ ein Vektor ist, der die Variablen für alle neuen Attribute enthält, die nur in R_+ vorkommen. Die bEGD wir nun auf dem R-Ausdruck aus dem Kopf der zugehörigen TGD definiert und hat die Form

$$R(\mathbf{k}, \mathbf{x}_1) \wedge R(\mathbf{k}, \mathbf{x}_2) \longrightarrow \mathbf{x}_1 = \mathbf{x}_2,$$

wobei $\mathbf{k}$ ein Vektor aus Variablen ist, die die Schlüsselattribute von R repräsentieren und $\mathbf{x}_1, \mathbf{x}_2$ Vektoren sind die Variablen für alle Nichtschlüsselattribute enthalten.

Für die Nachbereitung werden alle so erzeugten TGDs und bEGDs in einer Menge von Integritätsbedingungen $\mathcal{N}$ gesammelt. Diese können wir nun mit dem Standard-CHASE (der während der TGD-Ausführung die Tupel mit Provenance-Annotationen versieht) in die Ergebnisinstanz der erweiterten Schemaabbildung einchasen. In der Ergebnisinstanz des CHASE für die Nachbereitung enthält jede originale Relation R die bereinigten originalen und neuen Tupel aus R und R_+. Legacy-Anwendungen können somit weiterhin auf das bekannte R zugreifen und haben automatisch Zugriff auf die integrierten Informationen aus der globalen Datenbank. Der Algorithmus 7 fast die Erstellung der TGDs und bEGD für die Nachbereitung einer erweiterten Schemaabbildung noch einmal zusammen.

Algorithmus 7 Nachbereitung(S, S_+)

Input: Schemata S und S_+, wobei S_+ ein aus S entstandenes erweitertes Schema ist.
Output: Menge $\mathcal{N}$ von TGDs und bEGDs für die Nachbereitung einer erweiterten Schemaabbildung $\mathcal{M}_+ = (T, S_+, \Omega_+)$.

1: $\mathcal{N} = \emptyset$
2: **for all** $R_+ \in S_+ \wedge R_+ \notin S$ **do**
3: $\mathcal{N}$ += TGD und bEGD für die Nachbereitung von $R_+(\mathbf{x}, \mathbf{x}_a)$ mit Vektor $\mathbf{x}_a$ aus Variablen für neue Attribute a:

- TGD: $R_+(\mathbf{x}, \mathbf{x}_a) \longrightarrow R(\mathbf{x})$
- bEGD: $R(\mathbf{k}, \mathbf{x}_1) \wedge R(\mathbf{k}, \mathbf{x}_2) \longrightarrow \mathbf{x}_1 = \mathbf{x}_2$, mit $\mathbf{k}$ Vektor aus Variablen für Schlüsselattribute von $R \in S$

4: **Return** $\mathcal{N}$

Beispiel Theorie der Nachbereitung Betrachten wir als Beispiel für die Theorie der Nachbereitung eine Instanz I unter dem bekannten Relationenschema R-STUDIERENDE $\in S$ und einem Relationenschema R-STUDIERENDE$_+ \in S_+$, in dem noch ein Abschluss-Attribut hinzugefügt wurde. Die Relation[17] R-STUDIERENDE$_+$ entspreche der Relation für die Speicherung der neuen Tupel und Attribute für R-STUDIERENDE. Die mit Algorithmus Nachbereitung(S, S_+) erzeugte Menge von Integritätsbedingungen $\mathcal{N}$ für die Nachbereitung bestehe aus der TGD σ_1 und der bEGD σ_2.

[17] Durch eine mit dem Algorithmus Erweiterung$(\mathcal{M})$ erzeugte erweiterte Schemaabbildung erstellt.

Die Instanz I, die TGD σ_1 und die EGD σ_2 seien wie folgt gegeben:

$$
\begin{aligned}
I = \{&\text{R-STUDIERENDE}(2, \text{Sonne}, \eta_{\text{Vo}_1}, \text{ITTI}, \eta_{\text{Be}_1}),\\
&\text{R-STUDIERENDE}(2, \text{Sonne}, \eta_{\text{Vo}_1}, \eta_{\text{St}_1}, \eta_{\text{Be}_1}),\\
&\text{R-STUDIERENDE}_+(2, \text{Sonne}, \text{Sarah}, \text{Informatik}, \eta_{\text{Be}_2}, \text{Bachelor})\},\\
\sigma_1 = &\text{R-STUDIERENDE}_+(x_{\text{Ma}}, x_{\text{Na}}, x_{\text{Vo}}, x_{\text{St}}, x_{\text{Be}}, y_{\text{Ab}}) \longrightarrow\\
&\text{R-STUDIERENDE}(x_{\text{Ma}}, x_{\text{Na}}, x_{\text{Vo}}, x_{\text{St}}, x_{\text{Be}}),\\
\sigma_2 = &\text{R-STUDIERENDE}(x_{\text{Ma}}, x_{\text{Na}_1}, x_{\text{Vo}_1}, x_{\text{St}_1}, x_{\text{Be}_1}) \wedge\\
&\text{R-STUDIERENDE}(x_{\text{Ma}}, x_{\text{Na}_2}, x_{\text{Vo}_2}, x_{\text{St}_2}, x_{\text{Be}_2}) \longrightarrow\\
&(x_{\text{Na}_1} = x_{\text{Na}_2}), (x_{\text{Vo}_1} = x_{\text{Vo}_2}), (x_{\text{St}_1} = x_{\text{St}_2}), (x_{\text{Be}_1} = x_{\text{Be}_2}).
\end{aligned}
$$

Wir können nun mit der TGD σ_1 das Tupel in R-STUDIERENDE$_+$ zu R-STUDIERENDE hinzufügen. Dabei müssen wir die originalen und die neuen Tupel in R-STUDIERENDE mit Provenance-Annotationen versehen. Nach der Anwendung von σ_1 erhalten wir die Instanz I':

$$
\begin{aligned}
I' = \{&\text{R-STUDIERENDE}(2, \text{Sonne}, \eta_{\text{Vo}_1}, \text{ITTI}, \eta_{\text{Be}_1})|_{\text{original}},\\
&\text{R-STUDIERENDE}(2, \text{Sonne}, \eta_{\text{Vo}_1}, \eta_{\text{St}_1}, \eta_{\text{Be}_1})|_{\text{original}},\\
&\text{R-STUDIERENDE}(2, \text{Sonne}, \text{Sarah}, \text{Informatik}, \eta_{\text{Be}_2})|_{\text{neu}},\\
&\text{R-STUDIERENDE}_+(2, \text{Sonne}, \text{Sarah}, \text{Informatik}, \eta_{\text{Be}_2}, \text{Bachelor})\}.
\end{aligned}
$$

Mit Hilfe der bEGD σ_2 können wir nun die Tupel in R-STUDIERENDE über die Matrikelnummer gleichsetzen, obwohl sie widersprüchliche Studiengänge beinhalten. Da der Studiengang ITTI laut der Provenance-Annotation in einem originalen Tupel gespeichert ist und der Studiengang Informatik aus einem neuen Tupel stammt, können wir Informatik in dem neuen Tupel durch ITTI ersetzen. Den Nullwert η_{St_1} in dem einen originalen Tupel ersetzen wir ebenfalls durch den Studiengang ITTI aus dem anderen originalen Tupel. Den Nullwert η_{Vo_1} ersetzen wir in beiden originalen Tupeln wie bei normalen EGDs durch den Vornamen Sarah aus dem neuen Tupel. Auch die Gleichsetzung der Nullwerte für die Bemerkung erfolgt, wie wir es von den EGDs kennen, indem sich der Nullwert mit dem kleineren Index durchsetzt. Letztendlich haben die drei Tupel in R-STUDIERENDE für jedes Attribut den gleichen Wert und können auf ein originales Tupel reduziert werden. Die Instanz I'' enthält nun die bereinigten Tupel nach der Ausführung von σ_2:

$$
\begin{aligned}
I'' = \{&\text{R-STUDIERENDE}(2, \text{Sonne}, \text{Sarah}, \text{ITTI}, \eta_{\text{Be}_1})|_{\text{original}},\\
&\text{R-STUDIERENDE}_+(2, \text{Sonne}, \text{Sarah}, \text{Informatik}, \eta_{\text{Be}_2}, \text{Bachelor})\}.
\end{aligned}
$$

Der beschriebene Sonderfall in der Bearbeitung einer bEGD trat auch in diesem Beispiel ein, fiel aber durch die grobe Beschreibung der Abbildungen nicht weiter auf. Beschreibt man die Erstellung des Homomorphismus h zwischen den einzelnen CHASE-Schritten detaillierter, müssen wir beim Hinzufügen mancher Abbildungen auch die bereits hinzugefügten Abbildungen in h beachten. In unserem Beispiel müssten wir z. B. eine bestehende Abbildung des Nullwertes η_{St_1} auf den Studiengang Informatik in h durch die Abbildung vom Nullwert η_{St_1} auf den Studiengang ITTI ersetzen. Falls andersherum z. B. bereits eine Abbildung vom Nullwert η_{St_1} auf den Studiengang ITTI in h enthalten wäre, würden wir die Abbildung vom Nullwert η_{St_1} auf den Studiengang Informatik nicht mehr zu h hinzufügen.

4.3.3. Beispiel Erweiterung mit Nachbereitung

Mit den Algorithmen Erweiterung($\mathcal{M}$) und Nachbereitung(S, S_+) haben wir alles, um ein Beispiel für die Erweiterung von inversen Schemaabbildungen in unserem GaLVE-Verfahren anzugeben. Betrachten wir dafür die Schemaabbildungen $\mathcal{M} = (S_R, T, \Sigma)$ vom Rostocker Datenbankschema S_R in das globale Uni-Datenbankschema T (siehe allgemeines Beispiel im Abschnitt 1.1) und ihre mit Algorithmus 4 berechnete Inverse $\mathcal{M}' = (T, S_R, \Omega)$. Seien

$$\begin{aligned}\Sigma = \{&\text{R-STUDIERENDE}(x_{\text{Ma}}, x_{\text{Na}}, x_{\text{Vo}}, x_{\text{St}}, x_{\text{Be}})\\ &\longrightarrow \exists y_{\text{Ab}} : \text{T-STUDIERENDE}(x_{\text{Ma}}, x_{\text{Na}}, x_{\text{Vo}}, x_{\text{St}}, y_{\text{Ab}})\},\\ \Omega = \{\{&\text{T-STUDIERENDE}(x_{\text{Ma}}, x_{\text{Na}}, x_{\text{Vo}}, x_{\text{St}}, y_{\text{Ab}})\\ &\longrightarrow \exists x_{\text{Be}} : \text{R-STUDIERENDE}(x_{\text{Ma}}, x_{\text{Na}}, x_{\text{Vo}}, x_{\text{St}}, x_{\text{Be}})\}\}\end{aligned}$$

gegeben. Die Instanzen I_{S_R} von S_R und I_T von T seien wie folgt gegeben (Matrikelnummer ist der Schlüssel für R-STUDIERENDE, die Relation T-STUDIERENDE hat im globalen Schema keinen Schlüssel):

$$\begin{aligned}I_{S_R} = \{&\text{R-STUDIERENDE}(1, \text{Fieber}, \text{Fabian}, \text{Informatik}, \text{Tel: } 0123456789),\\ &\text{R-STUDIERENDE}(2, \text{Sonne}, \text{Sarah}, \text{ITTI}, \eta_{\text{Be}_1})\},\\ I_T = \{&\text{T-STUDIERENDE}(1, \text{Fieber}, \text{Fabian}, \text{Informatik}, \eta_{\text{Ab}_1}),\\ &\text{T-STUDIERENDE}(2, \text{Sonne}, \text{Sarah}, \text{ITTI}, \eta_{\text{Ab}_2}),\\ &\text{T-STUDIERENDE}(2, \text{Sonne}, \text{Sarah}, \text{Informatik}, \text{Bachelor}),\\ &\text{T-STUDIERENDE}(3, \text{Mueller}, \text{Max}, \eta_{\text{St}_1}, \text{Master})\}.\end{aligned}$$

Erweiterung Für die Erweiterung von $\mathcal{M}'$ müssen wir zuerst alle Bezeichner R der S-T TGD-Kopf-Ausdrücke mit neuen Bezeichnern R_+ ersetzen, wobei das Relationenschema R_+ nicht in S_R enthalten sein darf. In unseren Fall ersetzen wir demnach den Bezeichner R-STUDIERENDE durch R-STUDIERENDE$_+$, wodurch wir unsere neue Menge von Disjunktiven-Mengen Ω_+ erhalten:

$$\begin{aligned}\Omega_+ = \{\{&\text{T-STUDIERENDE}(x_{\text{Ma}}, x_{\text{Na}}, x_{\text{Vo}}, x_{\text{St}}, y_{\text{Ab}})\\ &\longrightarrow \exists x_{\text{Be}} : \text{R-STUDIERENDE}_+(x_{\text{Ma}}, x_{\text{Na}}, x_{\text{Vo}}, x_{\text{St}}, x_{\text{Be}})\}\}.\end{aligned}$$

Indem wir zum Quelldatenbankschema S_R alle neu erzeugten Relationenschemata hinzufügen, erhalten wir unser erweitertes Quellschema[18] S_{R+}. Anschließend müssen wir die Attribute suchen, mit denen wir die Ausdrücke in den S-T TGD-Köpfen erweitern wollen. Da nur eine Disjunktive-Menge mit nur einer S-T TGD in Ω_+ existiert, müssen wir nur prüfen, ob es Variablen im Rumpf gibt, die nicht im Kopf auftauchen[19]. Da unsere inverse Schemaabbildung mit Algorithmus 4 erzeugt wurde, wissen wir, dass für uns nur die Attribute interessant sind, die in jedem S-T TGD-Rumpf durch eine y-Variable repräsentiert werden. In unseren Fall erweitern wir unseren R-STUDIERENDE$_+$-Ausdruck demnach mit der y_{Ab}-Variablen, die im S-T TGD-Rumpf das Abschluss-Attribut repräsentiert. Wir erhalten somit

$$\begin{aligned}\Omega_+ = \{\{&\text{T-STUDIERENDE}(x_{\text{Ma}}, x_{\text{Na}}, x_{\text{Vo}}, x_{\text{St}}, y_{\text{Ab}})\\ &\longrightarrow \exists x_{\text{Be}} : \text{R-STUDIERENDE}_+(x_{\text{Ma}}, x_{\text{Na}}, x_{\text{Vo}}, x_{\text{St}}, x_{\text{Be}}, y_{\text{Ab}})\}\}.\end{aligned}$$

[18] $S_{R+} = S \cup \{\text{R-STUDIERENDE}_+(\text{Matrikelnr}, \text{Name}, \text{Vorname}, \text{Studiengang}, \text{Bemerkung})\}$

[19] Entspricht der Suche nach Attributen im S-T TGD-Rumpf, die im S-T TGD-Kopf durch keine Variable repräsentiert werden. Bei mehreren S-T TGDs und Disjunktiven-Mengen müssten wir alle S-T TGDs überprüfen.

Wichtig ist, dass wir das Attribut für den Abschluss auch zum R-STUDIERENDE$_+$-Relationenschema in S_{R+} hinzufügen. Da wir nur eine Disjunktive-Menge mit einer S-T TGD in Ω_+ haben, sind wir an dieser Stelle bereits fertig mit der Erweiterung. Unser Ergebnis ist die erweiterte inverse Schemaabbildung $\mathcal{M}'_+ = (T, S_{R+}, \Omega_+)$. Führen wir $\mathcal{M}'_+$ aus, erhalten wir die Instanz I'_{S_R}, welche wir im Folgenden noch nachbereiten müssen, damit Anfragen an R-STUDIERENDE auch auf die neuen Informationen in R-STUDIERENDE$_+$ zugreifen können:

$$\begin{aligned} I'_{S_R} = \{&\text{R-STUDIERENDE}(1, \text{Fieber}, \text{Fabian}, \text{Informatik}, \text{Tel: } 0123456789),\\ &\text{R-STUDIERENDE}(2, \text{Sonne}, \text{Sarah}, \text{ITTI}, \eta_{\text{Be}_1}),\\ &\text{R-STUDIERENDE}_+(1, \text{Fieber}, \text{Fabian}, \text{Informatik}, \eta_{\text{Be}_2}, \eta_{\text{Ab}_1}),\\ &\text{R-STUDIERENDE}_+(2, \text{Sonne}, \text{Sarah}, \text{ITTI}, \eta_{\text{Be}_3}, \eta_{\text{Ab}_2}),\\ &\text{R-STUDIERENDE}_+(2, \text{Sonne}, \text{Sarah}, \text{Informatik}, \eta_{\text{Be}_4}, \text{Bachelor}),\\ &\text{R-STUDIERENDE}_+(3, \text{Mueller}, \text{Max}, \eta_{\text{St}_1}, \eta_{\text{Be}_5}, \text{Master})\}. \end{aligned}$$

Nachbereitung Im nächsten Schritt müssen wir die mit $\mathcal{M}'_+$ erzeugte Instanz I'_{S_R} noch nachbereiten. Dafür erstellen wir für R-STUDIERENDE$_+$ eine TGD und bEGD nach dem im Algorithmus 7 angegebenen Muster und fügen beide Integritätsbedingungen zu $\mathcal{N}$ wie folgt hinzu:

$$\begin{aligned} \mathcal{N} = \{&TGD: \text{R-STUDIERENDE}_+(x_{\text{Ma}}, x_{\text{Na}}, x_{\text{Vo}}, x_{\text{St}}, x_{\text{Be}}, y_{\text{Ab}}) \longrightarrow\\ &\quad \text{R-STUDIERENDE}(x_{\text{Ma}}, x_{\text{Na}}, x_{\text{Vo}}, x_{\text{St}}, x_{\text{Be}}),\\ &EGD: \text{R-STUDIERENDE}(x_{\text{Ma}}, x_{\text{Na}_1}, x_{\text{Vo}_1}, x_{\text{St}_1}, x_{\text{Be}_1}) \wedge\\ &\quad \text{R-STUDIERENDE}(x_{\text{Ma}}, x_{\text{Na}_2}, x_{\text{Vo}_2}, x_{\text{St}_2}, x_{\text{Be}_2}) \longrightarrow\\ &\quad (x_{\text{Na}_1} = x_{\text{Na}_2}), (x_{\text{Vo}_1} = x_{\text{Vo}_2}), (x_{\text{St}_1} = x_{\text{St}_2}), (x_{\text{Be}_1} = x_{\text{Be}_2})\}. \end{aligned}$$

Führen wir nun die TGD in $\mathcal{N}$ aus[20] und übertragen alle neuen Tupel von R-STUDIERENDE$_+$ nach R-STUDIERENDE, erhalten wir die Instanz I''_{S_R}. Bei der Ausführung der TGD wird für die ursprünglich in R-STUDIERENDE vorkommenden Tupel die Provenance-Annotation $|_{original}$ und für die neu aus R-STUDIERENDE$_+$ stammenden Tupel die Provenance-Annotation $|_{neu}$ hinzugefügt. Wir erhalten:

$$\begin{aligned} I''_{S_R} = \{&\text{R-STUDIERENDE}(1, \text{Fieber}, \text{Fabian}, \text{Informatik}, \text{Tel: } 0123456789)|_{original},\\ &\text{R-STUDIERENDE}(2, \text{Sonne}, \text{Sarah}, \text{ITTI}, \eta_{\text{Be}_1})|_{original},\\ &\text{R-STUDIERENDE}(1, \text{Fieber}, \text{Fabian}, \text{Informatik}, \eta_{\text{Be}_2})|_{neu},\\ &\text{R-STUDIERENDE}(2, \text{Sonne}, \text{Sarah}, \text{Informatik}, \eta_{\text{Be}_4})|_{neu},\\ &\text{R-STUDIERENDE}(3, \text{Mueller}, \text{Max}, \eta_{\text{St}_1}, \eta_{\text{Be}_5})|_{neu},\\ &\text{R-STUDIERENDE}_+(1, \text{Fieber}, \text{Fabian}, \text{Informatik}, \eta_{\text{Be}_2}, \eta_{\text{Ab}_1}),\\ &\text{R-STUDIERENDE}_+(2, \text{Sonne}, \text{Sarah}, \text{ITTI}, \eta_{\text{Be}_3}, \eta_{\text{Ab}_2}),\\ &\text{R-STUDIERENDE}_+(2, \text{Sonne}, \text{Sarah}, \text{Informatik}, \eta_{\text{Be}_4}, \text{Bachelor}),\\ &\text{R-STUDIERENDE}_+(3, \text{Mueller}, \text{Max}, \eta_{\text{St}_1}, \eta_{\text{Be}_5}, \text{Master})\}. \end{aligned}$$

Anschließend müssen die nur noch die originalen und neuen Tupel von R-STUDIERENDE mit der bEGD aus $\mathcal{N}$ bereinigen. Optional können wir die Attribute in R-STUDIERENDE$_+$ noch auf die Schlüsselattribute von R-STUDIERENDE und die neuen Attribute beschränken, um die redundante Speicherung der nicht bereinigten originalen Attribute zu umgehen. Zusätzlich könnten wir auch noch eine bEGD[21]

[20] Die Ausführung der TGD und der bEGD mittels Standard-CHASE erfolgt in nicht deterministischer Reihenfolge. Wir führen nur der Übersichtlichkeit halber zuerst die TGD aus.

[21] Wir nutzen eine bEGD anstatt einer EGD, um inkonsistente Konstanten durch einen Nullwert ersetzt zu können. Die Provenance-Annotation aller Tupel in R-STUDIERENDE$_+$ wird dabei als $|_{neu}$ betrachtet.

anwenden, die die neuen Attribute bereinigt. Die letzten beiden Schritte gehören allerdings nicht zur Erweiterung im GaLVE-Verfahren und werden hier nur genutzt, um das Ergebnis zu verkleinern. Die Bereinigung der neuen Attribute wäre zudem nicht nötig, wenn bereits im globalen Schema für die Herstellung der Konsistenz gesorgt werden würde.

Unser Endergebnis der nachbereiteten Erweiterung ist in der Instanz I'''_{S_R} dargestellt. Die Tupel, die bei der Ausführung der bEGD in allen Attributen gleich gesetzt wurden, haben die Provenance-Annotation $|_{original}$ behalten:

$$\begin{aligned} I'''_{S_R} = \{&\text{R-STUDIERENDE}(1, \text{Fieber}, \text{Fabian}, \text{Informatik}, \text{Tel: } 0123456789)|_{original}, \\ &\text{R-STUDIERENDE}(2, \text{Sonne}, \text{Sarah}, \text{ITTI}, \eta_{\text{Be}_1})|_{original}, \\ &\text{R-STUDIERENDE}(3, \text{Mueller}, \text{Max}, \eta_{\text{St}_1}, \eta_{\text{Be}_5})|_{neu}, \\ &\text{R-STUDIERENDE}_+(1, \eta_{\text{Ab}_1}), \\ &\text{R-STUDIERENDE}_+(2, \text{Bachelor}), \\ &\text{R-STUDIERENDE}_+(3, \text{Master})\}. \end{aligned}$$

Im nächsten Abschnitt werden wir das gesamte GaLVE-Verfahren noch einmal zusammenfassen. Neben einer kurzen Betrachtung von möglichen virtuellen Umsetzungen des GaLVE-Verfahrens folgt auch noch ein großes und komplexeres Beispiel für die Techniken im GaLVE-Verfahren mit mehreren Disjunktiven-Mengen und mehreren S-T TGDs in der Schemaabbildung.

4.4. Zusammenfassung GaLVE-Verfahren

In den vorherigen Abschnitten haben wir alle Techniken vorgestellt, die wir in unserem neuen GaLVE-Verfahren verwenden wollen. Abschnitt 4.2.1 liefert uns mit dem Algorithmus 4 die Möglichkeit beliebige Schemaabbildungen zu invertieren. Die in Abschnitt 4.2.2 vorgestellten Theorien zum angepassten Disjunktiven-CHASE (siehe Definition 4.4) und der Sicheren Instanz (siehe Definition 4.7) helfen uns bei der Verarbeitung der inversen Schemaabbildung. In Abschnitt 4.3.1 haben wir mit Algorithmus 6 eine Berechnungsvorschrift für die Erweiterung einer inversen Schemaabbildung vorgestellt, sodass diese eine Quelldatenbank um neue Tupel und Attribute aus einer globalen Datenbank erweitern kann, ohne dabei die originalen Relationen zu verändern. Darauf aufbauend wurde in Abschnitt 4.3.2 gezeigt, wie diese erweiterte inverse Schemaabbildung nachbereitet werden muss, um die neuen Tupel zusammen mit den originalen Tupeln in bestehenden Anfragen direkt nutzen zu können. In diesem Abschnitt fassen wir nun das komplette GaLVE-Verfahren noch einmal zusammen (siehe Abschnitt 4.4.1) und werden auch einige weitere Ansätze dessen Umsetzung betrachten, bei denen Teile des GaLVE-Verfahrens virtuell anstatt materialisiert ablaufen (siehe Abschnitt 4.4.2). Den Abschluss bildet ein großes Beispiel des GaLVE-Verfahrens (siehe Abschnitt 4.4.3) auf der Basis unseres allgemeinen Beispiels aus Abschnitt 1.1.

4.4.1. Das komplette GaLVE-Verfahren

Das GaLVE-Verfahren unterliegt den in diesem Kapitel vorgestellten Annahmen und Einschränkungen. So erfolgt noch vor dem eigentlichen GaLVE-Verfahren eine Datenintegration von mehreren Quelldatenbanken in eine globale Zieldatenbank. Ausgehend von jeder Quelldatenbank erfolgt die Datenintegration dabei mit einer Schemaabbildung von dem entsprechenden Quellschema in das globale Zielschema. Für das GaLVE-Verfahren, so wie wir es bisher beschrieben haben, ist wichtig, dass die Quelldatenbanken nach der Datenintegration weiterhin existieren und die globale Datenbank materialisiert vorliegt. Weitere Varianten des GaLVE-Verfahrens, bei denen z. B. die globale Datenbank nur virtuell vorliegt, besprechen wir im nächsten Abschnitt.

An dieser Stelle startet das GaLVE-Verfahren für eine der Schemaabbildungen, die zuvor in der Datenintegration ausgeführt wurden. Da wir das GaLVE-Verfahren immer nur für eine Schemaabbildung von einem Quellschema ins globale Schema betrachten, erweitern wir in einem Durchlauf des GaLVE-Verfahrens auch nur eine Quelldatenbank um neue Tupel und Attribute aus der globalen Datenbank. Sollen mehrere Quellen erweitert werden, muss das GaLVE-Verfahren für jede dieser Quellen mit der entsprechenden Schemaabbildung einzeln ausgeführt werden.

Schritte des GaLVE-Verfahrens Zusammengefasst besteht das GaLVE-Verfahren aus den folgenden sieben Schritten. Die Motivation hinter den Schritten werden wir hier nicht mehr wiederholen, sondern nur angeben, in welchem Abschnitt oder in welcher Definition wir welche Technik eingeführt haben.

1.) Der erste Schritt im GaLVE-Verfahren ist die **Invertierung** der Schemaabbildung $\mathcal{M} = (S, T, \Sigma)$ (siehe Definition 2.10) vom Quellschema S der ausgewählten Quelldatenbank ins globale Schema T der globalen Datenbank mit den S-T TGDs (siehe Definition 2.7) in Σ. Dafür nutzen wir den Algorithmus 4, um eine Adapted Strong Maximum Extended Recovery $\mathcal{M}' = (T, S, \Omega)$ (siehe Definition 4.2) zu berechnen, deren S-T TGDs in den Disjunktive-Mengen (siehe Definition 4.1) in Ω, von dem globalen Schema T zurück in das Quellschema S abbilden.

2.) Da wir im Kontext des GaLVE-Verfahrens wissen, dass wir für das im 4. Schritt zu berechnende Ergebnis der inversen Schemaabbildung, im 5. Schritt die Sichere Instanz (siehe Definition 4.7) berechnen werden, können wir die Disjunktiven-Mengen in Ω im zweiten Schritt des GaLVE-Verfahrens **optimieren**. Dafür löschen wir mit den in Definition 4.8 angegebenen Regeln die Disjunktiven-Mengen, die nicht zu den Tupeln der Sicheren Instanz beitragen können.

3.) Im dritten Schritt des GaLVE-Verfahrens folgt die **Erweiterung** der inversen Schemaabbildung $\mathcal{M}' = (T, S, \Omega)$. Wir nutzen den Algorithmus 6, um aus der inversen Schemaabbildung $\mathcal{M}'$ eine erweiterte inverse Schemaabbildung $\mathcal{M}'_+ = (T, S_+, \Omega_+)$ zu erzeugen. Diese fügt in zusätzlichen, für das Quellschema neuen Relationen neue Tupel und Attribute aus der globalen Datenbank zu der Quelldatenbank hinzu. Die neuen Tupel und Attribute stammen aus den Relationen der globalen Datenbank, in die die Schemaabbildung aus dem ersten Schritt Daten integrieren würde. Die zusätzlichen, für das Quellschema neuen Relationen werden jeweils für eine originale Relation des Quellschemas erstellt. Die originalen Relationen entsprechen denen, aus denen die Schemaabbildung aus dem ersten Schritt Daten integrieren würde. Das erweiterte Quellschema S_+ besteht aus der Vereinigung des originalen Schemas S und den zusätzlichen, für das Quellschema neuen Relationen. Die erweiterten Disjunktiven-Mengen Ω_+ bilden nur noch auf die zusätzlichen, für das Quellschema neuen Relationen ab, sodass die originalen Relationen erhalten bleiben.

4.) Der vierte Schritt des GaLVE-Verfahrens führt die erweiterte inverse Schemaabbildung aus. Für die Verarbeitung der in den Disjunktiven-Mengen der erweiterten inversen Schemaabbildung enthaltenden S-T TGDs, haben wir in der Definition 4.4 den **angepassten Disjunktiven-CHASE** eingeführt. Das Ergebnis des vierten Schrittes kann aufgrund der Disjunktiven-Mengen eine Menge von Instanzen sein.

5.) Um aus der Menge von Instanzen eine **Sichere Instanz** (siehe Definition 4.7) mit gültigen Informationen zu berechnen, nutzen wir im fünften Schritt des GaLVE-Verfahrens den im Algorithmus 5 implementierten Durchschnitt unter einem Homomorphismus (siehe Definition 4.6). Die Sichere Instanz besteht an dieser Stelle aus den neuen Tupeln und Attributen in den zusätzlichen, für das Quellschema neuen Relationen aus dem dritten Schritt. Mit ihnen können wir nun die originalen Relationen in der Quelldatenbank erweitern.

6.) Der sechste Schritt des GaLVE-Verfahrens erstellt die **TGDs** (siehe Definition 2.7) und **bEGDs** (siehe Definition 4.9) die nötig sind, um die bisherige Erweiterung der zusätzlichen, für das Quellschema neuen Relationen nachzubereiten. Die TGDs fügen die neuen Tupel, die in den zusätzlichen, für das Quellschema neuen Relationen enthalten sind, zu den entsprechenden originalen Relationen hinzu. Die bEGDs bereinigen die originalen und neuen Tupel in den originalen Relationen, wobei die Informationen in den originalen Tupeln höherwertig angesehen werden. Der Algorithmus 7 berechnet uns eine entsprechende TGD und bEGD für jede zusätzliche, für das Quellschema neue Relation.

7.) Im letzten Schritt des GaLVE-Verfahrens führen wir die TGDs und bEGDs der **Nachbereitung** aus. Damit wir den Standard-CHASE (siehe Definition 2.18) auch für die neu definierten bEGDs anwenden können, haben wir den befangenen CHASE-Schritt auf Instanzen (siehe Definition 4.10) eingeführt. Dieser wird anstatt des Standard-CHASE-Schrittes (siehe Definition 2.17) ausgeführt, sobald im Standard-CHASE eine bEGDs verarbeitet werden soll.

Das GaLVE-Verfahren endet mit der Ausgabe der gesamten erweiterten Quelldatenbank, die unter dem erweiterten Quellschema S_+ (siehe dritter Schritt) definiert ist. Die originalen Relationen enthalten alle für die Quelle interessanten Informationen aus dem globalen Schema, was den neuen Tupeln und den um Informationen aus den neuen Tupeln erweiterten originalen Tupeln entspricht. Bestehende Anfragen an die originalen Relationen können ohne weitere Anfragetransformationen direkt auf die originalen und neuen Tupel zugreifen. Neue Anfragen können auf dem gewohnten Quellschema S entwickelt werden, da S in S_+ enthalten ist. Die zusätzlichen, für das Quellschema neuen Relationen enthalten die neuen Attribute für die originalen Relationen. Wir können diese Relationen nach dem GaLVE-Verfahren minimieren, indem wir sie auf die Verbundattribute zu den originalen Relationen und den neuen Attributen projizieren. Um in diesen Relationen konsistente Daten zu gewährleisten, müssen wir die neuen Attribute noch mit einer weiteren bEGD bereinigen. Im Abschnitt 4.3.3 haben wir das bereits an einem Beispiel gezeigt. Die zusätzlichen, für das Quellschema neuen Relationen können auf Wunsch auch gelöscht werden, falls die in ihnen gespeicherten neuen Attributen uninteressant für die Quelle sein sollten.

Im Abschnitt 4.4.3 gehen wir jeden Schritt des GaLVE-Verfahrens für unser allgemeines Beispiel aus Abschnitt 1.1 durch. Im nächsten Abschnitt folgt die Betrachtung weiterer GaLVE-Varianten, die sich aus der virtuellen Umsetzung der Datenintegration bzw. des GaLVE-Verfahrens ergeben.

4.4.2. Weitere GaLVE-Varianten

Bisher sind wir im GaLVE-Verfahren immer von einer materialisierten Datenbankintegration und einer materialisierten Erweiterung der Quelldatenbank ausgegangen. Dabei wäre es durchaus möglich, ein GaLVE-Verfahren zu entwickeln, das auf einer virtuellen Datenbankintegration aufsetzt bzw. die Quelldatenbank nur virtuell erweitert. Die sich daraus gegebenen Varianten des GaLVE-Verfahrens sind in der folgenden Aufreihung dargestellt:

1. **M**aterialisierte Datenintegration und **m**aterialisierte erweiterte Quelldatenbank
2. **M**aterialisierte Datenintegration und **v**irtuell erweiterte Quelldatenbank
3. **V**irtuelle Datenintegration und **m**aterialisierte erweiterte Quelldatenbank
4. **V**irtuelle Datenintegration und **v**irtuell erweiterte Quelldatenbank

Die 1. Variante spiegelt das GaLVE-Verfahren dieser Arbeit mit den in diesem Kapitel vorgestellten Techniken wieder. Eine materialisierte Datenintegration in Zusammenhang mit einer materialisierten erweiterten Quelldatenbank hat den Vorteil, dass wir keine S-T TGDs komponieren müssen und so in Logik 1. Ordnung bleiben können. Die Schemaabbildungen von den Quellschemata in das globale Schema werden vor dem GaLVE-Verfahren direkt ausgeführt und erzeugen eine materialisierte globale Datenbank, auf der wir auch die erweiterte inverse Schemaabbildung direkt anwenden können. Nach der materialisierten Erweiterung der Quelldatenbank und der Ausführung der Nachbereitung der erweiterten inversen Schemaabbildung können Anfragen direkt an die erweiterte Quelldatenbank gestellt werden.

Zusammengefasst können für eine Quelldatenbank die Schemaabbildung aus dem Quellschema, die erweiterte inverse Schemaabbildung ins Quellschema und die Anfragen auf dem Quellschema direkt ausgeführt werden. Für die Ausführung der Schemaabbildung, der erweiterten inversen Schemaabbildung und der Anfragen können wir daher den in dieser Arbeit definierten Standard-CHASE (siehe Definition 2.18) und angepassten Disjunktiven-CHASE (siehe Definition 4.4) verwenden.

Wollen wir ein GaLVE-Verfahren auf einer virtuellen Datenintegration aufsetzen, wie es in den Varianten 3 und 4 beschrieben ist, müssen wir in erster Linie mit den typischen Problemen der virtuellen Datenintegration arbeiten, die sich auch ohne ein nachfolgendes GaLVE-Verfahren ergeben würden. Führen wir die virtuelle Erzeugung der globalen Datenbank und die inverse Schemaabbildung hintereinander aus, ändert sich am Ablauf und an den Techniken des GaLVE-Verfahrens nichts. Da die globale Datenbank nicht direkt zur Verfügung steht, müssen wir diese dann allerdings vor jeder Ausführung einer inversen Schemaabbildung erneut berechnen und für diese zwischenspeichern.

Ist das Zwischenspeichern der globalen Datenbank nicht möglich, können wir dies mit der Komposition der Schemaabbildungen aus den Quellschemata in das globale Datenbankschema und der inversen Schemaabbildungen aus dem globalen Datenbankschema zurück in ein Quellschema vermeiden. Mit der Komposition erzeugen wir allerdings Formeln in Logik 2. Ordnung (ähnlich der Second-Order TGDs in Definition 2.13), die wir dann nicht mehr mit den vorgestellten CHASE-Algorithmen verarbeiten können. Diese Komposition mit Disjunktiven-Mengen müsste ebenfalls genauer untersucht werden. Was fest steht, ist, dass diese Komposition der S-T TGDs in den Schemaabbildungen und den Disjunktiven-Mengen in der inversen Schemaabbildung wieder Disjunktive-Mengen ergeben würde, wodurch auch ein CHASE-Algorithmus mit Second-Order TGDs, wie er z. B. in [FKPT05] vorgestellt wird, ähnlich wie der angepasste Disjunktive-CHASE (siehe Definition 4.4) angepasst werden müsste.

Ähnliche Probleme mit Disjunktiven-Mengen in Logik 2. Ordnung bekommen wir auch bei den nötigen Kompositionen in der virtuellen Umsetzung des GaLVE-Verfahrens für eine virtuelle Erweiterung der Quelle, wie sie in den Varianten 2 und 4 dargestellt ist. Auch hier könnten wir wieder jeden Schritt des GaLVE-Verfahrens einzeln berechnen und das Ergebnis für den nächsten Schritt zwischenspeichern, um die vorgestellten GaLVE-Techniken verwenden zu können. Das ist allerdings nicht immer möglich bzw. nicht immer erwünscht. Bei einer Komposition müssen wir in diesem Fall nicht nur die Disjunktiven-Mengen der inversen Schemaabbildung mir den TGDs und bEGDs der Nachbereitung komponieren, sondern auch die Anfragen an das erweiterte Quellschema dazu fügen. Die Berechnung der Sicheren Instanz müsste dann auf der Menge der Anfrageergebnisse erfolgen. Bei der 4. Variante würde zusätzlich noch die beschriebene Komposition mit den Schemaabbildungen von den Quellschemata ins globale Schema dazukommen. Variante 2 und vor allem die Variante 4 des GaLVE-Verfahrens sind daher nicht zu empfehlen, da der Rechenaufwand und der Primärspeicherbedarf bei der Berechnung einer Anfrage verhältnismäßig zu groß sind, im Gegensatz zum Sekundärspeicherverbrauch der materialisierten GaLVE-Variante, bei der Anfragen direkt an ein erweitertes Quellschema gestellt werden können.

So wie die einzelnen Schritte des GaLVE-Verfahrens in dieser Arbeit vorgestellt wurden, sind sie nicht für die Komposition untereinander geeignet. Mindestens das eigentliche GaLVE-Verfahren mit der Erweiterung der Quelldatenbank sollte daher vollkommen materialisiert umgesetzt werden, sodass die einzigen sinnvollen Varianten die Variante 1 und die Variante 3 mit Zwischenspeichern der globalen Datenbank sind. Das Beispiel im nächsten Abschnitt wird daher wieder für unser vorgestelltes, voll materialisiertes GaLVE-Verfahren angegeben.

4.4.3. Großes Beispiel

Zum Abschluss des Konzept-Kapitels werden wir das komplette GaLVE-Verfahren an einem größeren Beispiel durchgehen. Die genaue Anwendung der einzelnen Techniken werden wir dabei nicht wiederholen und verweisen dafür auf die Abschnitte (konkrete Abschnitte siehe Abschnitt 4.4.1) in denen die Techniken detailliert vorgestellt wurden. In diesem Abschnitt geht es nur um die Ergebnisse der einzelnen GaLVE-Schritte in einem zusammenhängenden Beispiel. Das Beispiel orientiert sich dabei an unserem allgemeinen Beispiel aus dem Abschnitt 1.1. Wir werden das dort vorgestellte Schema und die Daten

allerdings an einigen Stellen leicht abändern, um ein paar interessantere Fälle für das GaLVE-Verfahren zu erzeugen.

Seinen die Schemata der Kieler und Rostocker Uni-Datenbanken S_K und S_R mit den zugehörigen Instanzen I_{S_K} und I_{S_R} sowie das globale Datenbankschema T wie folgt gegeben:

$$
\begin{aligned}
S_K = \{&\text{K-STUDIERENDE} = \{\underline{\text{Matrikelnr}}, \text{Nachname}, \text{Vorname}, \text{Abschluss}\},\\
&\text{K-NOTEN} = \{\underline{\text{Modulnr, Matrikelnr}}, \text{Semester}, \text{Note}, \text{Bemerkung}\},\\
&\text{K-PRAKTIKUM} = \{\underline{\text{Matrikelnr}}\},\\
&\text{K-SOFTSKILL} = \{\underline{\text{Matrikelnr}}\}\},
\end{aligned}
$$

$$
\begin{aligned}
S_R = \{&\text{R-STUDIERENDE} = \{\underline{\text{Matrikelnr}}, \text{Nachname}, \text{Vorname}, \text{Studiengang}, \text{Bemerkung}\},\\
&\text{R-TEILNAHME} = \{\underline{\text{Teilnahmenr}}, \text{Modulnr}, \text{Matrikelnr}\},\\
&\text{R-NOTEN} = \{\underline{\text{Teilnahmenr}}, \text{Semester}, \text{Note}\},\\
&\text{R-ZUSATZ} = \{\underline{\text{Matrikelnr, Leistung}}\}\},
\end{aligned}
$$

$$
\begin{aligned}
T = \{&\text{T-STUDIERENDE} = \{\text{Matrikelnr}, \text{Nachname}, \text{Vorname}, \text{Studiengang}, \text{Abschluss}\},\\
&\text{T-NOTEN} = \{\text{Modulnr}, \text{Matrikelnr}, \text{Semester}, \text{Note}, \text{Bemerkung}\},\\
&\text{T-ZUSATZ} = \{\text{Matrikelnr}\}\},
\end{aligned}
$$

$$
\begin{aligned}
I_{S_K} = \{&\text{K-STUDIERENDE}(2, \text{Sonnensand}, \eta_{\text{Vo}_1}, \text{Bachelor}),\\
&\text{K-STUDIERENDE}(3, \eta_{\text{Na}_1}, \text{Max}, \text{Master}),\\
&\text{K-NOTEN}(2, 3, \eta_{\text{Se}_1}, 2.0, \eta_{\text{Be}_1}),\\
&\text{K-NOTEN}(2, 2, \text{SS } 21, 4.0, \text{Knapp}),\\
&\text{K-PRAKTIKUM}(3),\\
&\text{K-SOFTSKILL}(2),\\
&\text{K-SOFTSKILL}(3)\},
\end{aligned}
$$

$$
\begin{aligned}
I_{S_R} = \{&\text{R-STUDIERENDE}(1, \text{Fieber}, \text{Fabian}, \text{Informatik}, \text{Tel: } 0123456789),\\
&\text{R-STUDIERENDE}(2, \text{Sonne}, \text{Sarah}, \text{ITTI}, \text{Corona}(+) \text{ am } 07.12.2020),\\
&\text{R-TEILNAHME}(1, 1, 1),\\
&\text{R-TEILNAHME}(2, 2, 1),\\
&\text{R-TEILNAHME}(3, 1, 2),\\
&\text{R-NOTEN}(1, \text{WS } 20/21, 1.7),\\
&\text{R-NOTEN}(2, \text{SS } 21, 3.0),\\
&\text{R-NOTEN}(1, \text{WS } 20/21, 2.3),\\
&\text{R-ZUSATZ}(1, \text{Praktikum}),\\
&\text{R-ZUSATZ}(1, \text{Softskill}),\\
&\text{R-ZUSATZ}(2, \text{Praktikum})\}.
\end{aligned}
$$

Das globale Uni-Datenbankschema T hat in unserem Beispiel keine Schlüsselattribute. Die Bereinigung der Daten wird daher erst mit den bEGDs während der Nachbereitung der Erweiterung im 7. Schritt des GaLVE-Verfahrens erfolgen. Eine Veränderung in diesem Beispiel im Gegensatz zum ursprünglichen Beispiel in Abschnitt 1.1 ist das Attribut Bemerkung in K-NOTEN. Interessant ist, dass wir den Attributwert von Bemerkung in K-NOTEN nicht direkt ins Bemerkung-Attribut von T-NOTEN abbilden werden, sondern in T-NOTEN einen konstanten Wert für jedes Tupel eintragen. Als Schemaabbildungen definieren wir $\mathcal{M}_K = (S_K, T, \Sigma_K)$ vom Kieler Uni-Datenbankschema S_K und $\mathcal{M}_R = (S_R, T, \Sigma_R)$ vom Rostocker Uni-Datenbankschema S_R ins globale Uni-Datenbankschema T mit den jeweiligen S-T TGDs in Σ_K und Σ_R wie folgt:

$$
\begin{aligned}
\Sigma_K =& \{\sigma_{K1}, \sigma_{K2}, \sigma_{K3}, \sigma_{K4}\} \text{ mit} \\
\sigma_{K1} =& \text{K-STUDIERENDE}(x_{\text{Ma}}, x_{\text{Na}}, x_{\text{Vo}}, x_{\text{Ab}}) \\
& \longrightarrow \exists y_{\text{St}} : \text{T-STUDIERENDE}(x_{\text{Ma}}, x_{\text{Na}}, x_{\text{Vo}}, y_{\text{St}}, x_{\text{Ab}}), \\
\sigma_{K2} =& \text{K-NOTEN}(x_{\text{Mo}}, x_{\text{Ma}}, x_{\text{Se}}, x_{\text{No}}, x_{\text{Be}}) \\
& \longrightarrow \text{T-NOTEN}(x_{\text{Mo}}, x_{\text{Ma}}, x_{\text{Se}}, x_{\text{No}}, \text{"Von Kiel importiert"}) \wedge \text{T-ZUSATZ}(x_{\text{Ma}}), \\
\sigma_{K3} =& \text{K-PRAKTUKUM}(x_{\text{Ma}}) \longrightarrow \text{T-ZUSATZ}(x_{\text{Ma}}), \\
\sigma_{K4} =& \text{K-SOFTSKILL}(x_{\text{Ma}}) \longrightarrow \text{T-ZUSATZ}(x_{\text{Ma}}),
\end{aligned}
$$

$$
\begin{aligned}
\Sigma_R =& \{\sigma_{R1}, \sigma_{R2}, \sigma_{R3}\} \text{ mit} \\
\sigma_{R1} =& \text{R-STUDIERENDE}(x_{\text{Ma}}, x_{\text{Na}}, x_{\text{Vo}}, x_{\text{St}}, x_{\text{Be}}) \\
& \longrightarrow \exists y_{\text{Ab}} : \text{T-STUDIERENDE}(x_{\text{Ma}}, x_{\text{Na}}, x_{\text{Vo}}, x_{\text{St}}, y_{\text{Ab}}), \\
\sigma_{R2} =& \text{R-TEILNAHME}(x_{\text{Te}}, x_{\text{Mo}}, x_{\text{Ma}}) \wedge \text{R-NOTEN}(x_{\text{Te}}, x_{\text{Se}}, x_{\text{No}}) \\
& \longrightarrow \exists y_{\text{Be}} : \text{T-NOTEN}(x_{\text{Mo}}, x_{\text{Ma}}, x_{\text{Se}}, x_{\text{No}}, y_{\text{Be}}) \wedge \text{T-ZUSATZ}(x_{\text{Ma}}), \\
\sigma_{R3} =& \text{R-ZUSATZ}(x_{\text{Ma}}, x_{\text{Le}}) \longrightarrow \text{T-ZUSATZ}(x_{\text{Ma}}).
\end{aligned}
$$

Vorbereitung - Datenintegration Als erstes müssen wir noch vor dem Beginn des eigentlichen GaLVE-Verfahrens die Datenintegration von den Quellschemata S_K und S_R ins globale Schema T ausführen. Dafür wenden wir die Schemaabbildung $\mathcal{M}_K$ auf die Instanz I_{S_K} und die Schemaabbildung $\mathcal{M}_R$ auf die Instanz I_{S_R} an. Das Ergebnis der Schemaabbildungen berechnen wir mit dem Standard-CHASE (siehe Definition 2.18 und 2.19). Die Instanzen in den Quellen bleiben dabei natürlich bestehen, um sie später im GaLVE-Verfahren zu erweitern. Die Ausführung des CHASEs mit den S-T TGDs in Σ_K auf der Instanz I_{S_K} liefert die Instanz I_{T_K} und mit den S-T TGDs in Σ_R auf der Instanz I_{S_R} die Instanz I_{T_R} als Ergebnis. Beide Ergebnisinstanzen sind unter dem globalen Schema T definiert und ergeben sich wie folgt:

$$
\begin{aligned}
I_{T_K} = \{&\text{T-STUDIERENDE}(2, \text{Sonnensand}, \eta_{\text{Vo}_1}, \eta_{\text{St}_1}, \text{Bachelor}), \\
&\text{T-STUDIERENDE}(3, \eta_{\text{Na}_1}, \text{Max}, \eta_{\text{St}_2}, \text{Master}), \\
&\text{T-NOTEN}(2, 3, \eta_{\text{Se}_1}, 2.0, \text{"Von Kiel importiert"}), \\
&\text{T-NOTEN}(2, 2, \text{SS 21}, 4.0, \text{"Von Kiel importiert"}), \\
&\text{T-ZUSATZ}(2), \text{T-ZUSATZ}(3)\},
\end{aligned}
$$

$$
\begin{aligned}
I_{T_R} = \{&\text{T-STUDIERENDE}(1, \text{Fieber}, \text{Fabian}, \text{Informatik}, \eta_{\text{Ab}_1}),\\
&\text{T-STUDIERENDE}(2, \text{Sonne}, \text{Sarah}, \text{ITTI}, \eta_{\text{Ab}_2}),\\
&\text{T-NOTEN}(1, 1, \text{WS } 20/21, 1.7, \eta_{\text{Be}_1}),\\
&\text{T-NOTEN}(2, 1, \text{SS } 21, 3.0, \eta_{\text{Be}_2}),\\
&\text{T-NOTEN}(1, 2, \text{WS } 20/21, 2.3, \eta_{\text{Be}_3}),\\
&\text{T-ZUSATZ}(1), \text{T-ZUSATZ}(2)\}.
\end{aligned}
$$

Die Ergebnisinstanzen der einzelnen Schemaabbildungen müssen wir jetzt nur noch vereinigen. Da wir keine Schlüssel im globalen Schema haben, müssen wir die entstehenden inkonsistenten Daten nicht weiter beachten. Die Ergebnisinstanz I_T der gesamten Datenintegration der Schemaabbildungen $\mathcal{M}_K$ und $\mathcal{M}_R$ ist diejenige mit der wir nun das eigentliche GaLVE-Verfahren starten werden:

$$
\begin{aligned}
I_T = \{&\text{T-STUDIERENDE}(1, \text{Fieber}, \text{Fabian}, \text{Informatik}, \eta_{\text{Ab}_1}),\\
&\text{T-STUDIERENDE}(2, \text{Sonne}, \text{Sarah}, \text{ITTI}, \eta_{\text{Ab}_2}),\\
&\text{T-STUDIERENDE}(2, \text{Sonnensand}, \eta_{\text{Vo}_1}, \eta_{\text{St}_1}, \text{Bachelor}),\\
&\text{T-STUDIERENDE}(3, \eta_{\text{Na}_1}, \text{Max}, \eta_{\text{St}_2}, \text{Master}),\\
&\text{T-NOTEN}(1, 1, \text{WS } 20/21, 1.7, \eta_{\text{Be}_1}),\\
&\text{T-NOTEN}(2, 1, \text{SS } 21, 3.0, \eta_{\text{Be}_2}),\\
&\text{T-NOTEN}(1, 2, \text{WS } 20/21, 2.3, \eta_{\text{Be}_3}),\\
&\text{T-NOTEN}(2, 3, \eta_{\text{Se}_1}, 2.0, \text{"Von Kiel importiert"}),\\
&\text{T-NOTEN}(2, 2, \text{SS } 21, 4.0, \text{"Von Kiel importiert"}),\\
&\text{T-ZUSATZ}(1), \text{T-ZUSATZ}(2), \text{T-ZUSATZ}(3)\}.
\end{aligned}
$$

1. GaLVE-Schritt Da das GaLVE-Verfahren immer nur für eine Quelle gleichzeitig ausgeführt wird, wählen wir in unserem Beispiel die Kieler Uni-Datenbank als Quelle, die wir mit dem GaLVE-Verfahren erweitern wollen. Im ersten Schritt des GaLVE-Verfahrens invertieren wir daher die Schemaabbildung $\mathcal{M}_K = (S_K, T, \Sigma_K)$. Die so entstehende inverse Schemaabbildung $\mathcal{M}' = (T, S_K, \Omega)$ bildet nun vom globalen Schema T zurück in das Quellschema S_K ab. Die Köpfe der S-T TGDs aus Σ_K werden dazu normalisiert, dann invertiert und anschließend, geordnet nach Relationenschema in den neuen Rümpfen, in den Disjunktive-Mengen in Ω gespeichert. Die Disjunktiven-Mengen in Ω aus der inversen Schemaabbildung $\mathcal{M}'$ lauten:

$$
\begin{aligned}
\Omega &= \{\omega_1, \omega_2, \omega_3\} \text{ mit}\\
\omega_1 &= \{\text{T-STUDIERENDE}(x_{\text{Ma}}, x_{\text{Na}}, x_{\text{Vo}}, y_{\text{St}}, x_{\text{Ab}})\\
&\qquad \longrightarrow \text{K-STUDIERENDE}(x_{\text{Ma}}, x_{\text{Na}}, x_{\text{Vo}}, x_{\text{Ab}})\},\\
\omega_2 &= \{\text{T-NOTEN}(x_{\text{Mo}}, x_{\text{Ma}}, x_{\text{Se}}, x_{\text{No}}, \text{"Von Kiel importiert"})\\
&\qquad \longrightarrow \exists y_{\text{Be}} : \text{K-NOTEN}(x_{\text{Mo}}, x_{\text{Ma}}, x_{\text{Se}}, x_{\text{No}}, y_{\text{Be}})\},\\
\omega_3 &= \{\text{T-ZUSATZ}(x_{\text{Ma}})\\
&\qquad \longrightarrow \exists y_{\text{Mo}}, y_{\text{Se}}, y_{\text{No}}, y_{\text{Be}} : \text{K-NOTEN}(y_{\text{Mo}}, x_{\text{Ma}}, y_{\text{Se}}, y_{\text{No}}, y_{\text{Be}}),\\
&\qquad \text{T-ZUSATZ}(x_{\text{Ma}}) \longrightarrow \text{K-PRAKTUKUM}(x_{\text{Ma}}),\\
&\qquad \text{T-ZUSATZ}(x_{\text{Ma}}) \longrightarrow \text{K-SOFTSKILL}(x_{\text{Ma}})\}.
\end{aligned}
$$

2. GaLVE-Schritt Im zweiten Schritt des GaLVE-Verfahrens folgt normalerweise die Optimierung der Disjunktiven-Mengen für den angepassten Disjunktiven-CHASE in Verbindung mit der Sicheren Instanz. Diese liefert in unserem Beispiel die gleiche Menge Ω, wie wir sie im vorherigen Schritt angegeben haben. Das liegt daran, dass die Durchschnitte der Relationenschemata in den einzelnen Köpfen der S-T TGDs von ω_1 und ω_2 nicht leer sind und dass es für ω_3 mit ω_2 eine Disjunktive-Menge gibt, für die ein Relationenschema (K-NOTEN) aus dem nicht leeren Durchschnitt auch in den Köpfen der S-T TGDs in ω_3 vorkommt.

3. GaLVE-Schritt Im dritten Schritt des GaLVE-Verfahrens folgt die Erweiterung der inversen Schemaabbildung $\mathcal{M}' = (T, S_K, \Omega)$ hin zur erweiterten inversen Schemaabbildung $\mathcal{M}'_+ = (T, S_{K+}, \Omega_+)$. Das erweiterte Quellschema S_{K+} besteht aus der Vereinigung des originalen Schemas S und zusätzlichen, für das Quellschema neue Relationen, die wir für jede ursprünglich vorhandene Relation erstellen. Bemerkenswert ist hier, dass ein Attribut für den Studiengang zu der neuen Relation K-STUDIERENDE$_+$ als interessantes Attribut des globalen Schemas hinzugefügt wurde, da die Information zu dem Studiengang zuvor nicht im lokalen Schema zur Verfügung stand. Die erweiterten Disjunktiven-Mengen in Ω_+ bilden jetzt nur noch auf die zusätzlichen, für das Quellschema neuen Relationen in S_{K+} ab. Die Disjunktiven-Mengen in Ω_+ und das erweiterte Quellschema S_{K+} ergeben sich wie folgt:

$$
\begin{aligned}
\Omega_+ &= \{\omega_{1+}, \omega_{2+}, \omega_{3+}\} \text{ mit} \\
\omega_{1+} &= \{\text{T-STUDIERENDE}(x_{\text{Ma}}, x_{\text{Na}}, x_{\text{Vo}}, y_{\text{St}}, x_{\text{Ab}}) \\
&\qquad \longrightarrow \text{K-STUDIERENDE}_+(x_{\text{Ma}}, x_{\text{Na}}, x_{\text{Vo}}, x_{\text{Ab}}, y_{\text{St}})\}, \\
\omega_{2+} &= \{\text{T-NOTEN}(x_{\text{Mo}}, x_{\text{Ma}}, x_{\text{Se}}, x_{\text{No}}, \text{"Von Kiel importiert"}) \\
&\qquad \longrightarrow \exists y_{\text{Be}} : \text{K-NOTEN}_+(x_{\text{Mo}}, x_{\text{Ma}}, x_{\text{Se}}, x_{\text{No}}, y_{\text{Be}})\}, \\
\omega_{3+} &= \{\text{T-ZUSATZ}(x_{\text{Ma}}) \\
&\qquad \longrightarrow \exists y_{\text{Mo}}, y_{\text{Se}}, y_{\text{No}}, y_{\text{Be}} : \text{K-NOTEN}_+(y_{\text{Mo}}, x_{\text{Ma}}, y_{\text{Se}}, y_{\text{No}}, y_{\text{Be}}), \\
&\qquad \text{T-ZUSATZ}(x_{\text{Ma}}) \longrightarrow \text{K-PRAKTUKUM}_+(x_{\text{Ma}}), \\
&\qquad \text{T-ZUSATZ}(x_{\text{Ma}}) \longrightarrow \text{K-SOFTSKILL}_+(x_{\text{Ma}})\},
\end{aligned}
$$

$$
\begin{aligned}
S_{K+} = \{&\text{K-STUDIERENDE} = \{\underline{\text{Matrikelnr}}, \text{Nachname, Vorname, Abschluss}\}, \\
&\text{K-NOTEN} = \{\underline{\text{Modulnr, Matrikelnr}}, \text{Semester, Note, Bemerkung}\}, \\
&\text{K-PRAKTIKUM} = \{\underline{\text{Matrikelnr}}\}, \\
&\text{K-SOFTSKILL} = \{\underline{\text{Matrikelnr}}\} \\
&\text{K-STUDIERENDE}_+ = \{\text{Matrikelnr, Nachname, Vorname, Abschluss, Studiengang}\}, \\
&\text{K-NOTEN}_+ = \{\text{Modulnr, Matrikelnr, Semester, Note, Bemerkung}\}, \\
&\text{K-PRAKTIKUM}_+ = \{\text{Matrikelnr}\}, \\
&\text{K-SOFTSKILL}_+ = \{\text{Matrikelnr}\}\}.
\end{aligned}
$$

4. GaLVE-Schritt Der vierte Schritt des GaLVE-Verfahrens führt nun die erweiterte inverse Schemaabbildung $\mathcal{M}'_+$ aus. Wir werden hier nicht den gesamten CHASE-Baum des angepassten Disjunktiven-CHASE angeben und beschränken uns stattdessen auf die CHASE-Baum-Blätter mit den größten Unterschieden untereinander. Diese Blätter entsprechen den Instanzen, die durch die ausschließliche Anwendung einer S-T TGD pro Disjunktiven-Menge im CHASE-Baum entstehen. Die Instanz $I'_{S_K 1}$ spiegelt somit den Fall wieder, dass nur die S-T TGD mit K-NOTEN im Kopf aus ω_{3+} angewendet wird. Die Instanzen $I'_{S_K 2}$ und $I'_{S_K 3}$ ergeben sich äquivalent aus der alleinigen Anwendung der S-T TGD mit K-PRAKTIKUM bzw.

K-SOFTSKILL im Kopf. Da die Disjunktiven-Mengen ω_{1+} und ω_{2+} nur aus einer S-T TGD bestehen, werden sie auf jeder Instanz angewendet. Für die im nächsten Schritt folgende Berechnung der Sicheren Instanz sind diese Blätter des CHASE-Baumes ausreichend, da der größte mögliche Abstand aus allen möglichen S-T TGDs berechnet wurde und der Durchschnitt der Instanzen für die Sichere Instanz somit minimal wird.

Diese Art der Berechnung des angepassten Disjunktiven-CHASEs werden wir auch im folgenden Kapitel zu der Implementierung des GaLVE-Verfahrens verwenden, um den angepassten Disjunktiven-CHASE mit dem vorhandenen CHASE-Algorithmus in ChaTEAU umzusetzen. Für die im nächsten Schritt folgende Berechnung der Sicheren Instanz wurden die Nullwerte der folgenden Instanzen I'_{S_K1}, I'_{S_K2} und I'_{S_K3} bereits so angepasst, dass sie untereinander disjunkt sind:

$$
\begin{aligned}
I'_{S_K1} = \{&\text{K-STUDIERENDE}_+(1, \text{Fieber}, \text{Fabian}, \eta_{\text{Ab}_1}, \text{Informatik}),\\
&\text{K-STUDIERENDE}_+(2, \text{Sonne}, \text{Sarah}, \eta_{\text{Ab}_2}, \text{ITTI}),\\
&\text{K-STUDIERENDE}_+(2, \text{Sonnensand}, \eta_{\text{Vo}_1}, \text{Bachelor}, \eta_{\text{St}_1}),\\
&\text{K-STUDIERENDE}_+(3, \eta_{\text{Na}_1}, \text{Max}, \text{Master}, \eta_{\text{St}_2}),\\
&\text{K-NOTEN}_+(2, 3, \eta_{\text{Se}_1}, 2.0, \eta_{\text{Be}_1}),\\
&\text{K-NOTEN}_+(2, 2, \text{SS 21}, 4.0, \eta_{\text{Be}_2}),\\
&\text{K-NOTEN}_+(\eta_{\text{Mo}_1}, 2, \eta_{\text{Se}_2}, \eta_{\text{No}_1}, \eta_{\text{Be}_3})\},
\end{aligned}
$$

$$
\begin{aligned}
I'_{S_K2} = \{&\text{K-STUDIERENDE}_+(1, \text{Fieber}, \text{Fabian}, \eta_{\text{Ab}_3}, \text{Informatik}),\\
&\text{K-STUDIERENDE}_+(2, \text{Sonne}, \text{Sarah}, \eta_{\text{Ab}_4}, \text{ITTI}),\\
&\text{K-STUDIERENDE}_+(2, \text{Sonnensand}, \eta_{\text{Vo}_2}, \text{Bachelor}, \eta_{\text{St}_3}),\\
&\text{K-STUDIERENDE}_+(3, \eta_{\text{Na}_2}, \text{Max}, \text{Master}, \eta_{\text{St}_4}),\\
&\text{K-NOTEN}_+(2, 3, \eta_{\text{Se}_3}, 2.0, \eta_{\text{Be}_4}),\\
&\text{K-NOTEN}_+(2, 2, \text{SS 21}, 4.0, \eta_{\text{Be}_5}),\\
&\text{K-PRAKTIKUM}_+(1),\\
&\text{K-PRAKTIKUM}_+(2),\\
&\text{K-PRAKTIKUM}_+(3)\},
\end{aligned}
$$

$$
\begin{aligned}
I'_{S_K3} = \{&\text{K-STUDIERENDE}_+(1, \text{Fieber}, \text{Fabian}, \eta_{\text{Ab}_5}, \text{Informatik}),\\
&\text{K-STUDIERENDE}_+(2, \text{Sonne}, \text{Sarah}, \eta_{\text{Ab}_6}, \text{ITTI}),\\
&\text{K-STUDIERENDE}_+(2, \text{Sonnensand}, \eta_{\text{Vo}_3}, \text{Bachelor}, \eta_{\text{St}_5}),\\
&\text{K-STUDIERENDE}_+(3, \eta_{\text{Na}_3}, \text{Max}, \text{Master}, \eta_{\text{St}_6}),\\
&\text{K-NOTEN}_+(2, 3, \eta_{\text{Se}_4}, 2.0, \eta_{\text{Be}_6}),\\
&\text{K-NOTEN}_+(2, 2, \text{SS 21}, 4.0, \eta_{\text{Be}_7}),\\
&\text{K-SOFTSKILL}_+(1),\\
&\text{K-SOFTSKILL}_+(2),\\
&\text{K-SOFTSKILL}_+(3)\}.
\end{aligned}
$$

5. GaLVE-Schritt Die Sichere Instanz aus den Instanzen I'_{S_K1}, I'_{S_K2} und I'_{S_K3} besteht aus den neuen Tupeln und Attributen in den zusätzlichen, für das Quellschema neuen Relationen aus dem dritten Schritt. Mit ihnen können wir nun die originalen Relationen in der Quelldatenbank erweitern, wobei wir auch hier auf disjunkte Nullwerte zwischen der Sicheren Instanz und der originalen Instanz achten müssen. Die Sichere Instanz I'_{S_K} und die mit den Tupeln der Sicheren Instanz erweiterten Quellinstanz I_{S_K+} ergeben sich wie folgt:

$$
\begin{aligned}
I'_{S_K} = \{&\text{K-STUDIERENDE}_+(1, \text{Fieber}, \text{Fabian}, \eta_{\text{Ab}_1}, \text{Informatik}),\\
&\text{K-STUDIERENDE}_+(2, \text{Sonne}, \text{Sarah}, \eta_{\text{Ab}_2}, \text{ITTI}),\\
&\text{K-STUDIERENDE}_+(2, \text{Sonnensand}, \eta_{\text{Vo}_1}, \text{Bachelor}, \eta_{\text{St}_1}),\\
&\text{K-STUDIERENDE}_+(3, \eta_{\text{Na}_1}, \text{Max}, \text{Master}, \eta_{\text{St}_2}),\\
&\text{K-NOTEN}_+(2, 3, \eta_{\text{Se}_1}, 2.0, \eta_{\text{Be}_1}),\\
&\text{K-NOTEN}_+(2, 2, \text{SS } 21, 4.0, \eta_{\text{Be}_2})\},
\end{aligned}
$$

$$
\begin{aligned}
I_{S_K+} = \{&\text{K-STUDIERENDE}(2, \text{Sonnensand}, \eta_{\text{Vo}_1}, \text{Bachelor}),\\
&\text{K-STUDIERENDE}(3, \eta_{\text{Na}_1}, \text{Max}, \text{Master}),\\
&\text{K-NOTEN}(2, 3, \eta_{\text{Se}_1}, 2.0, \eta_{\text{Be}_1}),\\
&\text{K-NOTEN}(2, 2, \text{SS } 21, 4.0, \text{Knapp}),\\
&\text{K-PRAKTIKUM}(3),\\
&\text{K-SOFTSKILL}(2),\\
&\text{K-SOFTSKILL}(3),\\
&\text{K-STUDIERENDE}_+(1, \text{Fieber}, \text{Fabian}, \eta_{\text{Ab}_1}, \text{Informatik}),\\
&\text{K-STUDIERENDE}_+(2, \text{Sonne}, \text{Sarah}, \eta_{\text{Ab}_2}, \text{ITTI}),\\
&\text{K-STUDIERENDE}_+(2, \text{Sonnensand}, \eta_{\text{Vo}_2}, \text{Bachelor}, \eta_{\text{St}_1}),\\
&\text{K-STUDIERENDE}_+(3, \eta_{\text{Na}_2}, \text{Max}, \text{Master}, \eta_{\text{St}_2}),\\
&\text{K-NOTEN}_+(2, 3, \eta_{\text{Se}_2}, 2.0, \eta_{\text{Be}_2}),\\
&\text{K-NOTEN}_+(2, 2, \text{SS } 21, 4.0, \eta_{\text{Be}_3})\}.
\end{aligned}
$$

6. GaLVE-Schritt Der sechste Schritt des GaLVE-Verfahrens erstellt die TGDs und bEGDs für die Nachbereitung der Erweiterung. Diese sind wie folgt definiert, wobei wir triviale bEGDs mit leerem Kopf vernachlässigen:

$$
\begin{aligned}
\Sigma_{\text{TGD}} =& \{\sigma_{\text{TGD}_1}, \sigma_{\text{TGD}_2}, \sigma_{\text{TGD}_3}, \sigma_{\text{TGD}_4}\} \text{ mit}\\
\sigma_{\text{TGD}_1} =& \text{K-STUDIERENDE}_+(x_{\text{Ma}}, x_{\text{Na}}, x_{\text{Vo}}, x_{\text{Ab}}, x_{\text{St}})\\
&\longrightarrow \text{K-STUDIERENDE}(x_{\text{Ma}}, x_{\text{Na}}, x_{\text{Vo}}, x_{\text{Ab}}),\\
\sigma_{\text{TGD}_2} =& \text{K-NOTEN}_+(x_{\text{Mo}}, x_{\text{Ma}}, x_{\text{Se}}, x_{\text{No}}, x_{\text{Be}})\\
&\longrightarrow \text{K-NOTEN}(x_{\text{Mo}}, x_{\text{Ma}}, x_{\text{Se}}, x_{\text{No}}, x_{\text{Be}}),\\
\sigma_{\text{TGD}_3} =& \text{K-PRAKTUKUM}_+(x_{\text{Ma}}) \longrightarrow \text{K-PRAKTUKUM}(x_{\text{Ma}}),\\
\sigma_{\text{TGD}_4} =& \text{K-SOFTSKILL}_+(x_{\text{Ma}}) \longrightarrow \text{K-SOFTSKILL}(x_{\text{Ma}}),
\end{aligned}
$$

$$
\begin{aligned}
\Sigma_{\text{bEGD}} =& \{\sigma_{\text{bEGD}_1}, \sigma_{\text{bEGD}_2}\} \text{ mit} \\
\sigma_{\text{bEGD}_1} =& \text{K-STUDIERENDE}(x_{\text{Ma}_1}, x_{\text{Na}_1}, x_{\text{Vo}_1}, x_{\text{Ab}_1}) \wedge \\
& \text{K-STUDIERENDE}(x_{\text{Ma}_1}, x_{\text{Na}_2}, x_{\text{Vo}_2}, x_{\text{Ab}_2}) \\
& \longrightarrow (x_{\text{Na}_1} = x_{\text{Na}_2}) \wedge (x_{\text{Vo}_1} = x_{\text{Vo}_2}) \wedge (x_{\text{Ab}_1} = x_{\text{Ab}_2}), \\
\sigma_{\text{bEGD}_2} =& \text{K-NOTEN}(x_{\text{Mo}_1}, x_{\text{Ma}_1}, x_{\text{Se}_1}, x_{\text{No}_1}, x_{\text{Be}_1}) \wedge \\
& \text{K-NOTEN}(x_{\text{Mo}_1}, x_{\text{Ma}_1}, x_{\text{Se}_2}, x_{\text{No}_2}, x_{\text{Be}_2}) \\
& \longrightarrow (x_{\text{Se}_1} = x_{\text{Se}_2}) \wedge (x_{\text{No}_1} = x_{\text{No}_2}) \wedge (x_{\text{Be}_1} = x_{\text{Be}_2})
\end{aligned}
$$

7. GaLVE-Schritt Im letzten Schritt des GaLVE-Verfahrens führen wir die TGDs und bEGDs der Nachbereitung aus. Bei der Ausführung der TGD ergänzen wir die neu eingefügten Tupel mit der Provenance-Annotation $|_{neu}$ für ein neues Tupel. Die ursprünglich vorhandenen Tupel bekommen die Provenance-Annotation $|_{original}$ für ein originales Tupel. Nach der Ausführung der TGDs erhalten wir die Instanz:

$$
\begin{aligned}
I'_{S_K+} = \{&\text{K-STUDIERENDE}(2, \text{Sonnensand}, \eta_{\text{Vo}_1}, \text{Bachelor})|_{original}, \\
&\text{K-STUDIERENDE}(3, \eta_{\text{Na}_1}, \text{Max}, \text{Master})|_{original}, \\
&\text{K-STUDIERENDE}(1, \text{Fieber}, \text{Fabian}, \eta_{\text{Ab}_1})|_{neu}, \\
&\text{K-STUDIERENDE}(2, \text{Sonne}, \text{Sarah}, \eta_{\text{Ab}_2})|_{neu}, \\
&\text{K-STUDIERENDE}(2, \text{Sonnensand}, \eta_{\text{Vo}_2}, \text{Bachelor})|_{neu}, \\
&\text{K-STUDIERENDE}(3, \eta_{\text{Na}_2}, \text{Max}, \text{Master})|_{neu}, \\
&\text{K-NOTEN}(2, 3, \eta_{\text{Se}_1}, 2.0, \eta_{\text{Be}_1})|_{original}, \\
&\text{K-NOTEN}(2, 2, \text{SS } 21, 4.0, \text{Knapp})|_{original}, \\
&\text{K-NOTEN}(2, 3, \eta_{\text{Se}_2}, 2.0, \eta_{\text{Be}_2})|_{neu}, \\
&\text{K-NOTEN}(2, 2, \text{SS } 21, 4.0, \eta_{\text{Be}_3})|_{neu}, \\
&\text{K-PRAKTIKUM}(3)|_{original}, \\
&\text{K-SOFTSKILL}(2)|_{original}, \\
&\text{K-SOFTSKILL}(3)|_{original}, \\
&\text{K-STUDIERENDE}_+(1, \text{Fieber}, \text{Fabian}, \eta_{\text{Ab}_1}, \text{Informatik}), \\
&\text{K-STUDIERENDE}_+(2, \text{Sonne}, \text{Sarah}, \eta_{\text{Ab}_2}, \text{ITTI}), \\
&\text{K-STUDIERENDE}_+(2, \text{Sonnensand}, \eta_{\text{Vo}_2}, \text{Bachelor}, \eta_{\text{St}_1}), \\
&\text{K-STUDIERENDE}_+(3, \eta_{\text{Na}_2}, \text{Max}, \text{Master}, \eta_{\text{St}_2}), \\
&\text{K-NOTEN}_+(2, 3, \eta_{\text{Se}_1}, 2.0, \eta_{\text{Be}_2}), \\
&\text{K-NOTEN}_+(2, 2, \text{SS } 21, 4.0, \eta_{\text{Be}_3})\}.
\end{aligned}
$$

Die darauf folgende Ausführung der bEGDs ergibt das Endergebnis des GaLVE-Verfahrens. Für unser Beispiel erhalten wir die Instanz I''_{S_K+}, die im Vergleich zu ursprünglichen Instanz I_{S_K} innerhalb der ursprünglichen Relationen um Informationen aus dem globalen Schema ergänzt und um ganze neue Tupel erweitert wurde sowie in den neuen Relationen zusätzliche Attribute für die ursprünglichen Relationen enthält. In K-STUDIERENDE ist so z. B. das Tupel für den Studierenden mit der Matrikelnummer 1 hinzugefügt worden. Ebenso wurde das Tupel für die Studierende mit der Matrikelnummer 2 um einen passenden Vornamen ergänzt. Interessant ist auch, dass der Nachname der Studierenden mit der Matrikelnummer 2 immer noch der aus der Instanz I_{S_K} ist, obwohl aus der globalen Datenbank widersprüchliche Nachnamen in die Quelle integriert wurden. In der K-STUDIERENDE$_+$ Relation finden wir weitere In-

formationen zu den Studiengängen der Studierenden mit der Matrikelnummer 1 und 2. In K-NOTEN haben wir keine neuen Informationen erhalten. Das liegt daran, dass wir mit der Konstanten in der entsprechenden S-T TGD für K-NOTEN nach T-NOTEN in Σ_K eine Invertierung der S-T TGD erhalten, in der nach der Konstanten selektiert wird. Diese Selektion ergibt dann nur die Daten aus der globalen Datenbank, die auch aus I_{S_K} integriert wurden und liefert somit keine neuen Informationen für unsere lokale Instanz. Zu K-PRAKTIKUM und K-SOFTSKILL wurden ebenso keine neuen Informationen hinzugefügt. Das liegt daran, dass es keine Disjunktive-Menge gab, die für diese beiden Relationen Tupel erzeugt haben, die in der Sicheren Instanz enthalten waren.

Nach dem GaLVE-Verfahren könnten wir optional auch hier noch (wie am Ende von Abschnitt 4.3.3 beschrieben wurde) die Attribute der neuen Relationen auf die Schlüsselattribute von den jeweiligen originalen Relationen und die neuen Attribute beschränken. So könnten wir eine redundante Speicherung der nicht bereinigten originalen Attribute umgehen. Zusätzlich könnten wir auch noch eine bEGD auf die neuen Relationen anwenden, die die neuen Attribute bereinigt. Die einzigen so übrig bleibenden Tupel für die neuen Relationen wären K-STUDIERENDE$_+$(1, Informatik) und K-STUDIERENDE$_+$(2, ITTI). Diese Schritte sind allerdings optional und müssen von Anwendungsfall zu Anwendungsfall entschieden werden, sodass das Ergebnis des GaLVE-Verfahrens die Instanz I''_{S_K+} bleibt:

$$
\begin{aligned}
I''_{S_K+} = \{&\text{K-STUDIERENDE}(1, \text{Fieber}, \text{Fabian}, \eta_{\text{Ab}_1}),\\
&\text{K-STUDIERENDE}(2, \text{Sonnensand}, \text{Sarah}, \text{Bachelor}),\\
&\text{K-STUDIERENDE}(3, \eta_{\text{Na}_1}, \text{Max}, \text{Master}),\\
&\text{K-NOTEN}(2, 3, \eta_{\text{Se}_1}, 2.0, \eta_{\text{Be}_1}),\\
&\text{K-NOTEN}(2, 2, \text{SS } 21, 4.0, \text{Knapp}),\\
&\text{K-PRAKTIKUM}(3),\\
&\text{K-SOFTSKILL}(2),\\
&\text{K-SOFTSKILL}(3),\\
&\text{K-STUDIERENDE}_+(1, \text{Fieber}, \text{Fabian}, \eta_{\text{Ab}_1}, \text{Informatik}),\\
&\text{K-STUDIERENDE}_+(2, \text{Sonne}, \text{Sarah}, \eta_{\text{Ab}_2}, \text{ITTI}),\\
&\text{K-STUDIERENDE}_+(2, \text{Sonnensand}, \eta_{\text{Vo}_2}, \text{Bachelor}, \eta_{\text{St}_1}),\\
&\text{K-STUDIERENDE}_+(3, \eta_{\text{Na}_2}, \text{Max}, \text{Master}, \eta_{\text{St}_2}),\\
&\text{K-NOTEN}_+(2, 3, \eta_{\text{Se}_2}, 2.0, \eta_{\text{Be}_2}),\\
&\text{K-NOTEN}_+(2, 2, \text{SS } 21, 4.0, \eta_{\text{Be}_3})\}.
\end{aligned}
$$

Im nächsten Kapitel werden wir die Implementierung des GaLVE-Verfahrens als Erweiterung von ChaTEAU vorstellen. Im Anhang A dieser Arbeit ist dazu passend die Ausgabe des in Abschnitt 4.4.3 behandelten Beispiel von der implementierten Fassung des GaLVE-Verfahrens aufgeführt. In Anhang B sind ebenso als Ausgabe von der Implementierung des GaLVE-Verfahrens einige weitere interessante Minibeispiele aufgeführt.

5. Implementierung

In diesem Kapitel stellen wir vor, wie die in Kapitel 4 erarbeiteten Techniken des GaLVE-Verfahrens als Erweiterung von ChaTEAU implementiert wurden. Dafür werden wir uns als Erstes den zu erweiternden Datenbankforschungsprototypen ChaTEAU genauer anschauen und dessen interne Darstellung von Instanzen und Abbildungen erläutern, um die folgenden Implementierungen für das GaLVE-Verfahren besser erklären zu können.

5.1. ChaTEAU

Der Datenbankforschungsprototyp ChaTEAU (*Chase for Transforming, Evolving, and Adapting databases and queries, Universal approach*) ist ein Java-Programm, welches Integritätsbedingungen in Form von (S-T) TGDs und EGDs in Instanzen und als S-T TGDs formulierte Anfragen einchasen kann [AH19]. Mit der nativen Unterstützung von Instanzen und Anfragen in einer CHASE-Methode hebt sich ChaTEAU von anderen CHASE-Tools[1] ab, welche den CHASE jeweils nur für einen Anwendungsfall unterstützen. Die Funktionalität von ChaTEAU wurde innerhalb mehrerer studentischer Abschlussarbeiten implementiert (siehe [Jur18, Ren19, Zim20, Gö20, Ros20]). Der implementierte CHASE-Algorithmus orientiert sich sehr stark an der Theorie aus [BKM$^+$17]. Neben dem CHASE-Algorithmus sind in ChaTEAU auch einige Terminierungstests umgesetzt. So können TGDs und EGDs auf starke, schwache und normale Azyklizität geprüft werden, um noch vor der Ausführung des CHASE zu klären, ob dieser mit Sicherheit terminieren würde.

Die Grundlage für alle internen Konstrukte in ChaTEAU bilden die *Terme*, mit denen Konstanten, Variablen und Nullwerte gespeichert werden können. Ein Term ist dabei immer eine Konstante, Variable oder Null. Konstanten können vom Typ Integer, Double oder String sein. *Atome* bilden die nächsthöhere Abstraktionsebene. Dabei wird zwischen *relationalen* und *Gleichheitsatomen* unterschieden. Ein relationales Atom kann eine beliebige Menge von Termen beinhalten, während Gleichheitsatom nur zwei Terme beinhaltet, zwischen denen eine Gleichheitsrelation gelten soll.

Für das GaLVE-Verfahren bedeutet das, dass alle Attribut- oder Nullwerte und alle Variablen durch Terme dargestellt und in Atomen gespeichert werden. Ein relationales Atom spiegelt demnach eine Abstraktion eines Tupels oder eines relationalen Ausdrucks in einer Integritätsbedingung wieder. Eine Instanz besteht somit aus einer Menge von relationalen Atomen. Die Beziehung von einer Instanz zu den Termen ist beispielhaft in Abbildung 5.1 dargestellt. Der Aufbau einer Instanz ist auch in Listing 5.1 in den Zeilen 43-57 zu erkennen.

[1] Das CHASE-Tool *Graal* [BLM$^+$15, Gra21] bildet eine Ausnahme mit der teilweisen Unterstützung von Instanzen und Anfragen innerhalb verschiedener Anwendungsfälle.

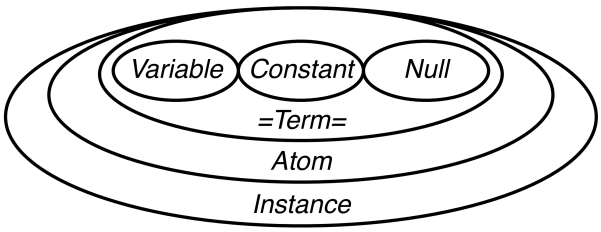

Abbildung 5.1.: Grober Aufbau einer Instanz in ChaTEAU

Die Integritätsbedingungen (TGDs, EGDs und S-T TGDs) werden in ChaTEAU genau wie in der Theorie in einen Rumpf und einen Kopf unterteilt. Der Rumpf einer Integritätsbedingung besteht dabei immer aus einer Menge von relationalen Atomen, die jeweils für einen relationalen Ausdruck stehen. Der Kopf einer (S-T) TGD besteht ebenfalls aus relationalen Atomen. Der Kopf einer EGD wird aus einer Menge der angesprochenen Gleichheitsatomen gebildet. Da Anfragen als S-T TGDs dargestellt werden, bestehen sie ebenfalls nur aus relationalen Atomen im Rumpf und im Kopf. Der Aufbau einer solchen S-T TGD ist auch in Listing 5.1 in den Zeilen 21-39 zu erkennen.

Die während des CHASE-Algorithmus bei der Suche nach Triggern oder dem Anwenden von EGDs aufgebauten Abbildungen zwischen zwei Termen werden in sogenannten *Termabbildungen* dargestellt. Die Menge von Termabbildungen zwischen den Instanzen bzw. Anfragen und den Integritätsbedingungen sowie die Abbildungen innerhalb einer Instanz oder Anfrage werden in ChaTEAU in einem *Homomorphismus* gespeichert. Homomorphismen sind somit ausschlaggebend für die Suche nach aktiven Triggern und deren folgende Anwendung. Beispiele für Homomorphismen wurden für die Definitionen 2.9 und 2.14 sowie im Abschnitt 2.4.2 genügend gegeben.

Für die Ausführung von ChaTEAU stehen ein Konsolen-Modus und eine GUI zur Verfügung. Als Eingabe werden Dateien im XML-Format genutzt. Eine beispielhafte XML-Eingabedatei ist in Listing 5.1 aufgeführt. Die Datei entspricht einem Ausschnitt des in Abschnitt 4.4.3 und Anhang A verwendeten Beispiels für die Kieler Uni-Datenbank. Die wichtigsten Strukturen sind das `schema` mit den `relations` und `dependencies` sowie die `instance`. Die `relations` bestimmen das Schema über dem die Integritätsbedingungen und die Instanz definiert werden müssen. Jede Relation wird dabei mit ihrem Namen und den zugehörigen Attributen definiert. Der optionale `tag` einer Relation gibt an, ob es sich bei einer Datenintegration um eine Relation des Quellschemas (`S`) oder des Zielschemas (`T`) handelt. In den `dependencies` werden alle Integritätsbedingungen hinterlegt. Es ist dabei egal, ob es sich um lokale Integritätsbedingungen auf dem Quell- bzw. Zielschema oder einer S-T TGD zur Datenintegration vom Quell- zum Zielschema handelt. Die `instance` speichert alle Tupel aus allen Relationen. Die Zugehörigkeit eines Tupels zu einer Relation wird über dem Namen des jeweiligen Atoms bestimmt.

Listing 5.1: Beispiel ChaTEAU-Eingabe

```
1  <input>
2      <schema>
3          <relations>
4              <relation name="K_STUDIERENDE" tag="S">
5                  <attribute name="Matrikelnr" type="int" key="true"/>
6                  <attribute name="Nachname" type="string"/>
7                  <attribute name="Vorname" type="string"/>
8                  <attribute name="Abschluss" type="string"/>
9              </relation>
10             ...
11             <relation name="T_STUDIERENDE" tag="T">
12                 <attribute name="Matrikelnr" type="int"/>
13                 <attribute name="Nachname" type="string"/>
14                 <attribute name="Vorname" type="string"/>
```

```
15                  <attribute name="Studiengang" type="string"/>
16                  <attribute name="Abschluss" type="string"/>
17              </relation>
18          <relations/>
19          ...
20          <dependencies>
21              <sttgd>
22                  <body>
23                      <atom name="K_STUDIERENDE">
24                          <variable name="Matrikelnr" type="V" index="1" />
25                          <variable name="Nachname" type="V" index="1" />
26                          <variable name="Vorname" type="V" index="1" />
27                          <variable name="Abschluss" type="V" index="1" />
28                      </atom>
29                  </body>
30                  <head>
31                      <atom name="T_STUDIERENDE">
32                          <variable name="Matrikelnr" type="V" index="1" />
33                          <variable name="Nachname" type="V" index="1" />
34                          <variable name="Vorname" type="V" index="1" />
35                          <variable name="Studiengang" type="E" index="1" />
36                          <variable name="Abschluss" type="V" index="1" />
37                      </atom>
38                  </head>
39              </sttgd>
40              ...
41          </dependencies>
42      </schema>
43      <instance>
44          <atom name="K_STUDIERENDE">
45              <constant value="2" />
46              <constant value="Sonnensand" />
47              <null name="firstname" index="1" />
48              <constant value="Bachelor" />
49          </atom>
50          <atom name="K_STUDIERENDE">
51              <constant value="3" />
52              <null name="lastname" index="1" />
53              <constant value="Max" />
54              <constant value="Master" />
55          </atom>
56          ...
57      </instance>
58  </input>
```

Die Abbildung 5.2 zeigt die Ausgabe von der ChaTEAU-GUI zu der eben angegebenen Eingabe-Datei in Listing 5.1. Im oberen Feld werden die Tupel der Eingabe-Instanz angegeben und das mittlere Feld zeigt die auf diese Instanz angewendeten Eingabe-Integritätsbedingungen. Das Ergebnis des CHASE-Algorithmus ist in unserem Fall die Ergebnis-Instanz im unteren Feld.

Anhand der Tab-Darstellung des "CHASE"-Tabs in Abbildung 5.2 ist zuerkennen, dass die Terminierungstests ("Tests"-Tab) vor der Ausführung des CHASE-Algorithmus durchgeführt werden können. Die Eingabe-Datei kann im "Start"-Tab eingelesen werden. Im "Log"-Tab können die innerhalb des CHASE-Algorithmus ausgeführten Schritte detailliert nachgelesen werden. ChaTEAU bietet zudem die Möglichkeit, das Ergebnis des CHASE-Algorithmus wieder als XML-Datei im Eingabe-Format zu speichern, um sie für eine weitere Verwendung in ChaTEAU nutzen zu können.

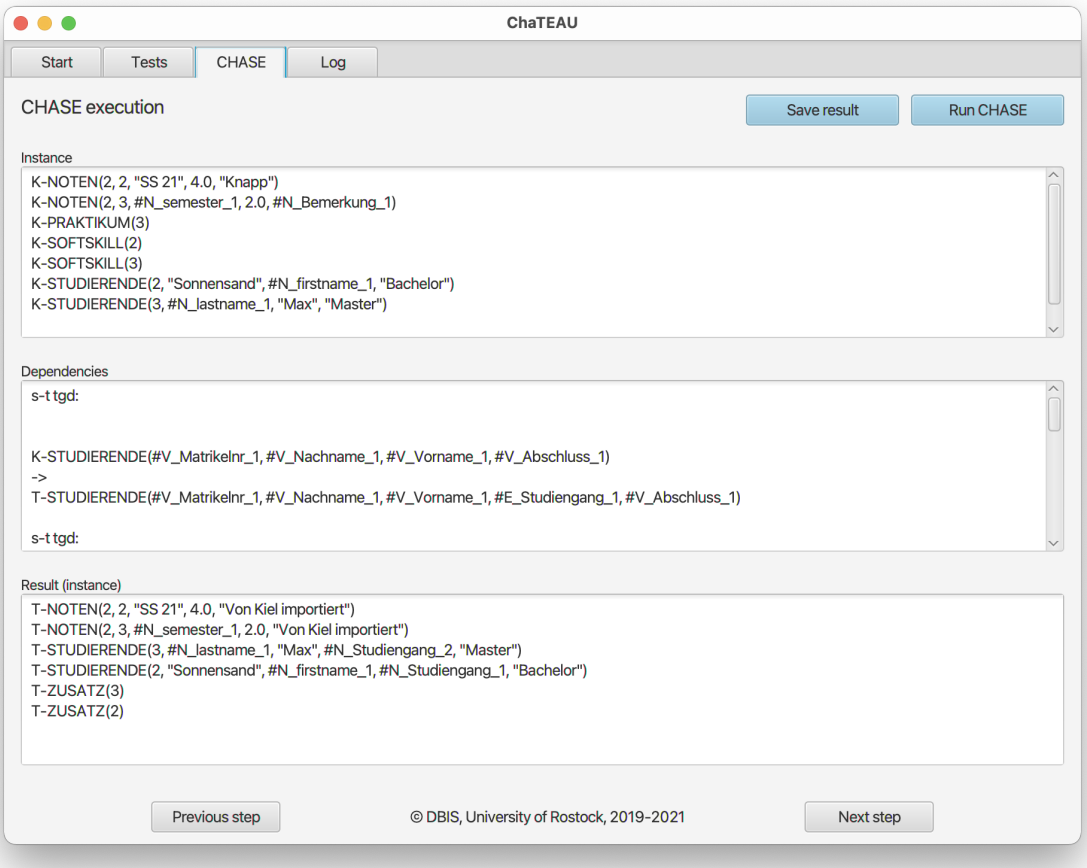

Abbildung 5.2.: ChaTEAU Beispielausgabe

Im nächsten Abschnitt werden wir die Implementierung der Techniken des GaLVE-Verfahrens als Erweiterung von ChaTEAU vorstellen. Das gezeigte XML-Eingabeformat werden wir genau so beibehalten, wie den internen Aufbau einer Instanz oder der Integritätsbedingungen.

5.2. Konzept-Implementierung

In diesem Abschnitt werden wir die Implementierung der einzelnen Techniken für das GaLVE-Verfahren vorstellen. Da die Funktionalität der einzelnen Methoden jeweils den Algorithmen und Definitionen aus dem 4. Kapitel entspricht, werden wir im Folgenden nur die wichtigen Entwurfselemente der implementierten Methoden vorstellen und die Algorithmen nicht erneut Schritt für Schritt durchgehen. Für die Darstellung der Methode werden wir eine abstrakte Schreibweise verwenden, bei der durch die "..."-Nutzung manche Datentypen bis hin zu ganzen Programmcodepassagen ersetzt wurden. Die aussagekräftigen Kommentare und Variablennamen des Programmcodes führen dazu, dass man die Struktur und den Ablauf der Methoden auch in dieser verkürzten Fassung gut erkennen kann. Der gesamte Programmcode der folgenden Techniken ist in Anhang C in der `GalveTechniques`- und zum Teil in der `Chase`-Klasse aufgeführt. Dort ist auch der Programmcode für alle in den GaLVE-Techniken genutzten Hilfsmethoden der `GalveUtil`-Klasse sowie die gesamte `GalveMain`-Klasse gegeben. Letztere enthält eine `main`-Methode, mit der ein kompletter Durchlauf des GaLVE-Verfahrens gestartet werden kann, indem die im Folgenden vorgestellten Techniken des GaLVE-Verfahrens in der richtigen Reihenfolge aufgerufen werden. Das Ergebnis eines solchen gesamten Durchlaufes des implementierten GaLVE-Verfahrens ist für unser Beispiel aus Abschnitt 4.4.3 im Anhang A zu finden. In Anhang B sind noch weitere interessante Minibeispiele für das implementiere GaLVE-Verfahren aufgelistet. Diese testen einige weitere Beispiele dieser Arbeit, prüfen speziell konstruierte Fälle der Optimierung und der Berechnung der Sicheren Instanz und durchlaufen einfache Fälle für Schemaabbildungen mit Dekomposition, Projektion und Selektion.

Vorbereitung Für die Eingabe des GaLVE-Verfahrens lesen wir die einzelnen Schemaabbildungen für die Datenintegration vor dem GaLVE-Verfahren als einzelne Eingabe-Dateien ein, so wie sie standardmäßig für ChaTEAU definiert werden würden. Diese Eingabe-Dateien dürfen ausschließlich S-T TGDs als Integritätsbedingungen besitzen, mit denen sie eine Instanz unter einem lokalen Schema in ein globales Schema abbilden. Da wir das globale Schema der durch die Datenintegration aufgebauten globalen Datenbank aus den einzelnen globalen Schemata der Schemaabbildungen zusammensetzten werden, sollten die globalen Schemata der einzelnen Schemaabbildungen immer gleich sein und dürfen sich nicht widersprechen (sprich z. B. für eine Relation des globalen Schemas unterschiedliche Attribute fordern). Das eigentliche GaLVE-Verfahren starten wir dann einfach mit der ersten eingelesenen Eingabe-Datei. Sie bestimmt das lokale Schema und die Instanz, die mit dem GaLVE-Verfahren erweitert werden soll und gibt auch die S-T TGDs der Schemaabbildung vor, aus denen die inverse Schemaabbildung berechnet wird. Diese erste Eingabe-Datei muss für die Erstellung der bEGD im 6. Schritt des GaLVE-Verfahrens Schlüsselattribute im lokalen Schema definieren, welche neu für ChaTEAU sind.

Zu der nötigen Erweiterung von ChaTEAU um eine Unterstützung für Schlüsselattribute kamen noch weitere kleine Anpassung dazu, die vor der Implementierung des GaLVE-Verfahrens angepasst oder hinzugefügt werden mussten. So mussten z. B. zu den Termabbildungen weitere Attribute für die Terme der Abbildungsquelle und dem Abbildungsziel hinzugefügt werden, die die Atome und die Position des jeweiligen abgebildeten Terms in den Atomen speichern, für welche die Termabbildungen ursprünglich erzeugt wurden. Diese Atome in den neuen Attributen der Termabbildungen benötigen wir bei der Anwendung von bEGDs, um auf bestehende Abbildungen auf Konstanten aus originalen oder neuen Atomen in einer Instanz zu testen. Die Unterscheidung zwischen originalen und neuen Atomen musste ebenfalls hinzugefügt werden und so auch bestehende Methoden in ChaTEAU angepasst werden, damit sie diese Information in den Atomen während des CHASEs nicht überschreiben. Dazu kamen noch einige weitere kleine Anpassungen, die für diese Arbeit keine wichtige Rolle spielen, aber notwendig für die Umsetzung des GaLVE-Verfahrens als Erweiterung von ChaTEAU waren.

0. GaLVE-Schritt Die Datenintegration vor dem eigentlichen GaLVE-Verfahren muss die einzelnen Schemaabbildungen aus den Eingabe-Dateien ausführen und dann die Ergebnisse vereinigen. Die implementierte Methode ist in Listing 5.2 dargestellt. Da wir uns dazu entscheiden haben, das Zielschema nicht direkt anzugeben, wird es in Zeile 7 aus den Schemata der Eingaben zusammengesetzt, die mit einem `tag` für ein Zielschema markiert sind (siehe `tag="T"` in Listing 5.1). Dabei gehen wir davon aus, dass keine widersprüchlichen Schemata in den einzelnen Eingabe-Dateien angegeben sind, da diese sich sonst gegenseitig überschreiben würden. Für die Ausführung der einzelnen Schemaabbildungen reicht es aus, den CHASE-Algorithmus jeweils für die Instanz und die Integritätsbedingungen (S-T TGDs) der separaten Eingabe-Dateien auszuführen (siehe Zeile 7). In Zeile 8 sammeln wir die einzelnen Ergebnisinstanzen des CHASE-Algorithmus für die jeweiligen Schemaabbildungen auf und versehen sie in Zeile 12 mit untereinander disjunkten Nullwerten. Dieser Schritt ist wichtig, da wir in den einzelnen Ergebnisinstanzen des CHASE-Algorithmus jedes Mal von vorne Indices für neue Nullwerte vergeben und bei einer einfachen Vereinigung dieser Ergebnisinstanzen mit möglicherweise gleich benannten Nullwerte, Gleichheiten implizieren, die es nicht gibt. Die disjunkten Nullwerte in einer Menge von Instanzen werden in der `GalveUtil`-Methode `makeDisjunctNulls` umgesetzt. Diese ist wie alle Hilfsmethoden im Anhang C zu finden. Unsere Ergebnisinstanz der gesamten Datenintegration und somit die globale Datenbank für die weiteren GaLVE-Schritte entsteht letztendlich in Zeile 13 durch die Vereinigung der einzelnen Ergebnisinstanzen des CHASE-Algorithmus mit den disjunkten Nullwerten.

Listing 5.2: Datenintegration vor dem GaLVE-Verfahren

```
1  static ... dataIntegration(... inputs) {
2  var globalInstances = new ...;
3  var globalInstance = new ...;
4  for (... input : inputs) {
5      ...
6      // fill global scheme
7      globalInstance.getSchema().putAll(...getSchemaForSchemaTag(...TARGET));
8      var resultInstance = GalveUtil.runChase(input);
9      globalInstances.add(resultInstance);
10 }
11 // make disjunct nulls
12 GalveUtil.makeDisjunctNulls(globalInstances);
13 globalInstances.forEach(instance -> globalInstance...addAll(instance...));
14 return globalInstance;
15 }
```

1. GaLVE-Schritt Für die prototypische Umsetzung des GaLVE-Verfahrens haben wir uns dazu entschieden, immer das lokale Schema und die Instanz der ersten Eingabe-Datei mit den Informationen aus der durch die Schemaabbildungen aller Eingabe-Dateien erzeugten globalen Datenbank zu erweitern. So startet unser GaLVE-Verfahren mit der in Listing 5.3 aufgeführten Invertierung der Integritätsbedingungen (S-T TGDs) der ersten Eingabe-Datei. Die Methode berechnet die Adapted Strong Maximum Extended Recovery nach dem Vorbild des Algorithmus 4. Tatsächlich ist dieser Algorithmus während der Implementierung von Algorithmus 3 entstanden, um den Umweg über die disjunktiven TGDs mit den internen Homomorphismen zu vermeiden.

In den Zeilen 4-9, innerhalb der ersten for-Schleife werden die Köpfe der S-T TGDs normalisiert. In den Zeilen 12-20 werden diese normalisierten S-T TGDs invertiert und dann in Disjunktiven-Mengen sortiert. Dabei wird für jedes neue Rumpf-Relationenschema (Relation des ursprünglichen Kopfes, siehe Zeile 13) eine neue Disjunktive-Menge angelegt. Die eigentliche Invertierung einer S-T TGD sowie die Anpassung der All- und Existenzquantoren der Variablen in den Rümpfen und Köpfen der S-T TGDs wird in der `GalveUtil`-Methode `invert` umgesetzt (siehe Anhang C). Das Ergebnis der Methode bildet

eine inverse Schemaabbildung mit den eben erzeugten Disjunktiven-Mengen, die nun von dem Zielschema der Eingabe-Schemaabbildung in dessen Quellschema abbilden.

Listing 5.3: Invertierung von Schemaabbildungen

```
static ... invert(... schemaMapping) {
// normalize sttgds
var normalizedSttgds = new ...;
for(... constraint : schemaMapping...) {
    for(... headAtom : constraint.getHead()) {
        var newHead = new ...;
        newHead.add((...) headAtom.copy());
        normalizedSttgds.add(new STTgd(((...)constraint).copyBody(),newHead));
}}
// invert the sttgds and collect them by relations in the bodies in disjunctive sets
var disjunctiveSets = new HashMap<String,LinkedHashSet<...>>();
for(... constraint : normalizedSttgds) {
    var headRelationName = ((...)constraint.getHead()...getName();
    if(disjunctiveSets.containsKey(headRelationName)) {
        disjunctiveSets.get(headRelationName).add(GalveUtil.invert((...)constraint));
    }else {
        var sttgds = new ...;
        sttgds.add(GalveUtil.invert((...)constraint));
        disjunctiveSets.put(headRelationName, sttgds);
}}
return new InverseSchemaMapping(schemaMapping.getTargetSchema(),
    schemaMapping.getSourceSchema(), disjunctiveSets);
}
```

2. GaLVE-Schritt Die in Listing 5.4 angegebene Methode zur Optimierung von Disjunktiven-Mengen für den angepassten Disjunktiven-CHASE in Verbindung mit der folgenden Berechnung der Sicheren Instanz orientiert sich an der Definition 4.8. Die Methode löscht aus einer Menge von Disjunktiven-Mengen alle Disjunktiven-Mengen, die bei ihrer Anwendung im angepassten Disjunktiven-CHASE keine Tupel für die Sichere Instanz beitragen können, da sie den CHASE-Baum im angepassten Disjunktiven-CHASE so aufspalten, dass die neuen Zweige keine Überlappungen mit einander oder mit Tupeln aus anderen Disjunktiven-Mengen haben. Dafür wird in Zeile 4 für jede Disjunktive-Menge mit der `GalveUtil`-Methode `headRelationsIntersection` (siehe Anhang C) der Durchschnitt der Relationen in den Köpfen der S-T TGDs der Disjunktiven-Menge gebildet und dann geprüft, ob dieser leer ist. Ist das der Fall wird in Zeile 5 mit der `GalveUtil`-Methode `existsDisjunktiveSetWithRelationInIntersection` (siehe Anhang C) geprüft, ob es in der gesamten Menge der Disjunktiven-Mengen eine andere Disjunktive-Menge gibt, deren Durchschnitt der Relationen in den Köpfen der S-T TGDs auch eine Relation enthält, die in irgendeinem Kopf der untersuchten Disjunktiven-Menge vorkommt. Sollte das nicht der Fall sein, sammeln wir die Disjunktive-Menge in Zeile 7 auf, um sie in Zeile 9 aus der Menge der Disjunktiven-Mengen entfernen zu können.

Listing 5.4: Optimierung von Disjunktiven-Mengen

```
static void optimizeDisjunctiveSets(... disjunctiveSets) {
var toRemove = new ...;
for(... disjunctiveSet : disjunctiveSets...) {
    if(GalveUtil.headRelationsIntersection(disjunctiveSet...).isEmpty()) {
        if(!GalveUtil.existsDisjunktiveSetWithRelationInIntersection(
            disjunctiveSet..., disjunctiveSets)) {
            toRemove.add(disjunctiveSet...);
}}}
toRemove.forEach(relationName -> disjunctiveSets.remove(relationName));
}
```

3. GaLVE-Schritt Die Methode in Listing 5.5 setzt die Erweiterung einer Schemaabbildung nach Algorithmus 6 um. Von Zeile 2 bis Zeile 16 werden die Integritätsbedingungen in einer inversen Schemaabbildung so verändert, dass sie auf neue Relationenschemata abbilden. Dafür werden die Relationennamen in den Köpfen der bestehenden Schemaabbildungen einfach um ein + erweitert. Die `GalveUtil`-Methode `generateNewName` (siehe Anhang C) sorgt in Zeile 8 dafür, dass wirklich neue Relationennamen für ein Datenbankschema entstehen und auch zufälligerweise bestehende Relationennamen mit einem + am Ende beachtet werden. Der restliche Teil der Methode von Zeile 17 bis Zeile 45 kümmert sich um die Erweiterung der neuen Schemata und der Integritätsbedingungen mit zusätzlichen interessanten Attributen aus dem globalen Schema. Aufgrund der Länge des Programmcodes ist dies hier sehr verkürzt angegeben. Bei weiterem Interesse an der Umsetzung der Erweiterung von Schemaabbildungen verweisen wir auf die detaillierte Implementierung im Anhang C.

Listing 5.5: Erweiterung einer Schemaabbildung

```
1  static void extend(... inverse) {
2  // generate disjunctive sets that map to new relations and extend scheme of the source
3  var newDisjunctiveSets = new HashMap<String,LinkedHashSet<...>>();
4  // copy the schema to not disturb the new name generation when the schema is extended
5  var originalTargetSchema = GalveUtil.copySchema(inverse.getTargetSchema());
6  for(... disjunctiveSet : inverse.getDisjunctiveSets()...) {
7      ...
8          var newRelationName = GalveUtil.generateNewName(
9              oldRelationName, originalTargetSchema);
10         newSttgd.getHead().add(new RelationalAtom(newRelationName, atom.getTerms()));
11     ...
12     }
13     newDisjunctiveSets.put(disjunctiveSet.getKey(), newDisjunctiveSet);
14 }
15 inverse.getDisjunctiveSets().clear();
16 inverse.getDisjunctiveSets().putAll(newDisjunctiveSets);
17 // collect all attributes from the relations of the body atoms of the disjunctive sets
18 // search for interesting attributes = attributes that appear in a body but not in a head
19 var allAttibutes, interestingAttibutes = ...;
20 // expand mappings of the disjunctive sets and
21 // the schema of the source with corresponding interesting attributes
22 for(... attribute : interestingAttibutes) {
23     for(... disjunctiveSet : inverse...) {
24         // expand only the disjunctive sets with the corresponding attribute in the body
25         if(inverse.getSourceSchema().get(disjunctiveSet...).containsKey(attribute)) {
26             ...
27             for(... sttgd : disjunctiveSet...) {
28                 // it is enough to find the first term,
29                 // the terms are the same by construction for each sttgd
30                 var bodyAtom = sttgd.getBody()...;
31                 var bodyTerm = bodyAtom...get(termPosition);
32                 for(... atom : sttgd.getHead()) {
33                     // attribute is new for the relation of the atom
34                     if(!inverse.getTargetSchema().get(atom...).containsKey(attribute)) {
35                         // add interesting attribute
36                         atom...add(bodyTerm);
37                         // change all corresponding atoms in the heads
38                         // of the sttgds in the disjunctive sets and adjust the schema
39                         ...
40                     }else {
41                         // attribute already exists in the relation of the atom
42                         // replace the existing variables in the sttgd head
43                         // with variables/constants from the body
44                         ...
45 });}}}}}}}
```

4. GaLVE-Schritt Im Abschnitt 4.4.3 haben wir bereits erwähnt, dass es beim angepassten Disjunktiven-CHASE ausreicht die Teile des CHASE-Baumes mit den maximalen Abständen zueinander zu berechnen, wenn man weiß, dass aus dem Ergebnis eine Sichere Instanz berechnet werden soll. Die Blätter mit den größten Unterschieden untereinander entstehen durch die ausschließliche Anwendung einer S-T TGD pro Disjunktiver-Menge in einem Zweig des CHASE-Baumes. Dieser *verfeinerte angepasste Disjunktive-CHASE* hat mehrere Vorteile: So müssen zum einen nicht so viele Kombinationen von S-T TGDs im CHASE-Baum verfolgt werden, wodurch dieser kleiner und handhabbarer wird. Ein kleinerer CHASE-Baum reduziert auch die Laufzeit der Berechnung der Sicheren Instanz, bei der nach Abbildungen für jedes Tupel in jeder Instanz gesucht wird. Da die Instanzen im Ergebnis des verfeinerten angepassten Disjunktiven-CHASE jeweils aus genau einer S-T TGD pro Disjunktiver-Menge entstanden sind, bilden sie so dieselbe Sichere Instanz wie die Instanzen des gesamten CHASE-Baumes. Dadurch, dass wir jede Disjunktive-Menge anwenden, ist die Sichere Instanz genau so groß (enthält mindestens so viele Tupel), da wir nur eine S-T TGD pro Disjunktiver-Menge anwenden, ist die Sichere Instanz genau so klein (enthält maximal so viele Tupel) wie die Sichere Instanz des gesamten CHASE-Baumes. Ein praktischer Vorteil für die Implementierung ist, dass wir den gesamten in ChaTEAU implementierten CHASE-Algorithmus rekursiv nutzen können, indem wir ihn jeweils für eine S-T TGD und eine Instanz ausführen können.

Würden wir diese Verfeinerung der Berechnung nicht umsetzen, müsste man den ChaTEAU-CHASE nach jeder Ausführung eines aktiven Triggers abbrechen und mit dem Zwischenergebnis die nächste Rekursion starten bzw. einen eigenen Disjunktiven-CHASE von Grund auf neu entwickeln. Für weitere Arbeiten bleibt so anzumerken, dass unsere implementierte Methode für den verfeinerten angepassten Disjunktiven-CHASE (siehe Listing 5.6 und 5.7) nicht den kompletten CHASE-Baum aufspannt und somit nicht die Definition 4.4 für den angepassten Disjunktiven-CHASE umsetzt. Für andere Anwendungsfälle, die auf dem Ergebnis des angepassten Disjunktiven-CHASE nicht die Sichere Instanz berechnen, müsste man daher untersuchen, ob die Vereinfachung des angepassten Disjunktiven-CHASEs ebenfalls keine Folgen für das Endergebnis hat.

Die Methode in Listing 5.6 startet die Ausführung des verfeinerten angepassten Disjunktiven-CHASEs. Dabei ist die Methode erst mal nur dafür zuständig, in Zeile 4 die rekursive Methode in Listing 5.7 aufzurufen und damit die eigentliche Ausführung des angepassten Disjunktiven-CHASEs zu beginnen. Diese wählt als Erstes eine Disjunktive-Menge aus der inversen Schemaabbildung aus (siehe Zeile 10) und fängt dann in den Zeilen 13-19 damit an den ChaTEAU-CHASE für die Eingabeinstanz und den einzelnen S-T TGDs der ausgewählten Disjunktive-Menge getrennt auszuführen. Interessant ist, dass aus den originalen S-T TGDs der Disjunktiven-Menge in Zeile 16 neue TGDs erzeugt werden, mit denen der ChaTEAU-CHASE letztendlich ausgeführt wird. Das hat den Grund, dass die Ergebnisinstanz im ChaTEAU-CHASE nach der Anwendung einer S-T TGD auf die Tupel aus dem Zielschema eingeschränkt wird. So würden die Tupel des Quellschemas nicht mehr für weitere Rekursionsaufrufe zur Verfügung stehen. Die nächste Ausführung des ChaTEAU-CHASEs für die S-T TGDs einer weiteren Disjunktiven-Mengen könnte so für die Ergebnisinstanz der vorherigen Ausführung und der aktuellen S-T TGD keine Triggern mehr finden und auch keine neuen Tupel hinzufügen.

Die Ergebnisinstanzen werden nach der Ausführung des CHASEs für die Eingabeinstanz und den einzelnen S-T TGDs in Zeile 18 direkt in einer Menge gesammelt. Der rekursive Aufruf der Methode erfolgt nun in den Zeilen 26-29 für jede einzelne zuvor berechnete Ergebnisinstanz des ChaTEAU-CHASEs und der Menge der Disjunktiven-Mengen, aus der die eben abgearbeitete Disjunktive-Menge entfernt wurde. Die Rekursion endet, wenn wir alle Disjunktiven-Mengen angewendet und sie darauf hin aus der Menge der Disjunktiven-Mengen entfernt haben. Diese Abbruchbedingung wird in den Zeilen 5-8 für die Disjunktiven-Mengen der Eingabe überprüft. Endet die Rekursion, geben wir die Eingabeinstanz als Ergebnisinstanz zurück. Beim rekursiven Abbruch der weiteren Methoden werden alle Ergebnisinstanzen

der rekursiven Aufrufe in Zeile 27 zusammengesammelt und dann letztendlich an die die Methode in Listing 5.6 zurückgegeben.

Die Methode in Listing 5.6, die die rekursive Methode in Listing 5.7 aufgerufen hat, muss nun nur noch die Tupel aus dem Quellschema der inversen Schemaabbildung, sprich die Tupel der globalen Datenbank, mit denen wir die rekursive Berechnung gestartet haben, aus den einzelnen Ergebnisinstanzen löschen. Dabei können wir auch gleich das Schema der Instanzen auf das Zielschema der inversen Schemaabbildung, sprich das Quellschema einschränken (siehe Zeile 6-10). Die Ergebnisinstanzen des verfeinerten angepassten Disjunktiven-CHASE bestehen nur noch aus den durch die S-T TGDs neu hinzugefügten Tupeln.

Listing 5.6: Verfeinertet angepasster Disjunktiver-CHASE

```
1  static ... runAdaptedDisjunvtiveChase(... targetInstance, ... inverse){
2  var resultInstances = new ...;
3  // start recursive executions of the chase
4  resultInstances = runAdaptedDisjunctiveChaseRec(targetInstance, inverse);
5  // remove the atoms of the global database and their relations from the schema
6  for(... instance : resultInstances) {
7      ...
8      instance...removeAll(...);
9      instance.getSchema().remove(...);
10 }
11 return resultInstances;
12 }
```

Listing 5.7: Verfeinertet angepasster Disjunktiver-CHASE - Rekursion

```
1  static ... runAdaptedDisjunctiveChaseRec(... instance, ... inverse){
2  var chaseResultInstances = new ...;
3  var resultInstances = new ...;
4  // termination condition of the recursion
5  if(inverse.getDisjunctiveSets().isEmpty()) {
6      resultInstances.add(instance);
7      return resultInstances;
8  }
9  // select the first disjunctive set and start the chase for each sttgd individually
10 var disjunctiveSet = inverse.getDisjunctiveSets()...get();
11 var disjunctiveSetRelation = disjunctiveSet.getKey();
12 var sttgds = disjunctiveSet.getValue();
13 for (... sttgd : sttgds) {
14     var constraints = new ...;
15     // tgd instead sttgd to preserve the source tuples in chateau chase
16     constraints.add(new Tgd(sttgd.getBody(), sttgd.getHead()));
17     // run chase and add chase result
18     chaseResultInstances.add(...runChase(...(instance, constraints)));
19 }
20 // copy the disjunctive sets to not disturb the recursion
21 var disjunctiveSetsCopy = GalveUtil.copyDisjunktiveSets(inverse...);
22 // remove the used disjunctive set
23 disjunctiveSetsCopy.remove(disjunctiveSetRelation);
24 var inverseCopy = new InverseSchemaMapping(inverse...);
25 // recursion for each instance created by each sttgds
26 for(... chaseResultInstance : chaseResultInstances) {
27     resultInstances.addAll(runAdaptedDisjunctiveChaseRec(
28         chaseResultInstance, inverseCopy));
29 }
30 return resultInstances;
31 }
```

5. GaLVE-Schritt Die Implementierung des Algorithmus 5 zur Berechnung der Sicheren Instanz ist in Listing 5.8 angegeben. Das erneute Starten der loop-1 aus dem Algorithmus 5, für den Fall, dass widersprüchlich abgebildete Nullwerte gefunden wurden, ist mit einer `do-while`-Schleife mit Beginn in Zeile 8 und Ende in Zeile 71 umgesetzt. So kann man die in Zeile 7 definierte `breakWhile`-Variable auf false setzen und loop-1 abbrechen, um loop-1 neu zu starten, wie es z. B. in den Zeilen 35 und 40 getan wird. Die logischen Oder-Ausdrücke in den if-Statements des Algorithmus 5, mit denen wir dargestellt haben, dass Homomorphismen von einem zum anderen Tupel oder umgekehrt gelten können, wurden mit einzelnen `if`-Statements in den Zeilen 28 und 43, 45 und 52 sowie 54 und 57 umgesetzt.

Die implementierte Methode setzt ansonsten genau den Algorithmus 5 um, wobei wieder einige Hilfsmethoden aus `GalveUtil` genutzt wurden. Die Methode `makeDisjunctNulls` haben wir bereits bei der Datenintegration kennengelernt. Sie sorgt auch hier dafür, dass die übergebenen Instanzen aus dem angepassten Disjunktiven-Schritt disjunkte Nullwerte haben, was eine Bedingung bei der Berechnung der Sicheren Instanz ist. Mit der Methode `applyTermMappings` wenden wir die einzelnen Termabbildungen solange auf ein Atom an, bis sich dieses nicht mehr ändert. Rufen wir `applyTermMappings` mit einer Instanz auf, werden die Termabbildungen so auf jedes Atom der Instanz angewendet. Mit der erneuten Anwendung der Termabbildungen, falls sich die Atome nach einem Durchlauf ändern, können wir sichergehen, dass auch Verkettungen von Termabbildungen angewandt werden. Die Methode `hasConflictingNulls` berechnet zu erst alle Nullwerte, die von einem Homomorphismus auf unterschiedliche Konstanten abgebildet werden und sucht dann in einem Atom nach diesen in Konflikt stehenden Nullwerten. Ebenfalls interessant ist die Methode `homomorphismIsExpandable`. Sie überprüft für einen existierenden Homomorphismus, ob dieser um Termabbildungen für zwei Atome erweitert werden kann. Die Methode `addTermMapping` fügt einfach eine neue Termabbildung für zwei Atome zu einem Homomorphismus hinzu, falls diese aufeinander abgebildet werden können. Alle eben genannten Methoden sind in Anhang C aufgeführt.

Listing 5.8: Berechnung Sichere Instanz

```
1  static ... safeInstance(... instances) {
2  var resultInstance = new ...;
3  // homomorphism mapping the atoms of the instances to the atoms in the result
4  // homomorphism collects the mappings over all computations
5  var homomorphism = new ...;
6  // the flag to not restart the calculation
7  var breakWhile = true;
8  do {
9      // set the flag to not restart the calculation
10     breakWhile = true;
11     // stores an atom with a conflicting null value
12     var toRemove = new ...;
13     // instance with an atom with conflicting null value
14     var instanceForRemove = new ...;
15     resultInstance = new ...;
16     if(instances.size()>0) {
17         var firstInstance = instances...get();
18         // disjunctive null values are important for the average calculation
19         GalveUtil.makeDisjunctNulls(instances);
20         // search for the first instance for matching atoms in the other instances
21         loop1: for(... outerAtom : firstInstance...) {
22         ...
23         for(... instance : instances) {
24             if(!instance.equals(firstInstance)) {
25                 var found = false;
26                 for(... innerAtom : instance...) {
27                     // if an atom already contains conflicting null values delete it
28                     if(outerAtom.hasHomomorphismTo(innerAtom) &&
29                     GalveUtil.hasConflictingNulls(outerAtom, homomorphism)) {
```

```
30                          // add atom with conflicting null value
31                          toRemove = outerAtom;
32                          // note instance containing the atom with conflicting null
33                          instanceForRemove = firstInstance;
34                          // set the flag to restart the calculation
35                          breakWhile = false;
36                          // extend homomorphism to be able to recognize later conflicts
37                          GalveUtil.addTermMapping(
38                              homomorphism, outerAtom, innerAtom);
39                          // Cancel the calculation
40                          break loop1;
41                      }
42                      // just like before for innerAtom
43                      ...
44                      // homomorphism can be extended
45                      if(GalveUtil.homomorphismIsExpandable(
46                          homomorphism, outerAtom, innerAtom)) {
47                          // extend homomorphism
48                          GalveUtil.addTermMapping(homomorphism,
49                              mappedOuterAtom, mappedInnerAtom);
50                          found = true;
51                      }else if(...){// just like before for innerAtom
52                      }else
53                      // homomorphism exists, but existing homomorphism cannot be extended
54                      if(outerAtom.hasHomomorphismTo(innerAtom)) {
55                          // just like first if statement
56                          ...
57                      }else if(...) {// just like first if statement for innerAtom
58                  }}
59                  if(!found) {
60                      continue loop1;
61              }}
62              // atom was found everywhere and can be added to the result
63              resultInstance...add(GalveUtil.applyTermMappings(outerAtom, homomorphism));
64          }
65          if(breakWhile) {
66              // apply the homomorphism to all atoms already added to the result
67              resultInstance = GalveUtil.applyTermMappings(resultInstance, homomorphism);
68          }else {
69              // find the instance with conflicting null, remove the corresponding atom
70              ...
71  }}}while(!breakWhile);
72  return resultInstance;
73  }
```

6. GaLVE-Schritt Die Erstellung der TGDs und bEGDs für die Nachbereitung der Erweiterung laufen genauso ab, wie wir es in Algorithmus 7 beschrieben haben. Die Erstellung der TGDs ist in Listing 5.9 aufgeführt. Was wegen der verkürzten Darstellung nicht zu erkennen ist, ist dass der Bezeichner des Kopf-Ausdruckes der TGD einfach aus dem Bezeichner des Rumpf-Ausdruckes der TGD gebildet wird, indem das letzte Zeichen abgeschnitten wird. Das können wir so machen, da die neuen Relationennamen der zusätzlichen, für das Quellschema neuen Relationen dadurch entstanden sind, dass ein + am Ende eines bestehenden Relationennamen hinzugefügt wurde.

Die Erstellung der bEGDs wurde in Listing 5.10 angegeben. Die neu zu ChaTEAU hinzugefügte Unterstützung von Schlüsselattributen kann hier genutzt werden, um in bEGDs alle Nichtschlüsselattribute über die Schlüsselattribute gleich zu setzen. Dafür werden die Variablen für Schlüsselattribute mit demselben Index

in den beiden Rumpfatomen versehen, während die Nichtschlüsselattribute unterschiedliche Indices bekommen und in einem Gleichheitsatom in den Kopf der bEGD übernommen werden.

Listing 5.9: Nachbereitung - TGD-Erstellung

```
static ... followUpTgds(... originSchema, ... extendedSchema, ... relationKeyAttribtes){
var followUpTgds = new ...;
for(... extendedSchemaEntry : extendedSchema...) {
    // consider only extended relations
    if(!originSchema.containsKey(extendedSchemaEntry...)) {
        ...
        for(... attribut : extendedAttributes) {
            tgdBodyAtom...add(new Variable(...));
        }
        tgdBody.add(tgdBodyAtom);
        ...
        for(... attribut : originAttributes) {
            tgdHeadAtom...add(new Variable(...));
        }
        tgdHead.add(tgdHeadAtom);
        followUpTgds.add(new Tgd(tgdBody,tgdHead));

}}
return followUpTgds;
}
```

Listing 5.10: Nachbereitung - bEGD-Erstellung

```
static ... followUpBEgds(... originSchema, ... extendedSchema, ... relationKeyAttribtes){
var followUpBEgds = new ...;
for(... extendedSchemaEntry : extendedSchema...) {
    // consider only extended relations
    if(!originSchema.containsKey(extendedSchemaEntry...)) {
        ...
        // create bEgd to equate all non-key attributes
        var keyAttributes = relationKeyAttribtes.get(originRelationName);
        // there are more attributes than key attributes
        if(originAttributes.size() > keyAttributes.size()) {
            ...
            for(int i = 0; i < originAttributes.size(); i++) {
                ...
                // select according to the key attributes
                if(keyAttributes.contains(attribute)) {
                    bEgdBodyAtom1...add(new Variable(..., 1));
                    bEgdBodyAtom2...add(new Variable(..., 1));
                }else {
                    bEgdBodyAtom1...add(new Variable(..., 1));
                    bEgdBodyAtom2...add(new Variable(..., 2));
                    bEgdHead.add(new EqualityAtom(new Variable(..., 1),
                        new Variable(..., 2)));
            }}
            bEgdBody.add(bEgdBodyAtom1);
            bEgdBody.add(bEgdBodyAtom2);
            followUpBEgds.add(new BEgd(bEgdBody, bEgdHead));
}}}
return followUpBEgds;
}
```

7. GaLVE-Schritt Der siebte Schritt des GaLVE-Verfahrens besteht nur noch aus der Ausführung der TGDs und bEGD (die im sechsten GaLVE-Schritt erstellt wurden) mit Hilfe des in ChaTEAU implementierten CHASE-Algorithmus. Da dieser bisher nur die CHASE-Schritte für TGDs und EGDs unterstützt, mussten wir ihn noch um den befangenen CHASE-Schritt für bEGD nach Definition 4.10 erweitern. Die Umsetzung ist in Listing 5.11 dargestellt. Die Methode `applyBEgd` wird im ChaTEAU-CHASE für bEGDs anstatt der Methode für EGDs aufgerufen. Ihr wird eine bEGD und ein Homomorphismus übergeben, welcher der aktive Trigger für die bEGD und die Instanz im CHASE ist.

Die Anwendung einer bEGD erfolgt für jedes Gleichheitsatom des Kopfes einzeln (siehe for-Schleife in Zeile 3) und startet damit, dass der Homomorphismus in Zeile 9 auf die Terme des Gleichheitsatoms angewendet wird. In Zeile 10 werden die Atome mit den zugehörigen Termpositionen identifiziert, für die die Termabbildung, die auf den Termen des Gleichheitsatoms angewendet wurde, erzeugt wurden. Da es sich bei dem Homomorphismus um einen aktiven Trigger und somit um Termabbildungen von der bEGD zu einer Instanz handelt, wird nur eine Termabbildung pro Term des Gleichheitsatoms angewendet, wodurch die Atome, die die Termabbildung erzeugt haben, eindeutig sind. In den Zeilen 12-29 werden die Fälle der Definition 4.10 umgesetzt, bei denen zwei unterschiedliche Konstanten aufeinander abgebildet werden sollen. Für die restlichen Fälle wird in Zeile 33 eine neue Termabbildung von einem der Terme des Gleichheitsatoms zum anderen erzeugt. Die Methode `buildMaxToMinMapping` sorgt dafür, dass Nullwerte auf Konstanten bzw. Nullwerte mit dem größeren Index auf Nullwerte mit dem kleineren Index abgebildet werden. Bildet diese neue Termabbildung von einem Nullwert auf eine Konstante ab, werden in den Zeilen 43-64 die entsprechenden Fälle der Definition 4.10 umgesetzt. Bildet die neue Termabbildung zwei Nullwerte aufeinander ab, wird sie in Zeile 68 zu dem Homomorphismus hinzugefügt.

Ein Problem bei dieser Implementierung des befangenen CHASE-Schritts für bEGDs in ChaTEAU ist, dass Konstanten aus neuen Atomen direkt in der originalen Instanz ersetzt werden, falls sie mit anderen Konstanten aus neuen oder originalen Atomen gleichgesetzt werden (siehe Zeilen 17, 22 und 27). Ändern wir diese Konstanten direkt in der originalen Instanz, kann diese Veränderung von ChaTEAU nicht erkannt werden. Das liegt daran, dass alle Änderungen normalerweise erst in einer Kopie der originalen Instanz durchgeführt werden, die dann für die Ermittlung der veränderten Daten mit der ursprünglichen Instanz verglichen wird. Da Atome mit direkt angepassten Konstanten in der originalen Instanz nicht als neue Atome für die Instanz erkannt werden, liefert der ChaTEAU-CHASE keine Veränderung der Instanz zwischen zwei Durchläufen über alle bEGDs, erreicht somit seine Abbruchbedingung und terminiert. Wir haben uns dazu entschieden, diese Eigenschaft nicht in ChaTEAU zu beheben, sonder den ChaTEAU-CHASE in der *GalveMain-main*-Methode (siehe Anhang C) sooft mit den bEGDs aufzurufen, bis sich das Ergebnis nicht mehr ändert. Sollte sich die Instanz für zwei aufeinanderfolgende Ausführungen des CHASE mit den bEGDs nicht ändern, haben wir die nachbereitete erweiterte Instanz unter dem erweiterten Quellschema gefunden und somit das Endergebnis unseres GaLVE-Verfahrens.

Listing 5.11: CHASE-Schritt mit bEGDs

```
1  private void applyBEgd(... bEgd, ... homomorphism) {
2  ...
3  for (... equalityHeadAtom : bEgd.getHead()) {
4  // temporary terms that are the result of applying the mappings of homomorphism
5  // to the equality atoms of the egd
6  var leftTerm = equalityHeadAtom.getTerm1();
7  var rightTerm = equalityHeadAtom.getTerm2();
8  // apply all mappings in homomorphism and get the atoms that created the mapping
9  ...
10 var leftAtom, rightAtom, leftTermPosition, rightTermPosition = ...;
11 // if both terms of the equality atom are constants
12 if (ChaseUtil.bothTermTypesAreConst(leftTerm, rightTerm)) {
13     if (!ChaseUtil.bothTermValuesAreEqual(leftTerm, rightTerm)) {
```

```
14          // left is origin & right is not origin
15          if(leftAtom.isOrigin() && !rightAtom.isOrigin()) {
16              // replace constant from new atom with constant from original atom
17              ...
18          }else
19          // right is origin & left is not origin
20          if(rightAtom.isOrigin() && !leftAtom.isOrigin()) {
21              // replace constant from new atom with constant from original atom
22              ...
23          }else
24          // left is not origin & right is not origin
25          if(!rightAtom.isOrigin() && !leftAtom.isOrigin()) {
26              // replace both constants with a new null value
27              ...
28          // left is origin & right is origin
29          }else { printErrorMessage(leftTerm, rightTerm);
30      }}
31  } else {
32      if (ChaseUtil.shouldExecuteMapping(leftTerm, rightTerm)) {
33          var maxToMinMapping = ChaseUtil.buildMaxToMinMapping(leftTerm, rightTerm);
34          // give the possibility to replace single constants from new
35          // tuples with other constants from original tuples with bEgds
36          ...
37          // mapping from null to constant
38          if(...) {
39              var mappingNull = maxToMinMapping.getMappingSource();
40              var mappingConstant = maxToMinMapping.getMappingTarget();
41              // homomorphismCacheForEGDs already has a mapping from
42              // the null value to a constant from an original tuple
43              if(...) { // noop
44              }else
45              // homomorphismCacheForEGDs already has a mapping from
46              // the null value to a constant from a new tuple
47              if(...) {
48                  var mappingToRemove = ...;
49                  // new mapping maps to a constant from an original tuple
50                  // left term is the constant and left atom is original
51                  // or right term is the constant and right atom is original
52                  if(...) {
53                      // replaced the previous mapping
54                      homomorphismCacheForEGDs...removeAll(mappingToRemove);
55                      homomorphismCacheForEGDs.addMapping(maxToMinMapping);
56                  }else
57                  // new mapping maps to ANOTHER constant from a new tuple
58                  if(...) {
59                      // delete the previous mapping without replacement
60                      homomorphismCacheForEGDs...removeAll(mappingToRemove);
61                  }
62                  // such a mapping does not yet exist
63              }else { homomorphismCacheForEGDs.addMapping(maxToMinMapping);
64                  ...
65          }}
66          // mapping from null to null
67          else { homomorphismCacheForEGDs.addMapping(maxToMinMapping);
68              ...
69  }}}
70  instanceWithNewFacts = homomorphismCacheForEGDs.applyMappingsTo(instanceWithNewFacts);
71  ...
72  }}
```

6. Fazit und Ausblick

Fazit Ziel dieser Arbeit war die Konzeption eines Verfahrens, das mit Hilfe von inversen Schemaabbildungen eine Datenintegration als Erweiterung lokaler relationaler Datenbanken umsetzt. Unser neu entwickeltes GaLVE-Verfahren macht genau das. Indem wir beliebige durch S-T TGDs formulierte Schemaabbildungen invertieren, erweitern und nachbereiten, erhalten wir eine um neue Tupel und interessante Attribute erweiterte Quellinstanz unter einem erweiterten Quellschema. Bestehende Anfragen an dieses erweiterte Quellschema können wie gewohnt angewendet werden und haben direkten Zugriff auf die neuen Tupel. Neue Anfragen können weiterhin auf dem bekannten Quellschema entwickelt werden und dann bei Bedarf die neuen Attribute in der Erweiterung des Quellschemas nutzen.

Für die Entwicklung unseres GaLVE-Verfahrens haben wir uns im 2. Kapitel zuerst die nötigen Grundlagen des Relationenmodells, der Datenintegration, der Schemaabbildungen und des CHASE-Algorithmus erarbeitet. Im 3. Kapitel haben wir dann nicht nur einen Überblick über die bisher erfolgte Forschung zur Invertierung von Schemaabbildungen gegeben, sondern diese gleichzeitig auch analysiert und verglichen, um eine geeignete Theorie für unser Konzept auszuwählen. Der Hauptbestandteil dieser Arbeit ist die Konzeptentwicklung des neuen GaLVE-Verfahrens im 4. Kapitel. Dort haben wir mit den *Disjunktiven-Mengen* eine abgewandelte Darstellung der disjunktiven TGDs vorgestellt, die als Ergebnis eines neuen, besonders einfachen Invertierungsalgorithmus für beliebige Schemaabbildungen entsteht. Für die Verarbeitung dieser inversen Schemaabbildungen mit den Disjunktiven-Mengen haben wir einen *angepassten Disjunktiven-CHASE* und Algorithmen für eine *Sichere Instanz* angegeben. Die Erweiterung der Quellen setzt ebenfalls auf unsere inversen Schemaabbildungen mit Disjunktiven-Mengen auf. Wir haben beschrieben, wie die Erweiterung der inversen Schemaabbildungen auflaufen muss, um die originalen Daten in den Quellen zuerst zu bewahren und wie die so erweiterte inverse Schemaabbildung dann mit TGDs und den in dieser Arbeit vorgestellten *bEGDs* nachbereitet werden muss, um die originalen Daten um neue, konsistente Informationen aus der globalen Datenbank zu erweitern. Die Implementierung unseres Konzepts wurde als Erweiterung vom Datenbankforschungsprototypen ChaTEAU umgesetzt und ist in Kapitel 5 beschrieben. Mit ihr kann das neue GaLVE-Verfahren leicht mit verschiedenen interessanten Fällen von Schemaabbildungen getestet werden. Zusammengefasst kann man sagen, dass das Ziel dieser Arbeit erfüllt wurde.

Ausblick Unser neues GaLVE-Verfahren bildet mit der Möglichkeit, eine Datenintegration auf den Quellen von Schemaabbildungen durchzuführen, eine gute Grundlage für folgende Arbeiten. Gerade auf den Algorithmus für die Invertierung beliebiger Schemaabbildungen und den damit verbundenen Disjunktiven-Mengen könnten weitere Konzepte der Datenbankforschung aufbauen.

In direktem Anschluss an diese Arbeit wäre eine Überprüfung der Grenzen des GaLVE-Verfahrens sinnvoll. Dafür müssten zum einen weitere Sonderfälle der Datenintegration erarbeitet werden, aber auch mehr praktische Beispiele getestet werden. Mit der praktischen Nutzung des GaLVE-Verfahrens wird zudem die Optimierung der Effizienz der theoretischen Algorithmen und der implementierten GaLVE-Techniken notwendig. Letztere müssten ebenfalls noch intensiver auf ihre Korrektheit geprüft werden, was aufgrund des limitierten zeitlichen Rahmens dieser Arbeit nicht ausreichend möglich war.

Neben bzw. anstatt der Optimierung der vorgestellten Algorithmen des GaLVE-Verfahrens kann es sinnvoll sein, bestehende Algorithmen auf die Anwendbarkeit für die einzelnen GaLVE-Techniken zu untersuchen. Die Literaturrecherche zu dieser Arbeit beschränkte sich auf die Algorithmen zur Invertierung von Schemaabbildungen. Alle auf die Invertierung von Schemaabbildungen und der Verarbeitung von inversen Schemaabbildungen mit Disjunktiven-Mengen folgenden GaLVE-Techniken wurden eigens für ihren Anwendungszweck entwickelt, ohne bestehende Techniken in Betracht zu ziehen. Hier sei vor allem der Algorithmus für die Sichere Instanz mit dem *Durchschnitt unter einem Homomorphismus* und die Theorie hinter den neu definierten bEGDs genannt. Eventuell gibt es ähnliche, bewährte Konzepte, die für die Übernahme in das GaLVE-Verfahren geeignet sind. Diese sollten mit den in dieser Arbeit vorgestellten Techniken verglichen werden und diese (wenn angebracht) ersetzen. Ein konkretes Beispiel ist der Vergleich der Theorie zum Data Cleaning im CHASE-Tool *Llunatic* [GMPS14, Llu21] mit der Theorie der bEGDs.

Eine denkbare Erweiterung des vorgestellten GaLVE-Verfahrens wäre eine Art Data Cleaning-Vorbereitung während der zuvor ablaufenden Datenintegration. In unserem Konzept haben wir uns auf die Schritte nach der Datenintegration konzentriert und Inkonsistenzen zwischen den neuen Tupeln und neuen Attributen mit Hilfe von bEGDs in der Nachbereitung der erweiterten inversen Schemaabbildung bereinigt. So wurden alle inkonsistenten Daten aus der globalen Datenbank durch Daten der lokalen Datenbank oder durch Nullwerte ersetzt.
Es wäre nun auch möglich, diese Art der Datenkonflikte durch ein Eingreifen des GaLVE-Verfahrens in der Datenintegration zu minimieren oder ganz zu vermeiden. Die Datenintegration vor dem GaLVE-Verfahren müsste dafür so angepasst werden, dass sie mehrere globale Datenbanken unter einem globalen Schema erzeugt. Spätestens wenn in der globalen Hauptdatenbank Datenkonflikte zwischen Daten aus zwei Quelldatenbanken entstehen, würde für die Quelldatenbanken der in Konflikt stehenden Daten jeweils eine globale Extradatenbank erstellt werden. Beim Auftreten von Datenkonflikten ist es dann möglich, anstatt ein Tupel zu der globalen Hauptdatenbank hinzu zufügen, jeweils einen in Konflikt stehenden Datensatz in die jeweilige globale Extradatenbank für die entsprechende Quelldatenbank zu speichern (bestehende in Konflikt stehende Tupel müssten aus der globalen Haupt- in die entsprechende Extradatenbank kopiert werden). Die globale Hauptdatenbank bestände dann nur aus den konsistenten integrierten Daten und die Extradatenbanken nur aus den untereinander in Konflikt stehenden Daten.
Bei der Rückabbildung in eine Quelldatenbank mit einer erweiterten inversen Schemaabbildung $\mathcal{M}'$, kann man $\mathcal{M}'$ auf der globalen Hauptdatenbank und zusätzlich auf ausgewählten globalen Extradatenbanken ausführen. In der Nachbereitung der erweiterten inversen Schemaabbildung müssen dann nur die Daten bereinigt werden, die aus den ausgewählten globalen Extradatenbanken stammen.
Weiter könnte man für die anderen Quelldatenbanken, deren in Konflikt stehende Daten man integrieren möchte, eine Art Vertrauensordnung angeben und diese bei der Konfliktauflösung der Daten aus den globalen Extradatenbanken beachten. Die Vertrauensordnung müsste dann von einer speziellen EGD aufgelöst werden, die jeweils die Tupel aus einer Datenbank mit höherer Vertrauensordnung den Tupeln aus anderen Datenbanken vorzieht. Die EGD könnte ähnlich zu den in dieser Arbeit vorgestellten bEGDs, mit Provenance-Annotationen für die Vertrauensordnung definiert werden.

Eine weitere Möglichkeit für eine Folgearbeit wäre die Umsetzung eines GaLVE-Verfahrens, welche sich näher am eigentlichen Ziel der Ursprungsidee aus [FHLM96] orientiert. Dazu zählt z. B. die Erweiterung der lokalen Datenbank um die komplette globale Datenbank, damit die lokale Datenbank wie gewohnt angefragt werden kann, aber auch die gesamte globale Datenbank lokal zur Verfügung steht. In unserem GaLVE-Verfahren erweitern wir die lokale Datenbank nur mit Daten der globalen Datenbank, die in Zusammenhang mit den Relationen in der lokalen Datenbank stehen. Dies können wir einfach ändern, indem wir auch die Relationen im globalen Schema, von denen keine Abbildungen in das Quellschema ausgehen, mit Hilfe einer neuen S-T TGD in einer zusätzlichen Disjunktiven-Menge zu der Quelldatenbank als neue Relation hinzufügen.

Zu den Zielen der Ursprungsidee des GaLVE-Verfahrens zählt auch die Beachtung von heterogenen lokalen Datenbanksystemen und deren zusätzliche Erweiterung, „um fehlende semantische Ausdrucksstärke und um Konzepte [...], die im jeweiligen lokalen Schema und dem zugrundeliegendem Datenmodell, aber auch dem Datenbanksystem fehlen“[FHLM96] sowie „die Entwicklung geeigneter Transaktionskonzepte, die komplexe Änderungsoperationen über einem föderierten Datenbestand erlauben“[FHLM96]. Diese Ziele sind eventuell nur mit komplett neuen GaLVE-Konzepten umsetzbar.

Die wichtigsten weiteren Forschungsansätze zum GaLVE-Verfahren beziehen sich auf die Einschränkungen der Schemaabbildungen und Anfragen durch die Verwendung von S-T TGDs sowie auf die Kombination vom GaLVE-Verfahren mit weiteren Konzepten der Datenbankforschung. Die durch die S-T TGDs folgenden Einschränkungen müssen spätestens dann umgangen und die S-T TGDs zu einer kompletten Anfragesprache erweitert werden, wenn man bestehende Schemaabbildungen für eine Verwendung im GaLVE-Verfahren transformieren möchte. Derzeit können nur konjunktive Anfragen mit Konstanten-Selektion als S-T TGDs formuliert und als Schemaabbildung genutzt werden. Der vorgestellte angepasste Disjunktive-CHASE unterstützt mit den Disjunktiven-Mengen zwar noch eine Art Disjunktion in den Schemaabbildungen und Anfragen, ist im GaLVE-Verfahren allerdings nur für die Rückabbildung mit einer inversen Schemaabbildung in Verbindung mit der Sicheren Instanz vorgesehen.
Möchte man das GaLVE-Verfahren mit anderen Konzepten der Datenbankforschung kombinieren, müssen mindestens die gleichen Probleme gelöst werden wie bei der Kombination der einzelnen GaLVE-Techniken untereinander. So wird bei der Kombination von anderen Techniken der Schemaabbildungstheorie mit den Disjunktiven-Mengen immer eine spezielle Darstellung der Komposition von S-T TGDs mit Disjunktiven-Mengen benötigt. Da diese Darstellung Second-Order TGDs beinhalten wird, muss dann nicht nur das GaLVE-Verfahren, sondern auch ChaTEAU um deren Unterstützung erweitert werden.

Literaturverzeichnis

[ABU79] Aho, A. V. ; Beeri, C. ; Ullman, J. D.: The Theory of Joins in Relational Databases. In: *ACM Trans. Database Syst.* 4 (1979), September, Nr. 3, 297–314. http://dx.doi.org/10.1145/320083.320091. – DOI 10.1145/320083.320091. – ISSN 0362–5915

[AH18a] Auge, Tanja ; Heuer, Andreas: Inverses in Research Data Management: Combining Provenance Management, Schema and Data Evolution (Inverse im Forschungsdatenmanagement). In: *Proceedings of the 30th GI-Workshop Grundlagen von Datenbanken, Wuppertal, Germany, May 22-25, 2018.*, 2018, 108–113

[AH18b] Auge, Tanja ; Heuer, Andreas: The Theory behind Minimizing Rese arch Data: Result equivalent CHASE-inverse Mappings. In: *Proceedings of the Conference "Lernen, Wissen, Daten, Analysen", LWDA 2018, Mannheim, Germany, August 22-24, 2018.*, 2018, 1–12

[AH19] Auge, Tanja ; Heuer, Andreas: ProSA—Using the CHASE for Provenance Management. In: Welzer, Tatjana (Hrsg.) ; Eder, Johann (Hrsg.) ; Podgorelec, Vili (Hrsg.) ; Kamišalić Latifić, Aida (Hrsg.): *Advances in Databases and Information Systems*. Cham : Springer International Publishing, 2019. – ISBN 978–3–030–28730–6, S. 357–372

[APR09] Arenas, Marcelo ; Pérez, Jorge ; Riveros, Cristian: The recovery of a schema mapping: Bringing exchanged data back. In: *ACM Trans. Database Syst.* 34 (2009), Nr. 4, 22:1–22:48. http://dx.doi.org/10.1145/1620585.1620589. – DOI 10.1145/1620585.1620589

[APR11] Arenas, Marcelo ; Perez, Jorge ; Reutter, Juan: Data Exchange Beyond Complete Data, 2011, S. 83–94

[APRR09] Arenas, Marcelo ; Pérez, Jorge ; Reutter, Juan L. ; Riveros, Cristian: Composition and Inversion of Schema Mappings. In: *CoRR* abs/0910.3372 (2009). http://arxiv.org/abs/0910.3372

[APRR13] Arenas, Marcelo ; Pérez, Jorge ; Reutter, Juan ; Riveros, Cristian: The language of plain SO-tgds: Composition, inversion and structural properties. In: *Journal of Computer and System Sciences* 79 (2013), Nr. 6, 763-784. http://dx.doi.org/https://doi.org/10.1016/j.jcss.2013.01.002. – DOI https://doi.org/10.1016/j.jcss.2013.01.002. – ISSN 0022–0000. – JCSS Foundations of Data Management

[Aug17] Auge, Tanja: *Umsetzung von Provenance-Anfragen in Big-Data-Analytics-Umgebungen*, Universität Rostock, Diplomarbeit, 2017

[BKM+17] Benedikt, Michael ; Konstantinidis, George ; Mecca, Giansalvatore ; Motik, Boris ; Papotti, Paolo ; Santoro, Donatello ; Tsamoura, Efthymia: Benchmarking the Chase. In: *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2017, Chicago, IL, USA, May 14-19, 2017*, 2017, 37–52

[BLM+15] Baget, Jean-François ; Leclère, Michel ; Mugnier, Marie-Laure ; Rocher, Swan ; Sipieter, Clément: Graal: A Toolkit for Query Answering with Existential Rules. In: *Rule Technologies: Foundations, Tools, and Applications - 9th International Symposium, RuleML 2015, Berlin, Germany, August 2-5, 2015, Proceedings*, 2015, 328–344

[Con97] Conrad, Stefan: *Föderierte Datenbanksysteme - Konzepte der Datenintegration.* Springer, 1997 http://www.springer.com/computer/database+management+%26+information+retrieval/book/978-3-540-63176-7. – ISBN 978–3–540–63176–7

[DHI12] Doan, AnHai ; Halevy, Alon Y. ; Ives, Zachary G.: *Principles of Data Integration.* Morgan Kaufmann, 2012 http://research.cs.wisc.edu/dibook/. – ISBN 978–0–12–416044–6

[DN07] Dinu, Valentin ; Nadkarni, Prakash: Guidelines for the effective use of entity-attribute-value modeling for biomedical databases. In: *International Journal of Medical Informatics* 76 (2007), November, Nr. 11-12, S. 769–779. http://dx.doi.org/10.1016/j.ijmedinf.2006.09.023. – DOI 10.1016/j.ijmedinf.2006.09.023. – ISSN 1386–5056

[DNR08] Deutsch, Alin ; Nash, Alan ; Remmel, Jeff: The Chase Revisited. In: *Proceedings of the Twenty-seventh ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems.* New York, NY, USA : ACM, 2008 (PODS '08). – ISBN 978–1–60558–152–1, 149–158

[Fag07] Fagin, Ronald: Inverting Schema Mappings. In: *ACM Trans. Database Syst.* 32 (2007), November, Nr. 4, 25–es. http://dx.doi.org/10.1145/1292609.1292615. – DOI 10.1145/1292609.1292615. – ISSN 0362–5915

[FHLM96] Flach, Guntram ; Heuer, Andreas ; Langer, Uwe ; Meyer, Holger: Transparente Anfragen in föderativen Datenbanksystemen, Universität Rostock, 1996

[FKMP03] Fagin, Ronald ; Kolaitis, Phokion G. ; Miller, Renée J. ; Popa, Lucian: Data Exchange: Semantics and Query Answering. In: Calvanese, Diego (Hrsg.) ; Lenzerini, Maurizio (Hrsg.) ; Motwani, Rajeev (Hrsg.): *Database Theory — ICDT 2003.* Berlin, Heidelberg : Springer Berlin Heidelberg, 2003. – ISBN 978–3–540–36285–2, S. 207–224

[FKMP05] Fagin, Ronald ; Kolaitis, Phokion G. ; Miller, R. ; Popa, Lucian: Data exchange: semantics and query answering. In: *Theor. Comput. Sci.* 336 (2005), S. 89–124

[FKPT05] Fagin, Ronald ; Kolaitis, Phokion G. ; Popa, Lucian ; Tan, Wang-Chiew: Composing Schema Mappings: Second-Order Dependencies to the Rescue. In: *ACM Trans. Database Syst.* 30 (2005), Dezember, Nr. 4, 994–1055. http://dx.doi.org/10.1145/1114244.1114249. – DOI 10.1145/1114244.1114249. – ISSN 0362–5915

[FKPT08] Fagin, Ronald ; Kolaitis, Phokion G. ; Popa, Lucian ; Tan, Wang-Chiew: Quasi-Inverses of Schema Mappings. In: *ACM Trans. Database Syst.* 33 (2008), Juni, Nr. 2. http://dx.doi.org/10.1145/1366102.1366108. – DOI 10.1145/1366102.1366108. – ISSN 0362–5915

[FKPT11a] Fagin, Ronald ; Kolaitis, Phokion G. ; Popa, Lucian ; Tan, Wang-Chiew: Reverse Data Exchange: Coping with Nulls. 36 (2011), Juni, Nr. 2. http://dx.doi.org/10.1145/1966385.1966389. – DOI 10.1145/1966385.1966389. – ISSN 0362–5915

[FKPT11b] Fagin, Ronald ; Kolaitis, Phokion G. ; Popa, Lucian ; Tan, Wang C.: Schema Mapping Evolution Through Composition and Inversion. Version: 2011. http://dx.doi.org/10.1007/978-3-642-16518-4_7. In: *Schema Matching and Mapping.* 2011. – DOI 10.1007/978–3–642–16518–4_7, 191–222

[GM15] Greco, Sergio ; Molinaro, Cristian: Datalog and Logic Databases. In: *Synthesis Lectures on Data Management* 7 (2015), Nr. 2, S. 1–169. http://dx.doi.org/10.2200/S00648ED1V01Y201505DTM041. – DOI 10.2200/S00648ED1V01Y201505DTM041

[GMO15] Grahne, Gösta ; Moallemi, Ali ; Onet, Adrian: Recovering Exchanged Data. In: *Proceedings of the 34th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*. New York, NY, USA : Association for Computing Machinery, 2015 (PODS '15). – ISBN 9781450327572, 105–116

[GMPS14] Geerts, Floris ; Mecca, G. ; Papotti, Paolo ; Santoro, Donatello: Mapping and Cleaning: the LLUNATIC Way, 2014

[GMS12] Greco, Sergio ; Molinaro, Cristian ; Spezzano, Francesca: *Incomplete Data and Data Dependencies in Relational Databases*. Morgan & Claypool Publishers, 2012. – ISBN 1608459268, 9781608459261

[Gra21] *Graal Homepage*. http://graphik-team.github.io/graal/, Dezember 2021. – Accessed: 2021-08-30

[Gö20] Görres, Andreas: Erweiterung des CHASE-Werkzeugs ChaTEAU um ein Terminierungskriterium, Universität Rostock, 2020. – Masterarbeit

[HSS18] Heuer, Andreas ; Saake, Gunter ; Sattler, Kai-Uwe: *Datenbanken - Konzepte und Sprachen, 6. Auflage*. MITP, 2018 `https://mitp.de/IT-WEB/Datenbanken/Datenbanken-Konzepte-und-Sprachen-oxid.html`. – ISBN 978–3–9584577–6–8

[Jur18] Jurklies, Martin: CHASE und BACKCHASE: Entwicklung eines Universal-Werkzeugs für eine Basistechnik der Datenbankforschung, Universität Rostock, 2018. – Masterarbeit

[Llu21] *Llunatic Homepage*. http://www.db.unibas.it/projects/llunatic/, Dezember 2021. – Accessed: 2021-08-30

[LMS95] Levy, Alon Y. ; Mendelzon, Alberto O. ; Sagiv, Yehoshua: Answering Queries Using Views (Extended Abstract). In: *Proceedings of the Fourteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*. New York, NY, USA : ACM, 1995 (PODS '95). – ISBN 0–89791–730–8, 95–104

[MG12] Mohapatra, Abhijeet ; Genesereth, Michael: Aggregation in Datalog Under Set Semantics / Stanford University. Stanford, CA, 2012 (LG-2012-01). – Forschungsbericht. – http://logic.stanford.edu/reports/LG-2012-01.pdf

[MMS79] Maier, David ; Mendelzon, Alberto O. ; Sagiv, Yehoshua: Testing Implications of Data Dependencies. In: *ACM Trans. Database Syst.* 4 (1979), Dezember, Nr. 4, 455–469. `http://dx.doi.org/10.1145/320107.320115`. – DOI 10.1145/320107.320115. – ISSN 0362–5915

[ÖV11] Özsu, M. T. ; Valduriez, Patrick: *Principles of Distributed Database Systems, Third Edition*. Springer, 2011. `http://dx.doi.org/10.1007/978-1-4419-8834-8`. `http://dx.doi.org/10.1007/978-1-4419-8834-8`. – ISBN 978–1–4419–8833–1

[Pér11] Pérez, Jorge: Schema Mapping Management in Data Exchange Systems, Escuela de Ingeniería, Pontificia Universidad Católica de Chile, 2011. – PhD thesis

[Pér13] Pérez, Jorge: The Inverse of a Schema Mapping. Version: 2013. `http://nbn-resolving.de/urn/resolver.pl?urn=urn:nbn:de:0030-drops-42909`. In: Kolaitis, Phokion G. (Hrsg.) ; Lenzerini, Maurizio (Hrsg.) ; Schweikardt, Nicole (Hrsg.): *Data Exchange, Integration, and Streams* Bd. 5. Dagstuhl, Germany : Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2013. – URN urn:nbn:de:0030–drops–42909. – ISBN 978–3–939897–61–3, 69–95

[Ren19] Renn, Fabian: Erweiterung des CHASE-Werkzeugs ChaTEAU um Anfragetransformationen, Universität Rostock, 2019. – Bachelorarbeit

[Ros20] Rose, Florian: Erweiterung des CHASE-Werkzeugs ChaTEAU um eine BACKCHASE-Phase., Universität Rostock, 2020. – Masterarbeit

[SBLH14] Straube, Georgi ; Bruder, Ilvio ; Löper, Dortje ; Heuer, Andreas: Data Integration in a Clinical Environment Using the Global-as-Local-View-Extension Technique. In: *HIS* Bd. 8423, Springer, 2014 (Lecture Notes in Computer Science), S. 148–159

[SP94] Spaccapietra, S. ; Parent, C.: View integration: a step forward in solving structural conflicts. In: *IEEE Transactions on Knowledge and Data Engineering* 6 (1994), Nr. 2, S. 258–274. http://dx.doi.org/10.1109/69.277770. – DOI 10.1109/69.277770

[SPD92] Spaccapietra, Stefano ; Parent, Christine ; Dupont, Y.: Model Independent Assertions forIntegration of Heterogeneous Schema. In: *VLDB Journal* 1 (1992), Nr. 1, 81-126. http://dx.doi.org/10.1007/BF01228708. – DOI 10.1007/BF01228708

[Str13] Straube, Georgi: Datenintegration für die „Global-as-local-view-extension"-Technik, Universität Rostock, 2013. – Masterarbeit

[Zim20] Zimmer, Jakob: Vereinheitlichung des CHASE auf Instanzen und Anfragen am Beispiel ChaTEAU, Universität Rostock, 2020. – Bachelorarbeit

[Zim21] Zimmer, Jakob: Aktuelle Forschung am DBIS-Lehrstuhl, Universität Rostock, 2021. – Ringvorlesung, WS 2020/21

A. Laufendes Beispiel in ChaTEAU

```
Read Input_1 from File: /K.xml

Read Input_2 from File: /R.xml

***Start data integration...***

-Input_1:
K-NOTEN(2, 2, "SS 21", 4.0, "Knapp")
K-NOTEN(2, 3, #N_semester_1, 2.0, #N_Bemerkung_1)
K-PRAKTIKUM(3)
K-SOFTSKILL(3)
K-SOFTSKILL(2)
K-STUDIERENDE(3, #N_lastname_1, "Max", "Master")
K-STUDIERENDE(2, "Sonnensand", #N_firstname_1, "Bachelor")

-Constraints_1:
[
K-STUDIERENDE(#V_Matrikelnr_1, #V_Nachname_1, #V_Vorname_1, #V_Abschluss_1)
->
T-STUDIERENDE(#V_Matrikelnr_1, #V_Nachname_1, #V_Vorname_1, #E_Studiengang_1,
        #V_Abschluss_1),

K-NOTEN(#V_Modulnr_1, #V_Matrikelnr_1, #V_Semester_1, #V_Note_1,
        #V_Bemerkung_1)
->
T-NOTEN(#V_Modulnr_1, #V_Matrikelnr_1, #V_Semester_1, #V_Note_1,
        "Von Kiel importiert"),
T-ZUSATZ(#V_Matrikelnr_1),

K-PRAKTIKUM(#V_Matrikelnr_1)
->
T-ZUSATZ(#V_Matrikelnr_1),

K-SOFTSKILL(#V_Matrikelnr_1)
->
T-ZUSATZ(#V_Matrikelnr_1)]

-Output1:
T-NOTEN(2, 2, "SS 21", 4.0, "Von Kiel importiert")
T-NOTEN(2, 3, #N_semester_1, 2.0, "Von Kiel importiert")
T-STUDIERENDE(3, #N_lastname_1, "Max", #N_Studiengang_1, "Master")
T-STUDIERENDE(2, "Sonnensand", #N_firstname_1, #N_Studiengang_2, "Bachelor")
T-ZUSATZ(2)
T-ZUSATZ(3)
```

```
-Input_2:
R-NOTEN(2, "SS 21", 3.0)
R-NOTEN(3, "WS 20/21", 2.3)
R-NOTEN(1, "WS 20/21", 1.7)
R-STUDIERENDE(1, "Fieber", "Fabian", "Informatik", "Tel: 0123456789")
R-STUDIERENDE(2, "Sonne", "Sarah", "ITTI", "Corona(+) am 07.12.2020")
R-TEILNAHME(3, 1, 2)
R-TEILNAHME(1, 1, 1)
R-TEILNAHME(2, 2, 1)
R-ZUSATZ(1, "Softskill")
R-ZUSATZ(2, "Praktikum")
R-ZUSATZ(1, "Praktikum")

-Constraints_2:
[
R-STUDIERENDE(#V_Matrikelnr_1, #V_Nachname_1, #V_Vorname_1, #V_Studiengang_1,
        #V_Bemerkung_1)
->
T-STUDIERENDE(#V_Matrikelnr_1, #V_Nachname_1, #V_Vorname_1, #V_Studiengang_1,
        #E_Abschluss_1),

R-NOTEN(#V_Teilnahmenr_1, #V_Semester_1, #V_Note_1),
R-TEILNAHME(#V_Teilnahmenr_1, #V_Modulnr_1, #V_Matrikelnr_1)
->
T-ZUSATZ(#V_Matrikelnr_1),
T-NOTEN(#V_Modulnr_1, #V_Matrikelnr_1, #V_Semester_1, #V_Note_1, #E_Bemerkung_1),

R-ZUSATZ(#V_Matrikelnr_1, #V_Leistung_1)
->
T-ZUSATZ(#V_Matrikelnr_1)]

-Output2:
T-NOTEN(1, 2, "WS 20/21", 2.3, #N_Bemerkung_3)
T-NOTEN(2, 1, "SS 21", 3.0, #N_Bemerkung_2)
T-NOTEN(1, 1, "WS 20/21", 1.7, #N_Bemerkung_1)
T-STUDIERENDE(2, "Sonne", "Sarah", "ITTI", #N_Abschluss_2)
T-STUDIERENDE(1, "Fieber", "Fabian", "Informatik", #N_Abschluss_1)
T-ZUSATZ(2)
T-ZUSATZ(1)

-Global instance after integration:
T-NOTEN(2, 2, "SS 21", 4.0, "Von Kiel importiert")
T-NOTEN(1, 2, "WS 20/21", 2.3, #N_Bemerkung_3)
T-NOTEN(2, 1, "SS 21", 3.0, #N_Bemerkung_2)
T-NOTEN(1, 1, "WS 20/21", 1.7, #N_Bemerkung_1)
T-NOTEN(2, 3, #N_semester_1, 2.0, "Von Kiel importiert")
T-STUDIERENDE(2, "Sonne", "Sarah", "ITTI", #N_Abschluss_2)
T-STUDIERENDE(3, #N_lastname_1, "Max", #N_Studiengang_1, "Master")
T-STUDIERENDE(2, "Sonnensand", #N_firstname_1, #N_Studiengang_2, "Bachelor")
T-STUDIERENDE(1, "Fieber", "Fabian", "Informatik", #N_Abschluss_1)
T-ZUSATZ(2)
T-ZUSATZ(3)
T-ZUSATZ(1)
```

```
-Generated global scheme:
T-ZUSATZ {Matrikelnr=int}
T-NOTEN {Modulnr=int, Matrikelnr=int, Semester=string, Note=double,
        Bemerkung=string}
T-STUDIERENDE {Matrikelnr=int, Nachname=string, Vorname=string,
        Studiengang=string, Abschluss=string}


******Start GaLVE for first input...******

***Start calculation inverse mapping...***

-Disjunctive sets in inverse mapping:

{T-NOTEN} = [
T-NOTEN(#V_Modulnr_1, #V_Matrikelnr_1, #V_Semester_1, #V_Note_1,
        "Von Kiel importiert")
->
K-NOTEN(#V_Modulnr_1, #V_Matrikelnr_1, #V_Semester_1, #V_Note_1, #E_Bemerkung_1)]

{T-ZUSATZ} = [
T-ZUSATZ(#V_Matrikelnr_1)
->
K-NOTEN(#E_Modulnr_1, #V_Matrikelnr_1, #E_Semester_1, #E_Note_1, #E_Bemerkung_1),

T-ZUSATZ(#V_Matrikelnr_1)
->
K-PRAKTIKUM(#V_Matrikelnr_1),

T-ZUSATZ(#V_Matrikelnr_1)
->
K-SOFTSKILL(#V_Matrikelnr_1)]

{T-STUDIERENDE} = [
T-STUDIERENDE(#V_Matrikelnr_1, #V_Nachname_1, #V_Vorname_1, #V_Studiengang_1,
        #V_Abschluss_1)
->
K-STUDIERENDE(#V_Matrikelnr_1, #V_Nachname_1, #V_Vorname_1, #V_Abschluss_1)]


***Start calculation optimized inverse mapping...***

-Disjunctive sets after optimization:

{T-NOTEN} = [
T-NOTEN(#V_Modulnr_1, #V_Matrikelnr_1, #V_Semester_1, #V_Note_1,
        "Von Kiel importiert")
->
K-NOTEN(#V_Modulnr_1, #V_Matrikelnr_1, #V_Semester_1, #V_Note_1, #E_Bemerkung_1)]

{T-ZUSATZ} = [
T-ZUSATZ(#V_Matrikelnr_1)
```

```
->
K-NOTEN(#E_Modulnr_1, #V_Matrikelnr_1, #E_Semester_1, #E_Note_1, #E_Bemerkung_1),

T-ZUSATZ(#V_Matrikelnr_1)
->
K-PRAKTIKUM(#V_Matrikelnr_1),

T-ZUSATZ(#V_Matrikelnr_1)
->
K-SOFTSKILL(#V_Matrikelnr_1)]

{T-STUDIERENDE} = [
T-STUDIERENDE(#V_Matrikelnr_1, #V_Nachname_1, #V_Vorname_1, #V_Studiengang_1,
        #V_Abschluss_1)
->
K-STUDIERENDE(#V_Matrikelnr_1, #V_Nachname_1, #V_Vorname_1, #V_Abschluss_1)]


***Start calculation extended inverse mapping...***

-Disjunctive sets after extension:

{T-NOTEN} = [
T-NOTEN(#V_Modulnr_1, #V_Matrikelnr_1, #V_Semester_1, #V_Note_1,
        "Von Kiel importiert")
->
K-NOTEN+(#V_Modulnr_1, #V_Matrikelnr_1, #V_Semester_1, #V_Note_1, #E_Bemerkung_1)]

{T-ZUSATZ} = [
T-ZUSATZ(#V_Matrikelnr_1)
->
K-NOTEN+(#E_Modulnr_1, #V_Matrikelnr_1, #E_Semester_1, #E_Note_1, #E_Bemerkung_1),

T-ZUSATZ(#V_Matrikelnr_1)
->
K-PRAKTIKUM+(#V_Matrikelnr_1),

T-ZUSATZ(#V_Matrikelnr_1)
->
K-SOFTSKILL+(#V_Matrikelnr_1)]

{T-STUDIERENDE} = [
T-STUDIERENDE(#V_Matrikelnr_1, #V_Nachname_1, #V_Vorname_1, #V_Studiengang_1,
        #V_Abschluss_1)
->
K-STUDIERENDE+(#V_Matrikelnr_1, #V_Nachname_1, #V_Vorname_1, #V_Abschluss_1,
        #V_Studiengang_1)]

-Extended source schema:
K-STUDIERENDE {Matrikelnr=int, Nachname=string, Vorname=string,
        Abschluss=string}
K-SOFTSKILL {Matrikelnr=int}
K-NOTEN {Modulnr=int, Matrikelnr=int, Semester=string, Note=double,
```

```
        Bemerkung=string}
K_PRAKTIKUM+ {Matrikelnr=int}
K_SOFTSKILL+ {Matrikelnr=int}
K_NOTEN+ {Modulnr=int, Matrikelnr=int, Semester=string, Note=double,
        Bemerkung=string}
K_STUDIERENDE+ {Matrikelnr=int, Nachname=string, Vorname=string,
        Abschluss=string, Studiengang=string}
K_PRAKTIKUM {Matrikelnr=int}


***Run disjunctive chase with extended inverse schema mapping...***

-Result instance set from adapted disjunctive chase:
K_NOTEN+(2, 2, "SS 21", 4.0, #N_Bemerkung_5)
K_NOTEN+(2, 3, #N_semester_1, 2.0, #N_Bemerkung_4)
K_NOTEN+(#N_Modulnr_1, 1, #N_Semester_2, #N_Note_1, #N_Bemerkung_6)
K_STUDIERENDE+(2, "Sonnensand", #N_firstname_1, "Bachelor", #N_Studiengang_2)
K_STUDIERENDE+(2, "Sonne", "Sarah", #N_Abschluss_2, "ITTI")
K_STUDIERENDE+(1, "Fieber", "Fabian", #N_Abschluss_1, "Informatik")
K_STUDIERENDE+(3, #N_lastname_1, "Max", "Master", #N_Studiengang_1)

K_NOTEN+(2, 2, "SS 21", 4.0, #N_Bemerkung_5)
K_NOTEN+(2, 3, #N_semester_1, 2.0, #N_Bemerkung_4)
K_PRAKTIKUM+(1)
K_PRAKTIKUM+(2)
K_PRAKTIKUM+(3)
K_STUDIERENDE+(2, "Sonnensand", #N_firstname_1, "Bachelor", #N_Studiengang_2)
K_STUDIERENDE+(2, "Sonne", "Sarah", #N_Abschluss_2, "ITTI")
K_STUDIERENDE+(1, "Fieber", "Fabian", #N_Abschluss_1, "Informatik")
K_STUDIERENDE+(3, #N_lastname_1, "Max", "Master", #N_Studiengang_1)

K_NOTEN+(2, 2, "SS 21", 4.0, #N_Bemerkung_5)
K_NOTEN+(2, 3, #N_semester_1, 2.0, #N_Bemerkung_4)
K_SOFTSKILL+(2)
K_SOFTSKILL+(3)
K_SOFTSKILL+(1)
K_STUDIERENDE+(2, "Sonnensand", #N_firstname_1, "Bachelor", #N_Studiengang_2)
K_STUDIERENDE+(2, "Sonne", "Sarah", #N_Abschluss_2, "ITTI")
K_STUDIERENDE+(1, "Fieber", "Fabian", #N_Abschluss_1, "Informatik")
K_STUDIERENDE+(3, #N_lastname_1, "Max", "Master", #N_Studiengang_1)


***Calculate safe instance from instance set...***

-Save instance:
K_NOTEN+(2, 2, "SS 21", 4.0, #N_Bemerkung_4)
K_NOTEN+(2, 3, #N_semester_5, 2.0, #N_Bemerkung_7)
K_STUDIERENDE+(2, "Sonne", "Sarah", #N_Abschluss_3, "ITTI")
K_STUDIERENDE+(1, "Fieber", "Fabian", #N_Abschluss_4, "Informatik")
K_STUDIERENDE+(2, "Sonnensand", #N_firstname_4, "Bachelor", #N_Studiengang_3)
K_STUDIERENDE+(3, #N_lastname_4, "Max", "Master", #N_Studiengang_4)

-Extended source instance:
```

```
K_NOTEN(2, 3, #N_semester_1, 2.0, #N_Bemerkung_1)
K_NOTEN(2, 2, "SS 21", 4.0, "Knapp")
K_NOTEN+(2, 2, "SS 21", 4.0, #N_Bemerkung_4)
K_NOTEN+(2, 3, #N_semester_5, 2.0, #N_Bemerkung_7)
K_PRAKTIKUM(3)
K_SOFTSKILL(2)
K_SOFTSKILL(3)
K_STUDIERENDE(2, "Sonnensand", #N_firstname_1, "Bachelor")
K_STUDIERENDE(3, #N_lastname_1, "Max", "Master")
K_STUDIERENDE+(2, "Sonne", "Sarah", #N_Abschluss_3, "ITTI")
K_STUDIERENDE+(3, #N_lastname_4, "Max", "Master", #N_Studiengang_4)
K_STUDIERENDE+(1, "Fieber", "Fabian", #N_Abschluss_4, "Informatik")
K_STUDIERENDE+(2, "Sonnensand", #N_firstname_4, "Bachelor", #N_Studiengang_3)


***Calculate follow-up Tgds and bEgds...***

-Follow-up constrains:
[
K_PRAKTIKUM+(#V_Matrikelnr_1)
->
K_PRAKTIKUM(#V_Matrikelnr_1),

K_SOFTSKILL+(#V_Matrikelnr_1)
->
K_SOFTSKILL(#V_Matrikelnr_1),

K_NOTEN+(#V_Modulnr_1, #V_Matrikelnr_1, #V_Semester_1, #V_Note_1, #V_Bemerkung_1)
->
K_NOTEN(#V_Modulnr_1, #V_Matrikelnr_1, #V_Semester_1, #V_Note_1, #V_Bemerkung_1),

K_STUDIERENDE+(#V_Matrikelnr_1, #V_Nachname_1, #V_Vorname_1, #V_Abschluss_1,
        #V_Studiengang_1)
->
K_STUDIERENDE(#V_Matrikelnr_1, #V_Nachname_1, #V_Vorname_1, #V_Abschluss_1)]
[
K_NOTEN(#V_Modulnr_1, #V_Matrikelnr_1, #V_Semester_1, #V_Note_1, #V_Bemerkung_1),
K_NOTEN(#V_Modulnr_1, #V_Matrikelnr_1, #V_Semester_2, #V_Note_2, #V_Bemerkung_2)
->
#V_Note_1 = #V_Note_2,
#V_Semester_1 = #V_Semester_2,
#V_Bemerkung_1 = #V_Bemerkung_2,

K_STUDIERENDE(#V_Matrikelnr_1, #V_Nachname_1, #V_Vorname_1, #V_Abschluss_1),
K_STUDIERENDE(#V_Matrikelnr_1, #V_Nachname_2, #V_Vorname_2, #V_Abschluss_2)
->
#V_Nachname_1 = #V_Nachname_2,
#V_Vorname_1 = #V_Vorname_2,
#V_Abschluss_1 = #V_Abschluss_2]

-Run chase with follow-up Tgds...

-Result from follow-up Tgds...
```

```
K_NOTEN(2, 2, "SS 21", 4.0, #N_Bemerkung_4)
K_NOTEN(2, 3, #N_semester_1, 2.0, #N_Bemerkung_1)
K_NOTEN(2, 2, "SS 21", 4.0, "Knapp")
K_NOTEN(2, 3, #N_semester_5, 2.0, #N_Bemerkung_7)
K_NOTEN+(2, 2, "SS 21", 4.0, #N_Bemerkung_4)
K_NOTEN+(2, 3, #N_semester_5, 2.0, #N_Bemerkung_7)
K_PRAKTIKUM(3)
K_SOFTSKILL(2)
K_SOFTSKILL(3)
K_STUDIERENDE(3, #N_lastname_4, "Max", "Master")
K_STUDIERENDE(2, "Sonnensand", #N_firstname_4, "Bachelor")
K_STUDIERENDE(2, "Sonnensand", #N_firstname_1, "Bachelor")
K_STUDIERENDE(2, "Sonne", "Sarah", #N_Abschluss_3)
K_STUDIERENDE(1, "Fieber", "Fabian", #N_Abschluss_4)
K_STUDIERENDE(3, #N_lastname_1, "Max", "Master")
K_STUDIERENDE+(2, "Sonne", "Sarah", #N_Abschluss_3, "ITTI")
K_STUDIERENDE+(3, #N_lastname_4, "Max", "Master", #N_Studiengang_4)
K_STUDIERENDE+(1, "Fieber", "Fabian", #N_Abschluss_4, "Informatik")
K_STUDIERENDE+(2, "Sonnensand", #N_firstname_4, "Bachelor", #N_Studiengang_3)


-Run chase with follow-up bEgds...

******Galve result:******
K_NOTEN(2, 3, #N_semester_1, 2.0, #N_Bemerkung_1)
K_NOTEN(2, 2, "SS 21", 4.0, "Knapp")
K_NOTEN+(2, 3, #N_semester_1, 2.0, #N_Bemerkung_1)
K_NOTEN+(2, 2, "SS 21", 4.0, "Knapp")
K_PRAKTIKUM(3)
K_SOFTSKILL(2)
K_SOFTSKILL(3)
K_STUDIERENDE(1, "Fieber", "Fabian", #N_Abschluss_4)
K_STUDIERENDE(3, #N_lastname_1, "Max", "Master")
K_STUDIERENDE(2, "Sonnensand", "Sarah", "Bachelor")
K_STUDIERENDE+(2, "Sonne", "Sarah", "Bachelor", "ITTI")
K_STUDIERENDE+(1, "Fieber", "Fabian", #N_Abschluss_4, "Informatik")
K_STUDIERENDE+(2, "Sonnensand", "Sarah", "Bachelor", #N_Studiengang_3)
K_STUDIERENDE+(3, #N_lastname_1, "Max", "Master", #N_Studiengang_4)
```

B. Laufende Minibeispiele in ChaTEAU

Beispiel zur Invertierung (siehe Abschnitte 3.2.4 und 4.2.1)

```
Read Input_1 from File: /MiniBeispiel1.xml

***Start data integration...***

-Input_1:
A(1, 2)
B(1)
C(1, 2, 2, 3)

-Constraints_1:
[A(#V_x_1, #V_x_2),
B(#V_x_1)
->
Q(#V_x_1, #V_x_2, #V_x_1, #V_x_1),

C(#V_x_1, #V_x_2, #V_x_2, #V_x_3)
->
Q(#V_x_1, #V_x_2, #V_x_2, #E_y_1),
R(#E_y_1, #V_x_2)]

-Output1:
Q(1, 2, 1, 1)
Q(1, 2, 2, #N_q4_1)
R(#N_q4_1, 2)

-Global instance after integration:
Q(1, 2, 1, 1)
Q(1, 2, 2, #N_q4_1)
R(#N_q4_1, 2)

-Generated global scheme:
Q {q1=int, q2=int, q3=int, q4=int}
R {r1=int, r2=int}


******Start GaLVE for first input...******

***Start calculation inverse mapping...***

-Disjunctive sets in inverse mapping:

{Q} = [
Q(#V_x_1, #V_x_2, #V_x_1, #V_x_1)
```

```
->
A(#V_x_1, #V_x_2),
B(#V_x_1),

Q(#V_x_1, #V_x_2, #V_x_2, #V_y_1)
->
C(#V_x_1, #V_x_2, #V_x_2, #E_x_3)]

{R} = [
R(#V_y_1, #V_x_2)
->
C(#E_x_1, #V_x_2, #V_x_2, #E_x_3)]


***Start calculation optimized inverse mapping...***

-Disjunctive sets after optimization:

{Q} = [
Q(#V_x_1, #V_x_2, #V_x_1, #V_x_1)
->
A(#V_x_1, #V_x_2),
B(#V_x_1),

Q(#V_x_1, #V_x_2, #V_x_2, #V_y_1)
->
C(#V_x_1, #V_x_2, #V_x_2, #E_x_3)]

{R} = [
R(#V_y_1, #V_x_2)
->
C(#E_x_1, #V_x_2, #V_x_2, #E_x_3)]


***Start calculation extended inverse mapping...***

-Disjunctive sets after extension:

{Q} = [
Q(#V_x_1, #V_x_2, #V_x_1, #V_x_1)
->
A+(#V_x_1, #V_x_2, #V_x_1),
B+(#V_x_1, #V_x_1),

Q(#V_x_1, #V_x_2, #V_x_2, #V_y_1)
->
C+(#V_x_1, #V_x_2, #V_x_2, #E_x_3, #V_y_1, #E_y_1)]

{R} = [
R(#V_y_1, #V_x_2)
->
C+(#E_x_1, #V_x_2, #V_x_2, #E_x_3, #E_y_1, #V_y_1)]
```

```
-Extended source schema:
A {a1=int, a2=int}
B {b1=int}
C {c1=int, c2=int, c3=int, c4=int}
C+ {c1=int, c2=int, c3=int, c4=int, q4=int, r1=int}
B+ {b1=int, q4=int}
A+ {a1=int, a2=int, q4=int}


***Run disjunctive chase with extended inverse schema mapping...***

-Result instance set from adapted disjunctive chase:
A+(1, 2, 1)
B+(1, 1)
C+(#N_c1_1, 2, 2, #N_c4_1, #N_q4_2, #N_q4_1)

C+(1, 2, 2, #N_c4_1, #N_q4_1, #N_r1_2)
C+(#N_c1_1, 2, 2, #N_c4_2, #N_q4_2, #N_q4_1)


***Calculate safe instance from instance set...***

-Save instance:
C+(#N_c1_2, 2, 2, #N_c4_2, #N_q4_3, #N_q4_3)

-Extended source instance:
A(1, 2)
B(1)
C(1, 2, 2, 3)
C+(#N_c1_2, 2, 2, #N_c4_2, #N_q4_3, #N_q4_3)


***Calculate follow-up Tgds and bEgds...***

-Follow-up constrains:
[

C+(#V_c1_1, #V_c2_1, #V_c3_1, #V_c4_1, #V_q4_1, #V_r1_1)
->
C(#V_c1_1, #V_c2_1, #V_c3_1, #V_c4_1),

B+(#V_b1_1, #V_q4_1)
->
B(#V_b1_1),

A+(#V_a1_1, #V_a2_1, #V_q4_1)
->
A(#V_a1_1, #V_a2_1)]
[
C(#V_c1_2, #V_c2_2, #V_c3_2, #V_c4_2),
C(#V_c1_1, #V_c2_1, #V_c3_1, #V_c4_1)
->
#V_c2_1 = #V_c2_2,
```

```
#V_c4_1 = #V_c4_2,
#V_c3_1 = #V_c3_2,
#V_c1_1 = #V_c1_2,

B(#V_b1_1),
B(#V_b1_2)
->
#V_b1_1 = #V_b1_2,

A(#V_a1_1, #V_a2_1),
A(#V_a1_2, #V_a2_2)
->
#V_a2_1 = #V_a2_2,
#V_a1_1 = #V_a1_2]

-Run chase with follow-up Tgds...

-Result from follow-up Tgds...
A(1, 2)
B(1)
C(1, 2, 2, 3)
C(#N_c1_2, 2, 2, #N_c4_2)
C+(#N_c1_2, 2, 2, #N_c4_2, #N_q4_3, #N_q4_3)


-Run chase with follow-up bEgds...

******Galve result:******
A(1, 2)
B(1)
C(1, 2, 2, 3)
C+(1, 2, 2, 3, #N_q4_3, #N_q4_3)
```

Interessante Optimierung

```
Read Input_1 from File: /MiniBeispiel2.xml

***Start data integration...***

-Input_1:
A("a")
B("b")

-Constraints_1:
[
A(#V_x_1)
->
R(#V_x_1),

B(#V_x_1)
->
R(#V_x_1)]

-Output1:
R("b")
R("a")

-Global instance after integration:
R("b")
R("a")

-Generated global scheme:
R {r=string}


******Start GaLVE for first input...******

***Start calculation inverse mapping...***

-Disjunctive sets in inverse mapping:

{R} = [
R(#V_x_1)
->
A(#V_x_1),

R(#V_x_1)
->
B(#V_x_1)]


***Start calculation optimized inverse mapping...***

-Disjunctive sets after optimization:


***Start calculation extended inverse mapping...***
```

```
-Disjunctive sets after extension:

-Extended source schema:
A {a=string}
B {b=string}


***Run disjunctive chase with extended inverse schema mapping...***

-Result instance set from adapted disjunctive chase:


***Calculate safe instance from instance set...***

-Save instance:


-Extended source instance:
A("a")
B("b")


***Calculate follow-up Tgds and bEgds...***

-Follow-up constrains:
[]
[]

-Run chase with follow-up Tgds...

-Result from follow-up Tgds...
A("a")
B("b")

-Run chase with follow-up bEgds...

******Galve result:******
A("a")
B("b")
```

Interessante Dekomposition

```
Read Input_1 from File: /MiniBeispiel3.xml

***Start data integration...***

-Input_1:
A(1, 2, 3)
A(4, 5, 6)

-Constraints_1:
[
A(#V_x_1, #V_x_2, #V_x_3)
->
Q(#V_x_1, #V_x_2),
R(#V_x_2, #V_x_3)]

-Output1:
Q(1, 2)
Q(4, 5)
R(2, 3)
R(5, 6)

-Global instance after integration:
Q(1, 2)
Q(4, 5)
R(2, 3)
R(5, 6)

-Generated global scheme:
Q {q1=int, q2=int}
R {r1=int, r2=int}


******Start GaLVE for first input...******

***Start calculation inverse mapping...***

-Disjunctive sets in inverse mapping:

{Q} = [
Q(#V_x_1, #V_x_2)
->
A(#V_x_1, #V_x_2, #E_x_3)]

{R} = [
R(#V_x_2, #V_x_3)
->
A(#E_x_1, #V_x_2, #V_x_3)]


***Start calculation optimized inverse mapping...***

-Disjunctive sets after optimization:
```

```
{Q} = [
Q(#V_x_1, #V_x_2)
->
A(#V_x_1, #V_x_2, #E_x_3)]

{R} = [
R(#V_x_2, #V_x_3)
->
A(#E_x_1, #V_x_2, #V_x_3)]


***Start calculation extended inverse mapping...***

-Disjunctive sets after extension:

{Q} = [
Q(#V_x_1, #V_x_2)
->
A+(#V_x_1, #V_x_2, #E_x_3)]

{R} = [
R(#V_x_2, #V_x_3)
->
A+(#E_x_1, #V_x_2, #V_x_3)]

-Extended source schema:
A {a1=int, a2=int, a3=int}
A+ {a1=int, a2=int, a3=int}


***Run disjunctive chase with extended inverse schema mapping...***

-Result instance set from adapted disjunctive chase:
A+(#N_a1_2, 2, 3)
A+(#N_a1_1, 5, 6)
A+(4, 5, #N_a3_1)
A+(1, 2, #N_a3_2)


***Calculate safe instance from instance set...***

-Save instance:
A+(#N_a1_2, 2, 3)
A+(#N_a1_1, 5, 6)
A+(4, 5, #N_a3_1)
A+(1, 2, #N_a3_2)

-Extended source instance:
A(1, 2, 3)
A(4, 5, 6)
A+(#N_a1_2, 2, 3)
A+(#N_a1_1, 5, 6)
```

```
A+(4, 5, #N_a3_1)
A+(1, 2, #N_a3_2)


***Calculate follow-up Tgds and bEgds...***

-Follow-up constrains:
[
A+(#V_a1_1, #V_a2_1, #V_a3_1)
->
A(#V_a1_1, #V_a2_1, #V_a3_1)]
[
A(#V_a1_1, #V_a2_1, #V_a3_1),
A(#V_a1_1, #V_a2_2, #V_a3_2)
->
#V_a3_1 = #V_a3_2,
#V_a2_1 = #V_a2_2]

-Run chase with follow-up Tgds...

-Result from follow-up Tgds...
A(#N_a1_1, 5, 6)
A(4, 5, #N_a3_1)
A(1, 2, 3)
A(4, 5, 6)
A(#N_a1_2, 2, 3)
A(1, 2, #N_a3_2)
A+(#N_a1_2, 2, 3)
A+(#N_a1_1, 5, 6)
A+(4, 5, #N_a3_1)
A+(1, 2, #N_a3_2)


-Run chase with follow-up bEgds...

******Galve result:******
A(#N_a1_1, 5, 6)
A(1, 2, 3)
A(4, 5, 6)
A(#N_a1_2, 2, 3)
A+(4, 5, 6)
A+(#N_a1_2, 2, 3)
A+(#N_a1_1, 5, 6)
A+(1, 2, 3)
```

Interessante Projektion und Selektion

```
Read Input_1 from File: /MiniBeispiel4.xml

***Start data integration...***

-Input_1:
A(3, 3)
A(2, 2)

-Constraints_1:
[
A(#V_x_1, 3)
->
Q(#V_x_1)]

-Output1:
Q(3)

-Global instance after integration:
Q(3)

-Generated global scheme:
Q {q1=int}


******Start GaLVE for first input...******

***Start calculation inverse mapping...***

-Disjunctive sets in inverse mapping:

{Q} = [
Q(#V_x_1)
->
A(#V_x_1, 3)]


***Start calculation optimized inverse mapping...***

-Disjunctive sets after optimization:

{Q} = [
Q(#V_x_1)
->
A(#V_x_1, 3)]


***Start calculation extended inverse mapping...***

-Disjunctive sets after extension:

{Q} = [
Q(#V_x_1)
```

```
->
A+(#V_x_1, 3)]

-Extended source schema:
A {a1=int, a2=int}
A+ {a1=int, a2=int}


***Run disjunctive chase with extended inverse schema mapping...***

-Result instance set from adapted disjunctive chase:
A+(3, 3)


***Calculate safe instance from instance set...***

-Save instance:
A+(3, 3)

-Extended source instance:
A(3, 3)
A(2, 2)
A+(3, 3)


***Calculate follow-up Tgds and bEgds...***

-Follow-up constrains:
[
A+(#V_a1_1, #V_a2_1)
->
A(#V_a1_1, #V_a2_1)]
[
A(#V_a1_1, #V_a2_2),
A(#V_a1_1, #V_a2_1)
->
#V_a2_1 = #V_a2_2]

-Run chase with follow-up Tgds...

-Result from follow-up Tgds...
A(3, 3)
A(2, 2)
A+(3, 3)


-Run chase with follow-up bEgds...

******Galve result:******
A(3, 3)
A(2, 2)
A+(3, 3)
```

Gegenbeispiel Quasi-Inverse (siehe Abschnitt 3.2.2)

```
Read Input_1 from File: /MiniBeispiel5.xml

***Start data integration...***

-Input_1:
A(1, 3)
A(3, 4)
A(6, 6)

-Constraints_1:
[
A(#V_x_1, #V_x_3),
A(#V_x_3, #V_x_2)
->
Q(#V_x_1, #V_x_2),
R(#V_x_3)]

-Output1:
Q(1, 4)
R(3)

-Global instance after integration:
Q(1, 4)
R(3)

-Generated global scheme:
Q {q1=int, q2=int}
R {r1=int}


******Start GaLVE for first input...******

***Start calculation inverse mapping...***

-Disjunctive sets in inverse mapping:

{Q} = [
Q(#V_x_1, #V_x_2)
->
A(#E_x_3, #V_x_2),
A(#V_x_1, #E_x_3)]

{R} = [
R(#V_x_3)
->
A(#E_x_1, #V_x_3),
A(#V_x_3, #E_x_2)]


***Start calculation optimized inverse mapping...***

-Disjunctive sets after optimization:
```

```
***Start calculation extended inverse mapping...***

-Disjunctive sets after extension:

-Extended source schema:
A {a1=int, a2=int, a3=int}


***Run disjunctive chase with extended inverse schema mapping...***

-Result instance set from adapted disjunctive chase:



***Calculate safe instance from instance set...***

-Save instance:


-Extended source instance:
A(1, 3)
A(3, 4)
A(6, 6)


***Calculate follow-up Tgds and bEgds...***

-Follow-up constrains:
[]
[]

-Run chase with follow-up Tgds...

-Result from follow-up Tgds...
A(1, 3)
A(3, 4)
A(6, 6)


-Run chase with follow-up bEgds...

******Galve result:******
A(1, 3)
A(3, 4)
A(6, 6)
```

Interessante Sichere Instanz

```
Read Input_1 from File: /MiniBeispiel6.xml

***Start data integration...***

-Input_1:
A(2)
A(1)
B(1)
B(3)

-Constraints_1:
[
A(#V_x_1)
->
Q(#V_x_1, #V_x_1),

B(#V_x_1)
->
Q(#V_x_1, #E_y_1),

B(#V_x_1),
A(#V_x_1)
->
R(#V_x_1)]

-Output1:
Q(1, 1)
Q(2, 2)
Q(3, #N_q2_1)
R(1)

-Global instance after integration:
Q(1, 1)
Q(2, 2)
Q(3, #N_q2_1)
R(1)

-Generated global scheme:
Q {q1=int, q2=int}
R {r1=int}


******Start GaLVE for first input...******

***Start calculation inverse mapping...***

-Disjunctive sets in inverse mapping:

{Q} = [
Q(#V_x_1, #V_x_1)
->
A(#V_x_1),
```

```
Q(#V_x_1, #V_y_1)
->
B(#V_x_1)]

{R} = [
R(#V_x_1)
->
B(#V_x_1),
A(#V_x_1)]


***Start calculation optimized inverse mapping...***

-Disjunctive sets after optimization:

{Q} = [
Q(#V_x_1, #V_x_1)
->
A(#V_x_1),

Q(#V_x_1, #V_y_1)
->
B(#V_x_1)]

{R} = [
R(#V_x_1)
->
B(#V_x_1),
A(#V_x_1)]


***Start calculation extended inverse mapping...***

-Disjunctive sets after extension:

{Q} = [
Q(#V_x_1, #V_x_1)
->
A+(#V_x_1, #V_x_1),

Q(#V_x_1, #V_y_1)
->
B+(#V_x_1, #V_y_1)]

{R} = [
R(#V_x_1)
->
A+(#V_x_1, #E_x_1),
B+(#V_x_1, #E_y_1)]

-Extended source schema:
A {a1=int}
```

```
B {b1=int}
B+ {b1=int, q2=int}
A+ {a1=int, q2=int}


***Run disjunctive chase with extended inverse schema mapping...***

-Result instance set from adapted disjunctive chase:
A+(1, #N_q2_2)
A+(1, 1)
A+(2, 2)
B+(1, #N_q2_3)

A+(1, #N_q2_2)
B+(3, #N_q2_1)
B+(1, #N_q2_3)
B+(2, 2)
B+(1, 1)


***Calculate safe instance from instance set...***

-Save instance:
A+(1, 1)
B+(1, #N_q2_4)

-Extended source instance:
A(2)
A(1)
A+(1, 1)
B(1)
B(3)
B+(1, #N_q2_4)


***Calculate follow-up Tgds and bEgds...***

-Follow-up constrains:
[
B+(#V_b1_1, #V_q2_1)
->
B(#V_b1_1),

A+(#V_a1_1, #V_q2_1)
->
A(#V_a1_1)]
[]

-Run chase with follow-up Tgds...

-Result from follow-up Tgds...
A(2)
A(1)
```

```
A+(1, 1)
B(1)
B(3)
B+(1, #N_q2_4)


-Run chase with follow-up bEgds...

******Galve result:******
A(2)
A(1)
A+(1, 1)
B(1)
B(3)
B+(1, #N_q2_4)
```

C. Programmcode GaLVE in ChaTEAU

GalveMain.java

```
1  package galve;
2
3  import java.io.File;
4  import java.io.IOException;
5  import java.util.ArrayList;
6  import java.util.HashMap;
7  import java.util.HashSet;
8  import java.util.LinkedHashMap;
9
10 import org.jdom2.JDOMException;
11
12 import instance.Instance;
13 import instance.SchemaTag;
14 import io.InputReader;
15 import io.SingleInput;
16 /**
17 * implementation of galve technique according to master thesis:
18 *  "Datenintegration durch inverse Schemaabbildung:
19 *       Erweiterung der Rostocker GaLVE-Technik"
20 *
21 *  console program accepts any number of file paths and calculates the
22 * schema mapping of the input.
23 *  the global schemas of the input schema mappings should be the
24 *  same.
25 *  for the calculation of the begds it is important to specify key
26 *  attributes.
27 *  the galve procedure will start for the first input
28 *
29 *  the galve techniques described in the master thesis are implemented
30 *  in GalveTechniques.java.
31 *  GalveUtil.java contains outsourced functionality of galve techniques.
32 *  the chases with begds is implemented in the applyBEgd method in
33 *  Chase.java.
34 *
35 * @author Jakob Zimmer
36 *
37 */
38 public class MainGaLVE {
39 public static void main(String[] args) {
40
41     // read input files
42     InputReader reader = new InputReader();
```

```
43      var inputs = new ArrayList<SingleInput>();
44
45      int i = 1;
46      for (String arg : args) {
47
48          File file = new File(arg);
49          if (file.exists() && !file.isDirectory()) {
50              System.out.println("Read Input_"+ i++ + " from File: " +
51               file.getAbsolutePath() + "\n");
52              SingleInput input;
53              try {
54                  input = reader.readFile(file);
55              } catch (JDOMException | IOException e) {
56                  e.printStackTrace();
57                  return;
58              }
59              inputs.add(input);
60          }
61      }
62      if(inputs.size() < 1) {
63          System.out.println("-Empty input...");
64      }else {
65          // get key attributes from first instance
66          var relationKeyAttributes = inputs.stream().findFirst().get().
67              getInstance().getRelationKeyAttributes();
68
69          // data integration
70          System.out.println("***Start data integration...***\n");
71          var globalInstance = GalveTechniques.dataIntegration(inputs);
72
73          System.out.println("-Global instance after integration:");
74          System.out.println(globalInstance);
75
76          System.out.println("-Generated global scheme:");
77          globalInstance.getSchema().entrySet().
78          forEach(entry -> System.out.println(entry.getKey() + " " +
79           entry.getValue()));
80
81          // calculate inverse
82          System.out.println("\n\n******Start GaLVE for first
83           input...******\n");
84          System.out.println("***Start calculation inverse mapping...***");
85
86          var source = inputs.get(0).getInstance();
87          var constaints = inputs.get(0).getConstraints();
88
89          var sourceSchema = GalveUtil.getSchemaForSchemaTag(source,
90           SchemaTag.SOURCE);
91          var targetSchema = GalveUtil.getSchemaForSchemaTag(
92              source, SchemaTag.TARGET);
93
94          var schemaMapping = new SchemaMapping(sourceSchema,
95           targetSchema, constaints);
```

```
96
97          var inverseMapping = GalveTechniques.invert(schemaMapping);
98
99          System.out.println("\n-Disjunctive sets in inverse mapping:");
100         inverseMapping.getDisjunctiveSets().entrySet().
101         forEach(entry -> System.out.println("\n{" + entry.getKey() + "} = "
102     + entry.getValue()));
103
104         // optimize
105         System.out.println("\n\n***Start calculation optimized inverse
106          mapping...***\n");
107         GalveTechniques.optimizeDisjunctiveSets(
108             inverseMapping.getDisjunctiveSets());
109
110         System.out.println("-Disjunctive sets after optimization:");
111         inverseMapping.getDisjunctiveSets().entrySet().
112         forEach(entry -> System.out.println("\n{" + entry.getKey() + "} = "
113             + entry.getValue()));
114
115         // extend
116         System.out.println("\n\n***Start calculation extended inverse
117          mapping...***");
118         GalveTechniques.extend(inverseMapping);
119
120         System.out.println("\n-Disjunctive sets after extension:");
121         inverseMapping.getDisjunctiveSets().entrySet().
122         forEach(entry -> System.out.println("\n{" + entry.getKey() + "} = "
123             + entry.getValue()));
124
125         System.out.println("\n-Extended source schema:");
126         var extendedSchema = inverseMapping.getTargetSchema();
127         extendedSchema.entrySet().
128         forEach(entry -> System.out.println(entry.getKey() + " " +
129          entry.getValue()));
130
131         // run adapted disjunctive chase with extended inverse schema
132         // mapping
133         System.out.println("\n***Run disjunctive chase with extended
134             inverse schema mapping...***");
135         var unionSchema = new HashMap<String,
136         LinkedHashMap<String, String>>();
137         unionSchema.putAll(extendedSchema);
138         unionSchema.putAll(targetSchema);
139
140         var resultInstance = new Instance(
141             globalInstance.getRelationalAtoms(),
142          unionSchema, globalInstance.getOriginTag(),
143              globalInstance.getSchemaTags());
144         var instanceSet = GalveTechniques.runAdaptedDisjunvtiveChase(
145             resultInstance, inverseMapping);
146
147         System.out.println("\n-Result instance set from adapted
148             disjunctive chase:");
```

```
149         instanceSet.forEach(instance -> System.out.println(instance));
150
151         // safe instance
152         System.out.println("\n\n***Calculate safe instance from
153             instance set...***");
154         var safeInstance = GalveTechniques.safeInstance(
155             instanceSet);
156
157         System.out.println("\n-Save instance:");
158         System.out.println(safeInstance);
159
160         // extend source with safe instance
161         var newNullHashSet = new HashSet<Instance>();
162         newNullHashSet.add(source);
163         newNullHashSet.add(safeInstance);
164         GalveUtil.makeDisjunctNulls(newNullHashSet);
165
166         safeInstance.getRelationalAtoms().forEach(atom ->
167             atom.setOrigin(false));
168         source.getRelationalAtoms().forEach(atom ->
169             atom.setOrigin(true));
170         safeInstance.getRelationalAtoms().forEach(atom ->
171          source.getRelationalAtoms().add(atom));
172
173         System.out.println("-Extended source instance:");
174         System.out.println(source);
175
176         // calculate follow-up constrains
177         System.out.println("\n***Calculate follow-up Tgds
178             and bEgds...***");
179         var followUpTgds = GalveTechniques.followUpTgds(
180             source.getSchema(), sourceSchema,
181             relationKeyAttributes);
182         var followUpBEgds = GalveTechniques.followUpBEgds(
183             source.getSchema(), sourceSchema, relationKeyAttributes);
184
185         System.out.println("\n-Follow-up constrains:");
186         System.out.println(followUpTgds);
187         System.out.println(followUpBEgds);
188
189         // run chase with follow-up constrains
190         System.out.println("\n-Run chase with follow-up Tgds...");
191
192         // adjust schema of the source instance
193         sourceSchema.entrySet().forEach(entry ->
194          source.getSchema().put(entry.getKey(), entry.getValue()));
195         source.getSchemaTags().clear();
196
197         var singleInputForTgdResult = new SingleInput(
198             source, followUpTgds);
199         var resultForTgds = GalveUtil.runChase(singleInputForTgdResult);
200
201         System.out.println("\n-Result from follow-up Tgds...");
```

```
202         System.out.println(resultForTgds);
203
204         System.out.println("\n-Run chase with follow-up bEgds...");
205
206         // run chase as many times as it takes to equate all constants
207         var result = new Instance(OriginTag.INSTANCE);
208         var tempResult = new Instance(OriginTag.INSTANCE);
209         do {
210             tempResult = new Instance(result);
211             var singleInputForBEgdResult = new SingleInput(
212                 resultForTgds, followUpBEgds);
213             result = GalveUtil.runChase(singleInputForBEgdResult);
214         }while(tempResult.equals(result));
215
216         System.out.println("\n******Galve result:******");
217         System.out.println(result);
218
219     }
220 }
221 }
```

GalveTechniques.java

```
1  package galve;
2
3  import java.util.ArrayList;
4  import java.util.Collections;
5  import java.util.HashMap;
6  import java.util.HashSet;
7  import java.util.LinkedHashMap;
8  import java.util.LinkedHashSet;
9  import java.util.Map.Entry;
10
11 import atom.Atom;
12 import atom.EqualityAtom;
13 import atom.RelationalAtom;
14 import homomorphism.TermMapping;
15 import instance.Instance;
16 import instance.OriginTag;
17 import instance.SchemaTag;
18 import integrityConstraint.IntegrityConstraint;
19 import integrityConstraint.STTgd;
20 import integrityConstraint.Tgd;
21 import io.SingleInput;
22 import term.Variable;
23 import term.VariableType;
24
25 /**
26 * implementation of galve technique according to master thesis:
27 *  "Datenintegration durch inverse Schemaabbildung:
28 *       Erweiterung der Rostocker GaLVE-Technik"
29 *
30 * @author Jakob Zimmer
31 *
32 */
33 public final class GalveTechniques{
34
35 /**
36 * executes the schema mappings
37 * combines the results with disjoint null values
38 * @param inputs
39 * @return global instance
40 */
41 static Instance dataIntegration(ArrayList<SingleInput> inputs) {
42
43     var globalInstances = new HashSet<Instance>();
44     var globalInstance = new Instance(OriginTag.INSTANCE);
45
46     for (SingleInput input : inputs) {
47         if(!input.getInstance().getOriginTag().equals(
48             OriginTag.INSTANCE)) {
49             System.out.println("GaLVE is defined only for instances.
50                 Input is skipped.\n");
51         }
```

```
52          if(!input.getConstraints().stream().allMatch(constraint ->
53              constraint instanceof STTgd)) {
54              System.out.println("GaLVE is defined only for S-T TGDs.
55                  Input is skipped.\n");
56          }
57
58          System.out.println("-Input_" + (inputs.indexOf(input) + 1) + ":");
59          System.out.println(input.getInstance());
60
61          System.out.println("-Constraints_" +
62              (inputs.indexOf(input) + 1) + ":");
63          System.out.println(input.getConstraints());
64
65          // fill global scheme
66          globalInstance.getSchema().putAll(GalveUtil.
67              getSchemaForSchemaTag(input.getInstance(),
68               SchemaTag.TARGET));
69
70          var resultInstance = GalveUtil.runChase(input);
71
72          System.out.println("\n-Output" + (inputs.indexOf(input) + 1) + ":");
73          System.out.println(resultInstance);
74
75          globalInstances.add(resultInstance);
76      }
77      // make disjunct nulls
78      GalveUtil.makeDisjunctNulls(globalInstances);
79
80      globalInstances.forEach(instance ->
81       globalInstance.getRelationalAtoms().addAll(instance.
82              getRelationalAtoms()));
83
84      return globalInstance;
85  }
86
87  /**
88  * calculate adapted strong maximum extended recovery
89  * @param schemaMapping
90  * @return inverse schema mapping
91  */
92  static InverseSchemaMapping invert(
93      SchemaMapping schemaMapping) {
94
95      // normalize sttgds
96      var normalizedSttgds = new LinkedHashSet<IntegrityConstraint>();
97
98      for(IntegrityConstraint constraint :
99          schemaMapping.getIntegrityConstraints()) {
100         for(Atom headAtom : constraint.getHead()) {
101             var newHead = new HashSet<RelationalAtom>();
102             newHead.add((RelationalAtom) headAtom.copy());
103             normalizedSttgds.add(
104                 new STTgd(((STTgd)constraint).copyBody(),newHead));
```

```
105         }
106     }
107
108     // invert the sttgds and collect them by relations in the bodies in
109     // disjunctive sets
110     var disjunctiveSets = new HashMap<String,LinkedHashSet<STTgd>>();
111
112     for(IntegrityConstraint constraint : normalizedSttgds) {
113         var headRelationName = ((RelationalAtom)constraint.
114         getHead().stream().findAny().get()).getName();
115         if(disjunctiveSets.containsKey(headRelationName)) {
116             disjunctiveSets.get(headRelationName).add(GalveUtil.invert(
117             (STTgd)constraint));
118         }else {
119             var sttgds = new LinkedHashSet<STTgd>();
120             sttgds.add(GalveUtil.invert((STTgd)constraint));
121
122             disjunctiveSets.put(headRelationName, sttgds);
123         }
124     }
125
126     return new InverseSchemaMapping(
127         schemaMapping.getTargetSchema(),
128         schemaMapping.getSourceSchema(), disjunctiveSets);
129 }
130
131 /**
132 * delete the disjunctive sets that will not contribute to the result of
133 * the safe instance
134 * (delete disjunctive sets that split the chase tree in the disjunctive
135 * chase
136 *   and have no overlap with other disjunctive sets)
137 * @param disjunctiveSets
138 */
139 static void optimizeDisjunctiveSets(
140     HashMap<String,LinkedHashSet<STTgd>> disjunctiveSets) {
141     var toRemove = new LinkedHashSet<String>();
142
143     for(Entry<String,LinkedHashSet<STTgd>> disjunctiveSet :
144      disjunctiveSets.entrySet()) {
145         if(GalveUtil.headRelationsIntersection(disjunctiveSet.
146             getValue()).isEmpty()) {
147             if(!GalveUtil.existsDisjunktiveSetWithRelationInIntersection(
148                 disjunctiveSet.getValue(), disjunctiveSets)) {
149                 toRemove.add(disjunctiveSet.getKey());
150             }
151         }
152     }
153
154     toRemove.forEach(relationName ->
155      disjunctiveSets.remove(relationName));
156 }
157
```

```
158 /**
159 * extend the schema mapping so that it maps into new relations
160 * and transfers additional interesting attributes from the global
161 * schema
162 * @param inverse
163 */
164 static void extend(InverseSchemaMapping inverse) {
165
166 // generate disjunctive sets that map to new relations
167 // extend scheme of the source
168 var newDisjunctiveSets =
169     new HashMap<String,LinkedHashSet<STTgd>>();
170
171 // copy the schema to not disturb the new name generation when
172 // the schema is extended
173 var originalTargetSchema = GalveUtil.copySchema(
174     inverse.getTargetSchema());
175
176 for(Entry<String,LinkedHashSet<STTgd>> disjunctiveSet :
177     inverse.getDisjunctiveSets().entrySet()) {
178     var newDisjunctiveSet = new LinkedHashSet<STTgd>();
179
180     for(STTgd sttgd : disjunctiveSet.getValue()) {
181
182         var newSttgd = new STTgd(sttgd.getBody(),
183             new HashSet<RelationalAtom>());
184
185         for(RelationalAtom atom : sttgd.getHead()) {
186
187             var oldRelationName = atom.getRelationName();
188             var newRelationName = GalveUtil. generateNewName(
189                 oldRelationName, originalTargetSchema);
190
191             newSttgd.getHead().add(
192                 new RelationalAtom(newRelationName, atom.getTerms()));
193
194             var newAttributeTypeMap =
195                 new LinkedHashMap<String, String>();
196
197             for(Entry<String, String> entry :
198          inverse.getTargetSchema().get(oldRelationName).
199                     entrySet()) {
200                 newAttributeTypeMap.put(
201                     entry.getKey(), entry.getValue());
202             }
203             inverse.getTargetSchema().put(
204                 newRelationName, newAttributeTypeMap);
205         }
206         newDisjunctiveSet.add(newSttgd);
207     }
208     newDisjunctiveSets.put(
209         disjunctiveSet.getKey(), newDisjunctiveSet);
210 }
```

```
211
212 inverse.getDisjunctiveSets().clear();
213 inverse.getDisjunctiveSets().putAll(newDisjunctiveSets);
214
215 // search for interesting attributes
216 // these are the attributes that appear in a body but not in a head
217 var allAttibutes = new HashSet<String>();
218 var interestingAttibutes = new HashSet<String>();
219
220 // collect all attributes that occur in the relations of the body atoms
221 // of the disjunctive sets
222 for(Entry<String,LinkedHashSet<STTgd>> disjunctiveSet :
223     inverse.getDisjunctiveSets().entrySet()) {
224     for(Entry<String, String> attributeTypeMap :
225         inverse.getSourceSchema().get(disjunctiveSet.getKey()).
226             entrySet()) {
227         allAttibutes.add(attributeTypeMap.getKey());
228     }
229 }
230
231 for(String attribute : allAttibutes) {
232
233     var interesting = true;
234
235     for (Entry<String, LinkedHashSet<STTgd>> disjunctiveSet :
236         inverse.getDisjunctiveSets().entrySet()) {
237
238         if(inverse.getSourceSchema().get(disjunctiveSet.getKey()).
239             containsKey(attribute)) {
240             var schemaAttributeList = new ArrayList<String>();
241             inverse.getSourceSchema().get(
242                 disjunctiveSet.getKey()).keySet().
243             forEach(schemaAttribute ->
244                 schemaAttributeList.add(schemaAttribute));
245
246             var termPosition = schemaAttributeList.indexOf(attribute);
247
248             interesting = disjunctiveSet.getValue().stream().
249             anyMatch(sttgd -> {
250                 var bodyTerm = sttgd.getBody().stream().findFirst().
251                     get().getTerms().get(termPosition);
252                 // only variable can be interesting
253                 if(!bodyTerm.isVariable()) {
254                     return false;
255                 }
256                 // if the variable also appears in the head, then it is not
257                 // interesting
258                 if(sttgd.getHead().stream().anyMatch(headAtom ->
259                 headAtom.getTerms().contains(bodyTerm))) {
260                     return false;
261                 }else {
262                     // else still interesting
263                     return true;
```

```
264                 }
265             });
266             if(!interesting) {
267                 break;
268             }
269         }
270     }
271     if(interesting) {
272         interestingAttibutes.add(attribute);
273     }
274 }
275
276 // expand mappings of the disjunctive sets and
277 // the schema of the source with corresponding interesting
278 // attributes
279 for(String attribute : interestingAttibutes) {
280 for(Entry<String, LinkedHashSet<STTgd>> disjunctiveSet :
281 inverse.getDisjunctiveSets().entrySet()) {
282 // expand only the disjunctive sets with the corresponding
283 // attribute in the body
284 if(inverse.getSourceSchema().get(disjunctiveSet.getKey()).
285 containsKey(attribute)) {
286
287 var schemaAttributeList = new ArrayList<String>();
288 inverse.getSourceSchema().get(disjunctiveSet.getKey()).
289     keySet().
290 forEach(schemaAttribute ->
291     schemaAttributeList.add(schemaAttribute));
292
293 var termPosition = schemaAttributeList.indexOf(attribute);
294
295 for(STTgd sttgd : disjunctiveSet.getValue()) {
296
297     // it is enough to find the first term, the terms are the same
298     // by construction for each sttgd
299     var bodyAtom = sttgd.getBody().stream().findFirst().get();
300     var bodyTerm = bodyAtom.getTerms().get(termPosition);
301
302     for(RelationalAtom atom : sttgd.getHead()) {
303
304         // attribute is new for the relation of the atom
305         if(!inverse.getTargetSchema().get(atom.
306             getRelationName()).containsKey(attribute)) {
307             // add interesting attribute
308             atom.getTerms().add(bodyTerm);
309
310             // change all corresponding atoms in the heads of the
311             // sttgds in the disjunctive sets
312             for(Entry<String, LinkedHashSet<STTgd>>
313                 innerDisjunctiveSet :
314                 inverse.getDisjunctiveSets().entrySet()) {
315                 for(STTgd innerSttgd :
316                     innerDisjunctiveSet.getValue()) {
```

```
317                     for(RelationalAtom innerAtom :
318                         innerSttgd.getHead()) {
319                         // atoms with the same name are corresponding
320                         // atoms that must be changed
321                         if(innerAtom.getRelationName().equals(
322                             atom.getRelationName())) {
323                             // atom is in a disjunctive set that does not
324                             // have the attribute in the body atom
325                             if(!inverse.getSourceSchema().get(
326                                 innerDisjunctiveSet.getKey()).
327                                     containsKey(attribute)) {
328
329                                 // if the atom is a variable, add a new
330                                 // existence quantified variable
331                                 if(bodyTerm.isVariable()) {
332                                     var bodyTermCopy = new Variable(
333                                         VariableType.EXISTS,
334                                             ((Variable) bodyTerm).
335                                             getIndexName(),
336                                             ((Variable) bodyTerm).getIndex());
337                                     if(!innerAtom.getTerms().contains(
338                                         bodyTermCopy)) {
339                                         innerAtom.getTerms().add(
340                                             bodyTermCopy);
341                                     }
342                                 }else {
343                                     // this block is not needed anymore,
344                                     // because constants can't result in
345                                     //interesting attributes
346                                     // if this is changed:
347                                     // with a constant as bodyTerm a NEW
348                                     // existence quantified variable for the
349                                     // head must be created and added
350                                     var existingIndices =
351                                         new ArrayList<Integer>();
352                                     innerSttgd.getHead().forEach(
353                                         otherAtom ->
354                                             otherAtom.getTerms().stream().
355                                     filter(otherTerm ->
356                                         otherTerm.isVariable() &&
357                      ((Variable)otherTerm).
358                                             getIndexName().equals(attribute)).
359                                     forEach(otherTerm ->
360                                     existingIndices.add(((Variable)
361                                         otherTerm).getIndex())));
362                                     var newIndex = 1;
363                                     if(!existingIndices.isEmpty()) {
364                                         Collections.sort(existingIndices,
365                                             Collections.reverseOrder());
366                                         newIndex = existingIndices.stream().
367                                             findFirst().get() + 1 ;
368                                     }
369
```

```
370                                     var newVariabel = new Variable(
371                                         VariableType.EXISTS, attribute,
372                                             newIndex);
373                                     innerAtom.getTerms().add(newVariabel);
374                                 }
375                             }else {
376                                 // also change atoms that have already
377                                 // been adjusted in other disjunctive sets
378                                 // this case occurs when two sttgds in a
379                                 // disjunctive set have the same atoms in
380                                 // the head
381                                 var innerBodyAtom = innerSttgd.getBody().
382                                     stream().findFirst().get();
383                                 var innerBodyTerm = innerBodyAtom.
384                                     getTerms().get(termPosition);
385 
386                                 if(bodyAtom.getRelationName().equals(
387                                     innerBodyAtom.getRelationName()) &&
388                                 atom.getRelationName().equals(
389                                     innerAtom.getRelationName()) &&
390                                     !innerAtom.getTerms().contains(
391                                         innerBodyTerm)) {
392                                     innerAtom.getTerms().add(
393                                         innerBodyTerm);
394                                 }
395                             }
396                         }
397                     }
398                 }
399             }
400             // adjust the schema
401             var type = inverse.getSourceSchema().
402                 get(disjunctiveSet.getKey()).get(attribute);
403             inverse.getTargetSchema().get(atom.
404                 getRelationName()).put(attribute, type);
405         }else {
406             // attribute already exists in the relation of the atom
407             // replace the existing variables in the sttgd head with
408             // variables/constants from the body
409             inverse.getTargetSchema().entrySet().stream().
410             filter(schemaEntry ->
411                 schemaEntry.getKey().equals(atom.
412                     getRelationName())).
413             forEach(schemaEntry -> {
414                 var attributes = new ArrayList<String>(
415                     schemaEntry .getValue().keySet());
416                 atom.getTerms().remove(attributes.indexOf(
417                     attribute));
418                 atom.getTerms().add(attributes.indexOf(attribute),
419                     bodyTerm);
420             });
421         }
422     }
```

```
423 }
424 }
425 }
426 }
427 }
428
429 /**
430 * This implementation of the disjunctive CHASE does not compute the
431 * complete CHASE tree.
432 * The individual STTgds in the disjunctive sets are completely chased
433 * in, instead of executing one trigger for them at a time.
434 * For the optimized disjunctive sets the result should be the same.
435 * @param targetInstance
436 * @param inverse
437 * @return result instances
438 */
439 static LinkedHashSet<Instance> runAdaptedDisjunvtiveChase(
440     Instance targetInstance, InverseSchemaMapping inverse){
441     var resultInstances = new LinkedHashSet<Instance>();
442
443     // start recursive executions of the chase
444     resultInstances = runAdaptedDisjunctiveChaseRec(
445         targetInstance, inverse);
446
447     // remove the atoms of the global database and their relations from
448     // the schema
449     for(Instance instance : resultInstances) {
450         for(String key : inverse.getSourceSchema().keySet()) {
451             instance.getRelationalAtoms().removeAll(instance.
452                 getRelationalAtomsBySchema(key));
453             instance.getSchema().remove(key);
454         }
455     }
456
457     return resultInstances;
458 }
459
460 /**
461 * recursive calculation of the disjunctive chase
462 * one sttgd of the disjunctive sets is applied to each instance
463 * multiple sttgds in a disjunctive set create multiple instances
464 * @param instance
465 * @param inverse
466 * @return intermediate instances
467 */
468 static LinkedHashSet<Instance> runAdaptedDisjunctiveChaseRec(
469     Instance instance,  InverseSchemaMapping inverse){
470
471     var chaseResultInstances = new LinkedHashSet<Instance>();
472     var resultInstances = new LinkedHashSet<Instance>();
473
474     // termination condition of the recursion
475     if(inverse.getDisjunctiveSets().isEmpty()) {
```

```
476             resultInstances.add(instance);
477             return resultInstances;
478         }
479 
480         // select the first disjunctive set and start the chase for each sttgd
481         // individually
482         var disjunctiveSet = inverse.getDisjunctiveSets().entrySet().
483             stream().findFirst().get();
484         var disjunctiveSetRelation = disjunctiveSet.getKey();
485         var sttgds = disjunctiveSet.getValue();
486 
487         for (STTgd sttgd : sttgds) {
488             var constraints = new LinkedHashSet<IntegrityConstraint>();
489             // tgd instead sttgd to preserve the source tuples in
490             // chateau chase
491             constraints.add(new Tgd(sttgd.getBody(), sttgd.getHead()));
492             // run chase and add chase result
493             chaseResultInstances.add(GalveUtil.runChase(
494                 new SingleInput(instance, constraints)));
495         }
496 
497         // copy the disjunctive sets to not disturb the recursion
498         var disjunctiveSetsCopy = GalveUtil.copyDisjunktiveSets(
499             inverse.getDisjunctiveSets());
500 
501         // remove the used disjunctive set
502         disjunctiveSetsCopy.remove(disjunctiveSetRelation);
503 
504         var inverseCopy = new InverseSchemaMapping(
505             inverse.getSourceSchema(),inverse.getSourceSchema(),
506             disjunctiveSetsCopy);
507 
508         // recursion for each instance created by each sttgds
509         for(Instance chaseResultInstance : chaseResultInstances) {
510             resultInstances.addAll(runAdaptedDisjunctiveChaseRec(
511                 chaseResultInstance, inverseCopy));
512         }
513 
514         return resultInstances;
515     }
516 
517     /**
518     * calculate the safe instance as the average under a homomorphism
519     * from a set of instances
520     * @param instances
521     * @return safe instance
522     */
523     static Instance safeInstance(LinkedHashSet<Instance> instances) {
524 
525     var resultInstance = new Instance(OriginTag.INSTANCE);
526     // homomorphism mapping the atoms of the instances to the atoms
527     // in the result
528     // homomorphism collects the mappings over all computations
```

```
529 var homomorphism = new HashSet<TermMapping>();
530 // the flag to not restart the calculation
531 var breakWhile = true;
532
533 do {
534     // set the flag to not restart the calculation
535     breakWhile = true;
536     // stores an atom with a conflicting null value
537     var toRemove = new RelationalAtom();
538     // instance with an atom with conflicting null value
539     var instanceForRemove = new Instance(OriginTag.INSTANCE);
540
541     resultInstance = new Instance(OriginTag.INSTANCE);
542
543     if(instances.size()>0) {
544         var firstInstance = instances.stream().findFirst().get();
545
546         // disjunctive null values are important for the average
547         // calculation
548         GalveUtil.makeDisjunctNulls(instances);
549
550         // search for each atom in the first instance for matching atoms
551         // in the other instances
552         loop1: for(RelationalAtom outerAtom :
553         firstInstance.getRelationalAtoms()) {
554
555             var mappedOuterAtom = GalveUtil.
556                 applyTermMappings(outerAtom, homomorphism);
557
558             for(Instance instance : instances) {
559
560                 if(!instance.equals(firstInstance)) {
561                     var found = false;
562
563                     for(RelationalAtom innerAtom :
564                         instance.getRelationalAtoms()) {
565
566                         // if an atom already contains conflicting null
567                         // values it will be deleted
568                         if(outerAtom.hasHomomorphismTo(innerAtom)
569                         && GalveUtil.hasConflictingNulls(
570                             outerAtom, homomorphism)) {
571                             // add atom with conflicting null value
572                             toRemove = outerAtom;
573
574                             // note the instance containing the atom with the
575                             // conflicting null value
576                             instanceForRemove = firstInstance;
577
578                             // set the flag to restart the calculation
579                             breakWhile = false;
580
581                             // extend homomorphism to be able to recognize
```

```
582                         // later conflicts
583                         GalveUtil.addTermMapping(
584                             homomorphism, outerAtom, innerAtom);
585 
586                         // Cancel the calculation
587                         break loop1;
588                     }
589                     // if an atom already contains conflicting null
590                     // values it will be deleted
591                     if(innerAtom.hasHomomorphismTo(outerAtom) &&
592                     GalveUtil.hasConflictingNulls(
593                         innerAtom, homomorphism)) {
594                         // add atom with conflicting null value
595                         toRemove = innerAtom;
596 
597                         // note the instance containing the atom with the
598                         // conflicting null value
599                         instanceForRemove = instance;
600 
601                         // set the flag to restart the calculation
602                         breakWhile = false;
603 
604                         // extend homomorphism to be able to recognize
605                         // later conflicts
606                         GalveUtil.addTermMapping(
607                             homomorphism, innerAtom, outerAtom);
608 
609                         // Cancel the calculation
610                         break loop1;
611                     }
612 
613                     var mappedInnerAtom = GalveUtil.
614                         applyTermMappings(innerAtom, homomorphism);
615 
616                     // homomorphism can be extended
617                     if(GalveUtil.homomorphismIsExpandable(
618                         homomorphism, outerAtom, innerAtom)) {
619                         // extend homomorphism
620                         GalveUtil.addTermMapping(homomorphism,
621                             mappedOuterAtom, mappedInnerAtom);
622 
623                         found = true;
624                     }else
625                     // homomorphism can be extended
626                     if(GalveUtil.homomorphismIsExpandable(
627                         homomorphism, innerAtom, outerAtom)) {
628                         // extend homomorphism
629                         GalveUtil.addTermMapping(homomorphism,
630                         mappedInnerAtom, mappedOuterAtom);
631 
632                         found = true;
633                     }else
634                     // homomorphism exists, but existing homomorphism
```

```
635                         // cannot be extended
636                         if(outerAtom.hasHomomorphismTo(innerAtom)) {
637                             // add atom with conflicting null value
638                             toRemove = outerAtom;
639 
640                             // note the instance containing the atom with the
641                             // conflicting null value
642                             instanceForRemove = firstInstance;
643 
644                             // set the flag to restart the calculation
645                             breakWhile = false;
646 
647                             // extend homomorphism to be able to recognize
648                             // later conflicts
649                             GalveUtil.addTermMapping(homomorphism,
650                                 outerAtom, innerAtom);
651                             // Cancel the calculation
652                             break loop1;
653                         }else
654                         // homomorphism exists, but existing homomorphism
655                         // cannot be extended
656                         if(innerAtom.hasHomomorphismTo(outerAtom)) {
657                             // add atom with conflicting null value
658                             toRemove = innerAtom;
659 
660                             // note the instance containing the atom with the
661                             // conflicting null value
662                             instanceForRemove = instance;
663 
664                             // set the flag to restart the calculation
665                             breakWhile = false;
666 
667                             // extend homomorphism to be able to recognize
668                             // later conflicts
669                             GalveUtil.addTermMapping(homomorphism,
670                                 innerAtom, outerAtom);
671                             // Cancel the calculation
672                             break loop1;
673                         }
674                     }
675                     if(!found) {
676                         continue loop1;
677                     }
678                 }
679             }
680             // atom was found everywhere and can be added to the result
681             resultInstance.getRelationalAtoms().add(GalveUtil.
682                 applyTermMappings(outerAtom, homomorphism));
683         }
684         if(breakWhile) {
685             // apply the homomorphism to all atoms already added to
686             // the result
687             resultInstance = GalveUtil.applyTermMappings(
```

```
688                 resultInstance, homomorphism);
689         }else {
690             // find the instance with conflicting null value and remove the
691             // corresponding atom
692             for(Instance instance : instances) {
693                 if(instance.equals(instanceForRemove)) {
694                     instance.getRelationalAtoms().remove(toRemove);
695                 }
696             }
697         }
698     }
699     // remove the atom and start again
700     // removing the atom from the result is not sufficient,
701     // otherwise the same atom could be deleted from other
702     // mappings as well
703 }while(!breakWhile);
704 
705 return resultInstance;
706 }
707 
708 /**
709 * calculates tgd, which copies the new tuples into the original relations
710 * @param originSchema
711 * @param extendedSchema
712 * @param relationKeyAttribtes
713 * @return tgds for extension follow-up
714 */
715 static LinkedHashSet<IntegrityConstraint> followUpTgds(
716 HashMap<String, LinkedHashMap<String, String>> originSchema,
717 HashMap<String, LinkedHashMap<String, String>> extendedSchema,
718 HashMap<String, HashSet<String>> relationKeyAttribtes){
719 
720     var followUpTgds = new LinkedHashSet<IntegrityConstraint>();
721 
722     for(Entry<String, LinkedHashMap<String, String>>
723         extendedSchemaEntry : extendedSchema.entrySet()) {
724         // consider only extended relations
725         if(!originSchema.containsKey(extendedSchemaEntry.getKey())) {
726 
727             var extendedRelationName = extendedSchemaEntry.getKey();
728             var extendedAttributes = new ArrayList<String>(
729                 extendedSchemaEntry.getValue().keySet());
730 
731             // create tgd from new extended relation to corresponding
732             // original relation
733             var tgdBody = new HashSet<RelationalAtom>();
734             var tgdBodyAtom = new RelationalAtom(
735                 extendedRelationName);
736 
737             for(String attribut : extendedAttributes) {
738                 tgdBodyAtom.getTerms().add(new Variable(
739                     VariableType.FOR_ALL, attribut, 1));
740             }
```

```
741             tgdBody.add(tgdBodyAtom);
742
743             var originRelationName = extendedRelationName.
744                 substring(0, extendedRelationName.length()-1);
745             var originAttributes = new ArrayList<String>(
746                 originSchema.get(originRelationName).keySet());
747
748             var tgdHead = new HashSet<RelationalAtom>();
749             var tgdHeadAtom = new RelationalAtom(originRelationName);
750
751             for(String attribut : originAttributes) {
752                 tgdHeadAtom.getTerms().add(
753                     new Variable(VariableType.FOR_ALL, attribut, 1));
754             }
755             tgdHead.add(tgdHeadAtom);
756
757             followUpTgds.add(new Tgd(tgdBody,tgdHead));
758         }
759     }
760     return followUpTgds;
761 }
762
763 /**
764 * calculates the begds that clean the new and original tuples in the
765 * original relations
766 * @param originSchema
767 * @param extendedSchema
768 * @param relationKeyAttribtes
769 * @return begds for extension follow-up
770 */
771 static LinkedHashSet<IntegrityConstraint> followUpBEgds(
772 HashMap<String, LinkedHashMap<String, String>> originSchema,
773 HashMap<String, LinkedHashMap<String, String>> extendedSchema,
774 HashMap<String, HashSet<String>> relationKeyAttribtes){
775
776     var followUpBEgds = new LinkedHashSet<IntegrityConstraint>();
777
778     for(Entry<String, LinkedHashMap<String, String>>
779         extendedSchemaEntry : extendedSchema.entrySet()) {
780         // consider only extended relations
781         if(!originSchema.containsKey(extendedSchemaEntry.getKey())) {
782
783             var extendedRelationName = extendedSchemaEntry.getKey();
784             var originRelationName = extendedRelationName.
785                 substring(0, extendedRelationName.length()-1);
786             var originAttributes = new ArrayList<String>(
787                 originSchema.get(originRelationName).keySet());
788
789             // create bEgd to equate all non-key attributes
790             var keyAttributes = relationKeyAttribtes.
791                 get(originRelationName);
792
793             // there are more attributes than key attributes
```

```
794             if(originAttributes.size() > keyAttributes.size()) {
795
796                 var bEgdBody = new HashSet<RelationalAtom>();
797                 var bEgdBodyAtom1 = new RelationalAtom(
798                     originRelationName);
799                 var bEgdBodyAtom2 = new RelationalAtom(
800                     originRelationName);
801
802                 var bEgdHead = new HashSet<EqualityAtom>();
803
804                 for(int i = 0; i < originAttributes.size(); i++) {
805                     var attribute = originAttributes.get(i);
806                     // select according to the key attributes
807                     if(keyAttributes.contains(attribute)) {
808                         bEgdBodyAtom1.getTerms().add(
809                             new Variable(VariableType.FOR_ALL, attribute, 1));
810                         bEgdBodyAtom2.getTerms().add(
811                             new Variable(VariableType.FOR_ALL, attribute, 1));
812                     }else {
813                         bEgdBodyAtom1.getTerms().add(
814                             new Variable(VariableType.FOR_ALL, attribute, 1));
815                         bEgdBodyAtom2.getTerms().add(
816                             new Variable(VariableType.FOR_ALL, attribute, 2));
817
818                         bEgdHead.add(new EqualityAtom(
819                             new Variable(VariableType.FOR_ALL, attribute, 1),
820                         new Variable(VariableType.FOR_ALL, attribute, 2)));
821                     }
822                 }
823                 bEgdBody.add(bEgdBodyAtom1);
824                 bEgdBody.add(bEgdBodyAtom2);
825
826                 followUpBEgds.add(new BEgd(bEgdBody, bEgdHead));
827             }
828         }
829     }
830     return followUpBEgds;
831 }
832 }
```

GalveUtil.java

```
1  package galve;
2
3  import java.util.ArrayList;
4  import java.util.Collections;
5  import java.util.HashMap;
6  import java.util.HashSet;
7  import java.util.LinkedHashMap;
8  import java.util.LinkedHashSet;
9  import java.util.Map;
10 import java.util.Map.Entry;
11
12 import atom.RelationalAtom;
13 import chase.Chase;
14 import homomorphism.Homomorphism;
15 import homomorphism.TermMapping;
16 import instance.Instance;
17 import instance.OriginTag;
18 import instance.SchemaTag;
19 import integrityConstraint.STTgd;
20 import io.ChaseLog;
21 import io.SingleInput;
22 import term.Null;
23 import term.Term;
24 import term.Variable;
25 import term.VariableType;
26
27 /**
28 * outsourced functionality for implementation of galve technique
29 * according to master thesis:
30 *  "Datenintegration durch inverse Schemaabbildung:
31 *       Erweiterung der Rostocker GaLVE-Technik"
32 *
33 * @author Jakob Zimmer
34 *
35 */
36 final class GalveUtil{
37
38     /**
39     * run standard chateau chase
40     * @param input
41     * @return chase result instance
42     */
43     static Instance runChase(SingleInput input) {
44
45         return Chase.chase(input.getInstance(), input.
46             getConstraints(), new ChaseLog());
47     }
48
49     /**
50     * Create a HashMap for counting the indexes of new generated
51     * Null values.
```

```
52      * It has the attributes of instance I as key and the counter starting
53      * at 0 as value.
54      *
55      * @return nullcounter
56      */
57      static HashMap<String, Integer> initializeNullCounter(
58          Instance instance) {
59
60          var nullCounter = new HashMap<String, Integer>();
61          for(LinkedHashMap<String,String> attributeTypeMap :
62              instance.getSchema().values()) {
63              for(String attributeName : attributeTypeMap.keySet()) {
64                  nullCounter.put(attributeName, 0);
65              }
66          }
67
68          return nullCounter;
69      }
70
71      /**
72      * update the nullCounter as long as nullCheck fails
73      * @param variable
74      */
75      static void updateNullCounter(Instance instance,
76          HashMap<String, Integer> nullCounter, String attribute) {
77          do {
78              incrementCounter(nullCounter, attribute);
79          } while (!nullCheck(instance, nullCounter, attribute));
80      }
81
82      /**
83      * increment nullCounter for an attribute
84      * @param counter
85      * @param key
86      */
87      static void incrementCounter(Map<String, Integer> counter,
88          String key) {
89          var currentValue = counter.get(key);
90
91          counter.replace(key, currentValue + 1);
92      }
93
94      /**
95      * Checks whether the new null value or #E already exists within
96      * the instance
97      *
98      * @param instance
99      * @param nullCounter
100     * @return boolean for nullCheck
101     */
102     static boolean nullCheck(Instance instance,
103         HashMap<String, Integer> nullCounter, String attribute) {
104         int index = nullCounter.get(attribute);
```

```
105
106         // relation together with the position of the attribute in the relation
107         var relationAttributePositionMap =
108             new HashMap<String, Integer>();
109
110         for(Entry<String, LinkedHashMap<String, String>>
111             relationSchema : instance.getSchema().entrySet()) {
112             var attributes = new ArrayList<String>(
113                 relationSchema.getValue().keySet());
114             for(String attributeInSchema : attributes) {
115                 if (attributeInSchema.equals(attribute)) {
116                     relationAttributePositionMap.put(relationSchema.getKey(),
117                         attributes.indexOf(attributeInSchema));
118                 }
119             }
120         }
121
122         for (var relationAttributePosition :
123             relationAttributePositionMap.entrySet()) {
124             var relationAtoms = instance.getRelationalAtomsBySchema(
125                 relationAttributePosition.getKey());
126
127             for (var relationalAtom : relationAtoms) {
128                 var attrPosIndex = relationAttributePosition.getValue();
129                 var term = relationalAtom.getTerms().get(attrPosIndex);
130
131                 if (instance.getOriginTag() == OriginTag.INSTANCE) {
132                     if (term.isNull()) {
133                         var nullTerm = (Null) term;
134
135                         if (nullTerm.getIndex() == index) {
136                             return false;
137                         }
138                     }
139                 } else {
140                     if (term.isVariable()) {
141                         var variable = (Variable) term;
142
143                         if (variable.getVariableType() == VariableType.EXISTS) {
144                             if (variable.getIndex() == index) {
145                                 return false;
146                             }
147                         }
148                     }
149                 }
150             }
151         }
152
153         return true;
154     }
155
156     /**
157     * calculates a subschema with respect to a schemaTag
```

```
158     * @param instance
159     * @param schemaTag
160     * @return schema for schemaTag
161     */
162     static HashMap<String, LinkedHashMap<String, String>>
163     getSchemaForSchemaTag(Instance instance,
164         SchemaTag schemaTag){
165         // get relations for schemaTag
166         var relations = new ArrayList<String>();
167         instance.getSchemaTags().entrySet().stream().
168         filter(schemaTagEntry ->
169             schemaTagEntry.getValue().equals(schemaTag)).
170         forEach(schemaTagEntry ->
171             relations.add(schemaTagEntry.getKey()));
172         // fill schema with relations
173         var schema =
174             new HashMap<String, LinkedHashMap<String, String>>();
175         instance.getSchema().entrySet().stream().
176         filter(schemaEntry -> relations.contains(schemaEntry.getKey())).
177         forEach(schemaEntry -> schema.put(
178             schemaEntry.getKey(), schemaEntry.getValue()));
179         return schema;
180     }
181
182     /**
183     * average of the relations for which there are atoms in the heads
184     * of the sttgds
185     * only relations that appear in each head are returned
186     * @param disjunctiveSet
187     * @return head relations intersection for sttgds
188     */
189     static ArrayList<String> headRelationsIntersection(
190         LinkedHashSet<STTgd> disjunctiveSet) {
191
192         var headRealations = new ArrayList<String>();
193         var realationsCounterMap = new HashMap<String,Integer>();
194
195         for(STTgd sttgd : disjunctiveSet) {
196             for(RelationalAtom atom : sttgd.getHead()) {
197                 if(realationsCounterMap.containsKey(
198                     atom.getRelationName())) {
199                     realationsCounterMap.put(atom.getRelationName(),
200                     realationsCounterMap.get(atom.getRelationName()) + 1);
201                 }else {
202                     realationsCounterMap.put(atom.getRelationName(), 1);
203                 }
204             }
205         }
206
207         // add to result if there is a common relation in any head
208         realationsCounterMap.entrySet().stream().
209         filter(realtionCounter ->
210             realtionCounter.getValue() == disjunctiveSet.size()).
```

```
211         forEach(realtionCounter ->
212             headRealations.add(realtionCounter.getKey()));
213 
214         return headRealations;
215     }
216 
217     /**
218     * checks if there is another disjunctive set whose average of the
219     * head atoms has an atom
220     * from the union of the head relations of the given disjunctive set
221     *
222     * disjunctive sets whose average is not empty find themselves and
223     * result in true
224     * @param disjunctiveSet
225     * @param disjunctiveSets
226     * @return exists disjunktive set with common relation in there
227     * intersection
228     */
229     static boolean existsDisjunktiveSetWithRelationInIntersection(
230         LinkedHashSet<STTgd> disjunctiveSet,
231         HashMap<String,LinkedHashSet<STTgd>> disjunctiveSets) {
232 
233         var headRealtionsUnion = new ArrayList<String>();
234 
235         // union of the head relations
236         disjunctiveSet.forEach(sttgd -> sttgd.getHead().
237         forEach(atom ->
238             headRealtionsUnion.add(atom.getRelationName())));
239 
240         var result = disjunctiveSets.entrySet().stream().
241         anyMatch(otherDisjunctiveSet ->
242             headRelationsIntersection(otherDisjunctiveSet.getValue()).
243             stream().anyMatch(relation ->
244                 headRealtionsUnion.contains(relation)));
245 
246         return result;
247     }
248 
249     /**
250     * invert a sttgd by swapping the body and head and adjusting the
251     * all quantification and existence quantification
252     * @param sttgd
253     * @return inverted sttgd
254     */
255     static STTgd invert(STTgd sttgd) {
256 
257         var newBody = new HashSet<RelationalAtom>();
258         var newHead = new HashSet<RelationalAtom>();
259 
260         var bodyHomomorphismus = new Homomorphism();
261         var headHomomorphismus = new Homomorphism();
262 
263         // create homomorphism that maps all existence-quantified
```

```
264         // variables to all-quantified ones
265         sttgd.getHead().forEach(atom -> {
266             atom.getTerms().stream().
267             filter(term -> term.isVariable() &&
268             ((Variable)term).getVariableType().
269                 equals(VariableType.EXISTS)).
270             forEach(term ->
271                 bodyHomomorphismus.addMapping(new TermMapping(
272                     term, new Variable(
273                         VariableType.FOR_ALL,((Variable)term).getIndexName(),
274                         ((Variable)term).getIndex())));
275             newBody.add(bodyHomomorphismus.applyMappingsTo(atom));
276         });
277 
278         // create a homomorphism that maps all all-quantified variables
279         // to existence-quantified variables
280         // if the new body has no matching all-quantified variable
281         sttgd.getBody().forEach(atom -> {
282             atom.getTerms().stream().
283             filter(term -> term.isVariable() &&
284             !newBody.stream().anyMatch(newBodyAtom ->
285                 newBodyAtom.getTerms().contains(term))).
286             forEach(term ->
287                 headHomomorphismus.addMapping( new TermMapping(
288                 term, new Variable(VariableType.EXISTS,
289                     ((Variable)term).getIndexName(),
290                     ((Variable)term).getIndex())));
291             newHead.add(headHomomorphismus.applyMappingsTo(atom));
292         });
293 
294         return new STTgd(newBody, newHead);
295     }
296 
297     /**
298     * copy the given schema
299     * @param schema
300     * @return copy for a schema
301     */
302     static HashMap<String, LinkedHashMap<String, String>>
303     copySchema(HashMap<String, LinkedHashMap<String, String>>
304     schema) {
305 
306         var schemaCopy =
307             new HashMap<String, LinkedHashMap<String, String>>();
308 
309         for(Entry<String, LinkedHashMap<String, String>> schemaEntry :
310             schema.entrySet()) {
311             var schemaValueCopy = new LinkedHashMap<String, String>();
312             schemaEntry.getValue().entrySet().forEach(value ->
313                 schemaValueCopy.put(value.getKey(), value.getValue()));
314             schemaCopy.put(schemaEntry.getKey(),schemaValueCopy);
315         }
316 
```

```
317         return schemaCopy;
318     }
319 
320     /**
321     * copy the given disjunctive set
322     * @param disjunctiveSets
323     * @return copy for a disjunctive set
324     */
325     static HashMap<String,LinkedHashSet<STTgd>>
326     copyDisjunktiveSets(HashMap<String,LinkedHashSet<STTgd>>
327     disjunctiveSets){
328 
329         var disjunctiveSetsCopy =
330             new HashMap<String,LinkedHashSet<STTgd>>();
331 
332         for(Entry<String,LinkedHashSet<STTgd>> disjunctiveSet :
333             disjunctiveSets.entrySet()) {
334             var sttgdsCopy = new LinkedHashSet<STTgd>();
335             disjunctiveSet.getValue().forEach(sttgd ->
336             sttgdsCopy.add(new STTgd(
337             sttgd.copyBody(),sttgd.copyHead())));
338             disjunctiveSetsCopy.put(disjunctiveSet.getKey(), sttgdsCopy);
339         }
340 
341         return disjunctiveSetsCopy;
342     }
343 
344     /**
345     * apply all term mappings to an atom
346     * @param atom
347     * @param termMappings
348     * @return mapped atom
349     */
350     static RelationalAtom applyTermMappings(RelationalAtom atom,
351         HashSet<TermMapping> termMappings) {
352 
353         var tempAtom = atom.copy();
354         var newAtom = atom.copy();
355 
356         do {
357             tempAtom = newAtom;
358 
359             for(TermMapping mapping : termMappings) {
360                 newAtom = mapping.apply(newAtom);
361             }
362 
363         }while(!tempAtom.equals(newAtom));
364 
365         return newAtom;
366     }
367 
368     /**
369     * apply all term mappings to all atoms of the instance
```

```
370     * @param instance
371     * @param termMappings
372     * @return mapped instance
373     */
374     static Instance applyTermMappings(Instance instance,
375         HashSet<TermMapping> termMappings) {
376
377         var newInstance = new Instance(null,
378             instance.getSchema(),instance.getOriginTag());
379
380         for(RelationalAtom atom : instance.getRelationalAtoms()) {
381             newInstance.getRelationalAtoms().add(
382                 applyTermMappings(atom, termMappings));
383         }
384
385         return newInstance;
386     }
387
388     /**
389     * test whether a homomorphism can be formed between the atoms
390     * given the existing homomorphism
391     * @param homomorphism
392     * @param sourceAtom
393     * @param targetAtom
394     * @return boolean for extensibility of the homomorphism
395     */
396     static boolean homomorphismIsExpandable(HashSet<TermMapping>
397     homomorphism, RelationalAtom sourceAtom,
398         RelationalAtom targetAtom) {
399         var homomorphismIsExpandable = false;
400         var sourceTerms = sourceAtom.getTerms();
401         var targetTerms = targetAtom.getTerms();
402
403         // test if the two atoms have the same relation name and same size
404         if (sourceAtom.getName().equals(targetAtom.getName()) &&
405         sourceTerms.size() == targetTerms.size()) {
406             for (int i = 0; i < sourceTerms.size(); i++) {
407                 var sourceTerm = sourceTerms.get(i);
408                 var targetTerm = targetTerms.get(i);
409
410                 // test whether a homomorphism can be formed between
411                 // the terms
412                 // and whether a mapping to another term already exists in
413                 // the homomorphism,
414                 // which cannot be mapped to the new target term
415                 if(sourceTerm.hasHomomorphismTo(targetTerm)
416                 &&!homomorphism.stream().anyMatch(mapping ->
417                 mapping.getMappingSource().equals(sourceTerm) &&
418                 !mapping.getMappingTarget().hasHomomorphismTo(
419                     targetTerm) &&
420                     !targetTerm.hasHomomorphismTo(
421                         mapping.getMappingTarget()))) {
422
```

```
423                     homomorphismIsExpandable = true;
424                 }else {
425                     homomorphismIsExpandable = false;
426                     break;
427                 }
428             }
429         }
430 
431         return homomorphismIsExpandable;
432     }
433 
434     /**
435     * add a new term mappings
436     * @param homomorphism
437     * @param sourceAtom
438     * @param targetAtom
439     */
440     static void addTermMapping(
441         HashSet<TermMapping> homomorphism,
442         RelationalAtom sourceAtom, RelationalAtom targetAtom) {
443 
444         var sourceTerms = sourceAtom.getTerms();
445         var targetTerms = targetAtom.getTerms();
446 
447         // new mapping for each term in the atoms
448         for (int i = 0; i < sourceTerms.size(); i++) {
449             var sourceTerm = sourceTerms.get(i);
450             var targetTerm = targetTerms.get(i);
451 
452             // do not add trivial mappings and check if a homomorphism
453             // can be formed
454             if(!sourceTerm.equals(targetTerm) &&
455             sourceTerm.hasHomomorphismTo(targetTerm)) {
456                 var newTermMapping = new TermMapping(
457                     sourceTerm, targetTerm);
458 
459                 // add mapping if it does not exist yet
460                 if(!homomorphism.stream().anyMatch(mapping ->
461                 mapping.getMappingSource().equals(
462                     newTermMapping.getMappingSource()) &&
463                     mapping.getMappingTarget().equals(
464                         newTermMapping.getMappingTarget()))) {
465                     homomorphism.add(newTermMapping);
466                 }
467             }
468         }
469     }
470 
471     /**
472     * returns the null values that are mapped to different values by
473     * the same homomorphism
474     * @param homomorphism
475     * @param sourceAtom
```

```
476     * @param targetAtom
477     * @param instance
478     * @return conflicting nulls
479     */
480     static HashSet<Null> getConflictingNulls(
481         HashSet<TermMapping> homomorphism) {
482 
483         var conflictingNulls = new HashSet<Null>();
484 
485         // search conflicting null values that are mapped by the
486         // same homomorphism
487         // to different constants
488         homomorphism.stream().
489         filter(outerMapping -> outerMapping.getMappingSource().isNull()
490         && homomorphism.stream().anyMatch(innerMapping ->
491         innerMapping.getMappingSource().equals(
492             outerMapping.getMappingSource()) &&
493             !innerMapping.getMappingTarget().hasHomomorphismTo(
494                 outerMapping.getMappingTarget()) &&
495                 !outerMapping.getMappingTarget().hasHomomorphismTo(
496                     innerMapping.getMappingTarget()))).
497         forEach(outerMapping ->
498             conflictingNulls.add((Null) outerMapping.getMappingSource()));
499 
500         return conflictingNulls;
501     }
502     /**
503     * returns true if atoms containing null values
504     * that are mapped to different values by the same homomorphism
505     * @param homomorphism
506     * @param sourceAtom
507     * @param targetAtom
508     * @param instance
509     * @return atom has conflicting nulls
510     */
511     static boolean hasConflictingNulls(RelationalAtom atom,
512         HashSet<TermMapping> homomorphism) {
513 
514         var conflictingNulls = getConflictingNulls(homomorphism);
515 
516         return conflictingNulls.stream().anyMatch(conflictingNull ->
517             atom.getTerms().contains(conflictingNull));
518     }
519 
520     /**
521     * add a new term mappings for two atoms without mapping
522     * the conflicting null values
523     * that are mapped to different values by the same homomorphism
524     * @param homomorphism
525     * @param sourceAtom
526     * @param targetAtom
527     * @param instance
528     * @return
```

```
529     */
530     static void addTermMappingWithoutConflictingNulls(
531         HashSet<TermMapping> homomorphism,
532         RelationalAtom sourceAtom, RelationalAtom targetAtom) {
533 
534         var conflictingNulls = getConflictingNulls(homomorphism);
535 
536         var conflictingNullsIndices = new ArrayList<Integer>();
537 
538         conflictingNulls.forEach(conflictingNull ->
539             conflictingNullsIndices.add(sourceAtom.getTerms().
540                 indexOf(conflictingNull)));
541 
542         var sourceAtomCopy = sourceAtom.copy();
543         var targetAtomCopy = targetAtom.copy();
544 
545 
546         for(Integer index : conflictingNullsIndices) {
547             sourceAtomCopy.getTerms().remove(
548                 sourceAtom.getTerms().get(index));
549             targetAtomCopy.getTerms().remove(
550                 targetAtom.getTerms().get(index));
551         }
552 
553         addTermMapping(homomorphism, sourceAtom, targetAtom);
554     }
555 
556 
557     /**
558     * change the null values in each instance so that there is no overlap
559     * of null values among instances
560     * @param instances
561     */
562     static void makeDisjunctNulls(HashSet<Instance> instances) {
563 
564         // a single instance does not need to be changed
565         if(instances.size()>1) {
566             var firstInstance = instances.stream().findFirst().get();
567 
568             var nullcounter =
569             GalveUtil.initializeNullCounter(firstInstance);
570 
571             // the seen instances are checked for the presence of null values
572             var seenInstances = new HashSet<Instance>();
573             seenInstances.add(firstInstance);
574 
575             for(Instance instance : instances) {
576             // first instance does not need to be changed
577             if(!seenInstances.contains(instance)) {
578 
579             for(RelationalAtom atom : instance.getRelationalAtoms()) {
580                 for(Term term : atom.getTerms()) {
581                     if(term.isNull()){
```

```
582                         // is the zero value contained in a seen instance
583                         if(seenInstances.stream().anyMatch(seenInstance ->
584                         seenInstance.getRelationalAtoms().stream().
585                         anyMatch(seenAtom ->
586                             seenAtom.getTerms().stream().
587                         anyMatch(seenTerm ->
588                             seenTerm.equals(term))))) {
589                             // update null counter or all seen instances
590                             String attribute = instance.getAttributes(
591                             atom.getRelationName())
592                             .get(atom.getTerms().indexOf(term));
593                             seenInstances.forEach(seenInstance ->
594                                 GalveUtil.updateNullCounter(seenInstance,
595                                 nullcounter, attribute));
596
597                             var termCopy = term.copy();
598
599                             // also adjust all other equal null values to
600                             // the new index
601                             instance.getRelationalAtoms().
602                             forEach(otherAtom ->
603                                 otherAtom.getTerms().stream().
604                             filter(otherTerm -> otherTerm.equals(termCopy)).
605                             forEach(otherTerm ->
606                                 ((Null) otherTerm).setIndex(
607                                 nullcounter.get(attribute))));
608                         }
609                     }
610                 }
611             }
612             seenInstances.add(instance);
613             }
614             }
615         }
616     }
617
618     /**
619     * generates a new relation name for a given schema by adding a + to it
620     * @param oldName
621     * @param schema
622     * @return new name
623     */
624     static String generateNewName(String oldName, HashMap<String,
625         LinkedHashMap<String, String>> schema) {
626         do {
627             oldName = oldName.concat("+");
628         }while(schema.containsKey(oldName));
629
630         return oldName;
631     }
632 }
```

Chase.java

```
...
public class Chase {
...
    /**
* implements the customized chase step for bEgds
* for the galve technique
* @param bEgd
* @param homomorphism
*/
private void applyBEgd(BEgd bEgd, Homomorphism homomorphism) {
    log.addEntry("Applying homomorphism to egd...");

    for (var equalityHeadAtom : bEgd.getHead()) {

        // temporary terms that are the result of applying the mappings
        // of homomorphism to the
        // equality atoms of the egd
        var leftTerm = equalityHeadAtom.getTerm1();
        var rightTerm = equalityHeadAtom.getTerm2();

        // keep the applied mappings to be able to read them out afterwards
        var leftAppliedMappings = new HashSet<TermMapping>();
        var rightAppliedMappings = new HashSet<TermMapping>();

        for (var mapping : homomorphism.getTermMappings()) {
            if(mapping.getMappingSource().equals(leftTerm)) {
                log.addEntry("Apply mapping " +
                mapping.toString());
                leftTerm = mapping.apply(leftTerm);
                leftAppliedMappings.add(mapping);
            }
            if(mapping.getMappingSource().equals(rightTerm)) {
                log.addEntry("Apply mapping " +
                mapping.toString());
                rightTerm = mapping.apply(rightTerm);
                rightAppliedMappings.add(mapping);
            }

        }

        // get the atoms that created the mapping
        var leftAppliedMapping =
            leftAppliedMappings.stream().findFirst().get();
        var rightAppliedMapping = rightAppliedMappings.
            stream().findFirst().get();

        var leftAtom = homomorphismCacheForEGDs.
            applyMappingsTo(leftAppliedMapping.
                getTargetTermRelationPosition().getKey());
        var rightAtom = homomorphismCacheForEGDs.
            applyMappingsTo(rightAppliedMapping.
```

```
52                  getTargetTermRelationPosition().getKey());
53
54          var leftTermPosition = leftAppliedMapping.
55              getTargetTermRelationPosition().getValue();
56          var rightTermPosition = rightAppliedMapping.
57              getTargetTermRelationPosition().getValue();
58
59          // if both terms of the equality atom are constants
60          if (ChaseUtil.bothTermTypesAreConst(leftTerm, rightTerm)) {
61
62              if (!ChaseUtil.bothTermValuesAreEqual(leftTerm, rightTerm)) {
63
64                  // left is origin & right is not origin
65                  if(leftAtom.isOrigin() && !rightAtom.isOrigin()) {
66                      // replace constant from new atom with constant from
67                      // original atom (in instance & instanceWithNewFacts)
68                      for(RelationalAtom atom :
69                      instanceWithNewFacts.getRelationalAtoms()) {
70                          if(atom.equals(rightAtom)) {
71                              atom.getTerms().remove(atom.getTerms().
72                                  get(rightTermPosition));
73                              atom.getTerms().add(rightTermPosition, leftTerm);
74                          }
75                      }
76                      for(RelationalAtom atom : instance.getRelationalAtoms()) {
77                          if(atom.equals(rightAtom)) {
78                              atom.getTerms().remove(atom.getTerms().
79                                  get(rightTermPosition));
80                              atom.getTerms().add(rightTermPosition, leftTerm);
81                          }
82                      }
83                  }else
84                  // right is origin & left is not origin
85                  if(rightAtom.isOrigin() && !leftAtom.isOrigin()) {
86                      //replace constant from new atom with constant from original
87                      // atom (in instance & instanceWithNewFacts)
88                      for(RelationalAtom atom :
89                      instanceWithNewFacts.getRelationalAtoms()) {
90                          if(atom.equals(leftAtom)) {
91                              atom.getTerms().remove(atom.getTerms().
92                                  get(leftTermPosition));
93                              atom.getTerms().add(leftTermPosition, rightTerm);
94                          }
95                      }
96                      for(RelationalAtom atom : instance.getRelationalAtoms()) {
97                          if(atom.equals(leftAtom)) {
98                              atom.getTerms().remove(atom.getTerms().
99                                  get(leftTermPosition));
100                             atom.getTerms().add(leftTermPosition, rightTerm);
101                         }
102                     }
103                 }else
104                 // left is not origin & right is not origin
```

```
105                 if(!rightAtom.isOrigin() && !leftAtom.isOrigin()) {
106                     // replace both constants with a new null value
107                     // (in instance & instanceWithNewFacts)
108                     var attribute = new ArrayList<String>(
109                         instanceWithNewFacts.getSchema().get(
110                             leftAtom.getRelationName()).keySet()).get(
111                                 leftTermPosition);
112                     var newNull = ChaseUtil.createNewNull(
113                         attribute, nullCounter);
114 
115                     for(RelationalAtom atom :
116                         instanceWithNewFacts.getRelationalAtoms()) {
117                         if(atom.equals(leftAtom)) {
118                             atom.getTerms().remove(atom.getTerms().get(
119                                 leftTermPosition));
120                             atom.getTerms().add(leftTermPosition, newNull);
121                         }
122                     }
123 
124                     for(RelationalAtom atom :
125                         instanceWithNewFacts.getRelationalAtoms()) {
126                         if(atom.equals(rightAtom)) {
127                             atom.getTerms().remove(atom.getTerms().get(
128                                 rightTermPosition));
129                             atom.getTerms().add(rightTermPosition, newNull);
130                         }
131                     }
132 
133                     for(RelationalAtom atom : instance.getRelationalAtoms()) {
134                         if(atom.equals(leftAtom)) {
135                             atom.getTerms().remove(atom.getTerms().get(
136                                 leftTermPosition));
137                             atom.getTerms().add(leftTermPosition, newNull);
138                         }
139                     }
140 
141                     for(RelationalAtom atom : instance.getRelationalAtoms()) {
142                         if(atom.equals(rightAtom)) {
143                             atom.getTerms().remove(atom.getTerms().get(
144                                 rightTermPosition));
145                             atom.getTerms().add(rightTermPosition, newNull);
146                         }
147                     }
148                 }
149                 // left is origin & right is origin
150                 else
151                 {
152                     printErrorMessage(leftTerm, rightTerm);
153                 }
154             }
155         } else {
156             if (ChaseUtil.shouldExecuteMapping(leftTerm, rightTerm)) {
157                 var maxToMinMapping = ChaseUtil.
```

```
158                     buildMaxToMinMapping(leftTerm, rightTerm);
159
160                 // give the possibility to replace single constants from
161                 // new tuples
162                 // with other constants from original tuples with bEgds
163                 if(leftTerm.equals(maxToMinMapping.getMappingSource())) {
164                     maxToMinMapping.setSourceTermRelationPosition(
165                         new Pair<>(leftAtom, leftTermPosition));
166                     maxToMinMapping.setTargetTermRelationPosition(
167                         new Pair<>(rightAtom, rightTermPosition));
168                 }else {
169                     maxToMinMapping.setSourceTermRelationPosition(
170                         new Pair<>(rightAtom, rightTermPosition));
171                     maxToMinMapping.setTargetTermRelationPosition(
172                         new Pair<>(leftAtom, leftTermPosition));
173                 }
174
175                 // mapping from null to constant
176                 if(maxToMinMapping.getMappingSource().isNull() &&
177                 maxToMinMapping.getMappingTarget().isConstant()) {
178                     var mappingNull = maxToMinMapping.getMappingSource();
179                     var mappingConstant =
180                         maxToMinMapping.getMappingTarget();
181
182                     // homomorphismCacheForEGDs already has a mapping from
183                     // the null value to a constant from an original tuple
184                     if(homomorphismCacheForEGDs.getTermMappings().
185                     stream().anyMatch(mapping ->
186                         mapping.getMappingSource().equals(mappingNull)
187                     && mapping.getMappingTarget().isConstant() &&
188                     mapping.getTargetTermRelationPosition().
189                         getKey().isOrigin())) {
190                         // noop
191                     }else
192
193                     // homomorphismCacheForEGDs already has a mapping
194                     // from the null value to a constant from a new tuple
195                     if(homomorphismCacheForEGDs.getTermMappings().
196                     stream().anyMatch(mapping ->
197                         mapping.getMappingSource().equals(mappingNull)
198                     && mapping.getMappingTarget().isConstant() &&
199                     !mapping.getTargetTermRelationPosition().
200                         getKey().isOrigin())) {
201
202                         var mappingToRemove = new HashSet<TermMapping>();
203                         homomorphismCacheForEGDs.getTermMappings().
204                         stream().filter(mapping ->
205                             mapping.getMappingSource().equals(mappingNull)
206                         && mapping.getMappingTarget().isConstant() &&
207                         !mapping.getTargetTermRelationPosition().
208                             getKey().isOrigin()).
209                         forEach(mapping -> mappingToRemove.add(mapping));
210
```

```
211                         // new mapping maps to a constant from an original tuple
212                         // left term is the constant and
213                         // left atom is original or
214                         // right term is the constant and right atom is original
215                         if(leftTerm.isConstant() && leftAtom.isOrigin() ||
216                         rightTerm.isConstant() && rightAtom.isOrigin()) {
217                             // replaced the previous mapping
218                             homomorphismCacheForEGDs.getTermMappings().
219                                 removeAll(mappingToRemove);
220                             homomorphismCacheForEGDs.addMapping(
221                                 maxToMinMapping);
222                         }else
223
224                         // new mapping maps to ANOTHER constant from a
225                         // new tuple
226                         if(leftTerm.isConstant() && !leftAtom.isOrigin() &&
227                         !leftTerm.equals(mappingConstant)
228                         || rightTerm.isConstant() && !rightAtom.isOrigin() &&
229                         !rightTerm.equals(mappingConstant)) {
230                             // delete the previous mapping without replacement
231                             homomorphismCacheForEGDs.getTermMappings().
232                                 removeAll(mappingToRemove);
233                         }
234
235                         // such a mapping does not yet exist
236                     }else {
237                         homomorphismCacheForEGDs.addMapping(
238                             maxToMinMapping);
239                         homomorphismCacheForEGDs =
240                             homomorphismCacheForEGDs.composeWith(
241                                 homomorphismCacheForEGDs);
242                     }
243                 }
244                 // mapping from null to null
245                 else {
246                     homomorphismCacheForEGDs.addMapping(
247                         maxToMinMapping);
248                     homomorphismCacheForEGDs =
249                         homomorphismCacheForEGDs.composeWith(
250                             homomorphismCacheForEGDs);
251                 }
252             }
253         }
254
255         instanceWithNewFacts = homomorphismCacheForEGDs.
256             applyMappingsTo(instanceWithNewFacts);
257
258         log.addEntry("done");
259     }
260 }
261 }
```

D. Datenträger

Aufgrund der digitalen Abgabe dieser Arbeit existiert kein Datenträger mit der verwendeten Literatur und dem implementierten Programmcode. Sowohl die verwendete Literatur als auch der implementierte Programmcode stehen online zur Verfügung.

Die verwendete Literatur ist in der STUD.IP-Veranstaltung

Projekt: MA: Datenintegration durch inverse Schemaabbildungen

Link: https://studip.uni-rostock.de/dispatch.php/course/details?sem_id=cc5743eed7245379aef94008cd8cb10b

hochgeladen.

Der implementierte Programmcode ist im ChaTEAU-GitLab-Projekt

Branch: Masterarbeit-GaLVE-Jakob-Zimmer

Link: https://git.informatik.uni-rostock.de/ta093/chateau.git

versioniert.

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Masterarbeit selbstständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Stellen sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Rostock, den 31.08.2021

Jakob Zimmer