Universität Rostock Traditio et Innovatio

Masterarbeit

SO-TGDs und Skolemisierung für ChaTEAU

eingereicht von: Max Tilman Kaseler

eingereicht am: 30. August 2022

Gutachter: M.Sc. Tanja Auge
Prof. Dr.-Ing. Alke Martens

Zusammenfassung

Schemaabbildungen und Evolution können als ST-TGDs dargestellt werden. Da die Komposition von zwei Schemaabbildungen sich nicht immer als ST-TGD darstellen lässt, wurden sogenannte SO-TGDs eingeführt. Wir betrachten, wann zur Darstellung der Komposition SO-TGDs notwendig sind und ob die Inversen von Schemaabbildungen immer als ST-TGD dargestellt werden können. Außerdem betrachten wir die Inverse von SO-TGDs. Abschließend stellen wir vor, wie wir SO-TGDs und Skolemfunktionen in dem Chase-Tool ChaTEAU darstellen können.

Abstract

Shema mappings and evolutions can be expressed through ST-TGDs. The composition of two shema mappings can not always be represented by an ST-TGD. Therefore, SO-TGDs were introduced. In this thesis, we describe in which cases a SO-TGD is needed to represent the composition of two mappings and if the inverse of a mapping can always be represented as an ST-TGD. Furthermore, we investigate the inverse of SO-TGDs. At last, we present possible ways to represent SO-TGDs and skolemfunctions within the Chase-tool ChaTEAU.

Inhaltsverzeichnis

1. Einleitung

Der Chase-Algorithmus ist ein Universalwerkzeug der Datenbanktheorie und kann beispielsweise für die Integration, Reinigung und den Austausch von Daten genutzt werden [BKM+17]. Um diese Probleme zu lösen, übergeben wir dem Chase eine Menge von Abhängigkeiten. Eine Abhängigkeit ist ein logischer Ausdruck erster Ordnung und kann verschiedene Bedingungen darstellen. So können wir zum Beispiel Integritätsbedingungen wie Schlüssel- oder Fremdschlüsselbedingungen, aber auch Schemaabbildungen darstellen.
Nutzen wir zu Veranschaulichung eine Datenbank S in einer Geschäftsstelle. In der Datenbank werden die Kosten und Einnahmen von Aufträgen hinterlegt. Diese Daten sollen in einer geschäftsstellenübergreifenden Datenbank T hinzugefügt werden. Die Übertragung der Daten können wir durch eine Abbildung, ausgedrückt als eine Abhängigkeit A, formulieren. Während dieser Datenaustausch besteht, ändern sich die Bedürfnisse der Geschäftsstelle an ihre Datenbank. Es soll nun für einen Auftrag zusätzlich noch eine Auftragsnummer hinterlegt werden. Auch diese Schemaevolution stellen wir als eine Abhängigkeit E dar. Um die Schemaevolution durchzuführen, können wir die Abhängigkeit E mit dem Chase auf S anwenden und erhalten die Datenbank S'. In Abbildung 1.1 fassen wir diesen Sachverhalt graphisch zusammen.

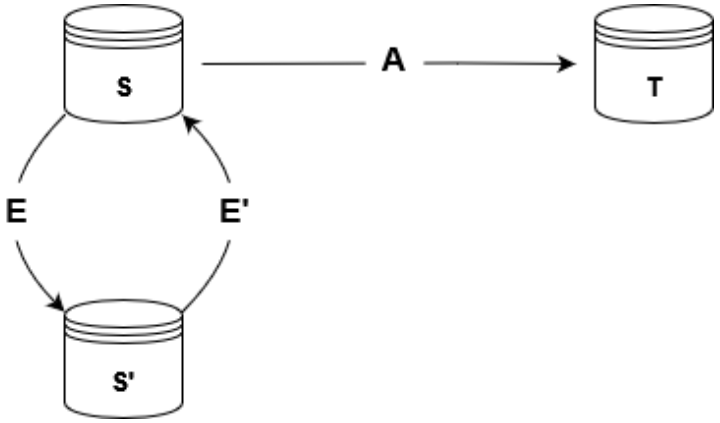

Abbildung 1.1.: Datenaustausch zwischen einer Datenbank S und T und eine Schemaevolution in der Datenbank S.

Die Auftragsdaten sollen aber weiterhin an die übergreifende Datenbank T weitergegeben werden. Um eine neue Abbildung zu erstellen, invertieren wir die Schemaevolution und verbinden die Inverse mit der ursprünglichen Schemaabbildung A. Hierzu stellen wir auch die Inverse als Abhängigkeit E' dar. Die beiden Abbildungen verbinden wir durch die Komposition $E' \circ A$. Hierdurch erhalten wir eine Abbildung von der Datenbank S' zur Datenbank T.

Wenn wir in einer Schemaevolution ein neues Attribut hinzufügen, drücken wir dies in der Abhängigkeit als eine existenzquantifizierte Variable aus. Dies sorgt dafür, dass bei der Anwendung der Abhängigkeit Nullwerte entstehen. Bei der Skolemisierung von Abhängigkeiten werden die existenzquantifizierten Variablen jeweils durch eine neue Skolemfunktion ersetzt. Diese erzeugt basierend auf ihrer Eingabe einen neuen Nullwert. Durch die Skolemisierung kann erreicht werden, dass weniger Nullwerte entstehen.

Außerdem finden Skolemfunktionen in Abhängigkeiten zweiter Ordnung Anwendung. In dieser Arbeit wollen wir zeigen, wie wir Skolemfunktionen in dem an der Universität Rostock entwickelten Chase-Tool ChaTEAU darstellen können.

Abhängigkeiten werden in der Logik erster Ordnung formuliert. Die Komposition von zwei Abbildungen kann aber nicht immer in der Logik erster Ordnung dargestellt werden. Um die Komposition trotzdem darstellen zu können, wurden in [FKPT05] SO-TGDs eingeführt. SO-TGDs sind Abhängigkeiten, welche in der Logik zweiter Ordnung formuliert sind. In dieser Arbeit betrachten wir, wann SO-TGDs bei der Komposition entstehen, ob bei der Invertierung SO-TGDs entstehen können und welche Form die Inverse einer SO-TGD hat. Außerdem werden wir vorstellen, wie wir SO-TGDs in ChaTEAU darstellen können.

1.1. Durchgehendes Studierenden-Beispiel

Wenn wir für ein Beispiel eine Instanz benötigen, werden wir diese als Menge von Fakten angeben. Ein Tupel der Relation R_i mit den Attribut- und Nullwerten $w_i, \ldots, w_n$ geben wir als den Fakt $R_i(w_i, \ldots, w_n)$ an. Eine Instanz mit zwei Tupeln in der STUDENT-Relation geben wir durch die Menge

$$\begin{aligned} \mathrm{I} = \{\ &\mathrm{STUDENT}(\texttt{1}, \texttt{Moore}, \texttt{Donald}, \texttt{Computer Science for Teaching}), \\ &\mathrm{STUDENT}(\texttt{2}, \texttt{Morgan}, \texttt{Sarah}, \texttt{Mathematics})\} \end{aligned}$$

an. Als durchgehendes Beispiel verwenden wir in dieser Arbeit eine fiktive Datenbank, welche von einer Universität über ihre Studierenden geführt wird. Diese enthält die Namen und die Noten der Studierenden. Unser Beispiel baut auf der in [SRH+20] genutzten Beispieldatenbank auf. Unsere Beispieldatenbank enthält die zwei Relationen STUDENT und GRADES. Die STUDENT-Relation enthält den Namen, Studiengang und die Matrikelnummer der Studierenden. In der GRADES-Relation werden die erreichten Noten mit Kurs, Semester und Matrikelnummer hinterlegt. Die Schemata der verwendeten Relationen sind:

- STUDENT = {`student_id`, `lastname`, `firstname`, `study_course`}
- GRADES = {`course_nr`, `student_id`, `semester`, `grade`}

Im Folgenden stellen wir den weiteren Aufbau der Arbeit vor.

1.2. Strukturierung der Arbeit

Als Erstes befassen wir uns im Kapitel 2 mit den für die Arbeit benötigten Grundlagen. Hierzu gehören die Logik erster und zweiter Ordnung. Diese nutzen wir um Abhängigkeiten zu formulieren. Des Weiteren befassen wir uns mit dem Chase, da dieser für die Anwendungen der Abhängigkeiten genutzt wird. Im 4. Kapitel geben wir einen Überblick über den Stand der Forschung und Technik. Als erstes betrachten wir, für welche Fälle bereits beschrieben ist, ob Abhängigkeiten zweiter Ordnung benötigt werden. Danach schauen wir uns bestehende Umsetzungen von Skolemisierung und Skolemfunktionen im Bereich des Chases an. Abschließend betrachten wir die Chase-Implementation ChaTEAU. Im Kapitel 4 stellen wir fest, wo in ChaTEAU Abhängigkeiten zweiter Ordnung entstehen können. Außerdem betrachten wir die Inverse der Abhängigkeiten zweiter Ordnung und wie wir diese Abhängigkeiten in ChaTEAU darstellen können. Abschließend realisieren wir Skolemfunktionen in ChaTEAU. Im Fazit (Kapitel 5) fassen wir die Erkenntnisse der Arbeit zusammen und zeigen auf, welche Fragen noch geklärt werden müssen.

2. Grundlagen

In diesem Kapitel werden wir die Grundlagen für die Themen dieser Arbeit betrachten. Als Erstes stellen wir im Abschnitt 2.1 die Logik erster und zweiter Ordnung vor. Im Abschnitt 2.2 stellen wir die für uns relevanten Aspekte der relationalen Datenbanken vor. Hiernach nutzen wir logische Ausdrücke, um im Abschnitt 2.3 Abhängigkeiten zu formulieren. Abschließend stellen wir im Abschnitt 2.4 den Chase-Algorithmus vor, welcher die vorher definierten Abhängigkeiten als Eingabe erhält.

2.1. Logik erster und zweiter Ordnung

Während die meisten Abhängigkeiten, die dem Chase übergeben werden, in der Prädikatenlogik erster Ordnung formuliert sind, wird für die Beschreibung von SO-TGDs die Logik zweiter Ordnung benötigt. In diesem Abschnitt soll erklärt werden, welche Formeln in Logik erster und welche in Logik zweiter Ordnung möglich sind. Dafür definieren wir als Erstes die kleinsten Elemente, die Symbole. Auf diesen aufbauend werden sogenannte Terme definiert, welche wir abschließend für die Definition von Ausdrücken nutzen.

Definition 2.1 (Ausdrücke der Logik erster Ordnung nach [EFT18])**.** Ausdrücke der Logik erster Ordnung lassen sich durch die folgenden Regeln erstellen:

1. *Variablen* sind Symbole der Form x_i mit $i \in \mathbb{N}$.
2. Eine *Funktion* hat die Form $f_i^{(k)}$, mit $i, k \in \mathbb{N}$. Hierbei ist i der Index zur Unterscheidung der verschiedenen Funktionen und k kann genutzt werden, um die Stelligkeit der Funktion anzugeben. Eine nullstellige Funktion heißt auch *Konstante*.
3. Ein *Prädikats-* oder *Relationssymbol* hat die Form $P_i^{(k)}$, mit $i, k \in \mathbb{N}$. Hierbei wird i zur Unterscheidung der verschiedenen Prädikate genutzt und k kann verwendet werden, um die Stelligkeit des Prädikats anzugeben.
4. *Terme* können durch endlich viele Anwendungen folgender Regeln entstehen:
   a) Jede Variable ist ein Term.
   b) Ist f eine n-stellige Funktion und $t_1, \ldots, t_n$ Terme, so ist auch $f(t_1, \ldots, t_n)$ ein Term.
5. Ausdrücke erster Ordnung können durch endlich viele Anwendungen folgender Regeln entstehen:
   a) Sind t_1 und t_2 Terme so ist $t_1 \equiv t_2$ ein Ausdruck.
   b) Ist P ein k-stelliges Prädikatssymbol und sind $t_1, \ldots, t_n$ Terme, so ist $P(t_1, \ldots, t_n)$ ein Ausdruck.
   c) Sind F und G Ausdrücke, so sind auch $\neg F$, $(F \vee G)$, $(F \wedge G)$, $F \rightarrow G$ und $(F \leftrightarrow G)$ Ausdrücke.
   d) Ist x eine Variable und f ein Ausdruck, so sind $\forall x : F$ und $\exists x : F$ Ausdrücke.

Einen Ausdruck bezeichnen wir als *atomar*, wenn dieser keinen kleineren Ausdruck enthält.
In der Logik zweiter Ordnung sind mehr Ausdrücke als in der Logik der ersten Ordnung möglich. Zusätzlich zu den in der Logik erster Ordnung definierten Ausdrücken sind hier auch Relations- und Funktionsvariablen möglich. Diese können auch in Verbindung mit einem Existenzquantor verwendet werden. Hierdurch kann beschrieben werden, dass eine Funktion existiert, aber es muss keine explizite Funktion angegeben werden. Für die Formulierung von Abhängigkeiten zweiter Ordnung werden wir Ausdrücke der Logik zweiter Ordnung verwenden.

Definition 2.2 (Ausdrücke der Logik zweiter Ordnung nach [EFT18])**.** Die Definition der Logik zweiter Ordnung baut auf der der ersten Ordnung auf, führt aber *Relationsvariablen* ein.

1. Eine *Relationsvariable* hat die Form $V_i^{(n)}$ mit $i, n \in \mathbb{N}$, wobei i zur Unterscheidung der verschiedenen Variablen genutzt wird und n zur Angabe der Stelligkeit verwendet werden kann.

2. Ein *Ausdruck der Logik zweiter Ordnung* kann durch endliche Anwendungen folgender Regeln entstehen:

   a) Alle Ausdrücke der Logik erster Ordnung nach der Definition 2.1 sind auch Ausdrücke zweiter Ordnung.

   b) Ist X eine n-stellige Relationsvariable und sind $t_1, \ldots, t_n$ Terme nach 2.1, so ist $X(t_1, \ldots, t_n)$ ein Ausdruck.

   c) Ist ϕ ein Ausdruck und ist X eine Relationsvariable, so ist $\exists X : \phi$ ein Ausdruck.

Für die Logik zweiter Ordnung können außerdem noch Funktionsvariablen eingeführt werden. Diese können zusammen mit einem Existenzquantor auftreten [EFT18].

Nachdem wir nun die Logik der ersten und zweiten Ordnung definiert haben, werden wir in einem Beispiel den Unterschied der beiden Logiken verdeutlichen.

Beispiel 2.1. *Als Beispiel für die Logik der ersten Ordnung stellen wir dar, dass für jede Note n ein Element in der Relation Prüfung P existiert, welches der Note einen Studierenden s zuordnet. Dies stellen wir mit folgendem Ausdruck dar:*

$$\forall n \rightarrow \exists s : P(s, n).$$

Um darzustellen, dass die Note nur vom Studierenden s abhängt, können wir eine existensquantifizierte Funktion f verwenden:

$$\exists f : P(s, f(s))$$

Da wir die Funktion nicht kennen und somit auch nicht explizit angeben können, kann dies nur in der Logik zweiter Ordnung ausgedrückt werden.

Nachdem wir die Logik der ersten und zweiten Ordnung betrachtet haben, werden wir diese im Abschnitt 2.3 nutzen um Abhängigkeiten in einer Datenbank darzustellen. Im nächsten Abschnitt stellen wir die für uns relevanten Aspekte der relationalen Datenbanken vor.

2.2. Relationale Datenbanken

Eine Relationale Datenbank besteht aus einer Menge von Relationen. Eine Relation können wir uns als Tabelle mit einer festen Länge vorstellen. Die Spalten dieser Tabelle nennen wir Attribute. Haben wir eine Relation R mit den Attributen $A_1, \ldots, A_n$ können wir diese mit einem Relationenschema $R(A_1, \ldots, A_n)$ beschreiben. Eine Zeile in der Relation bezeichnen wir als Tupel. Ein Tupel können wir als Fakt $R(v_1, \ldots, v_n)$ angeben. Die Werte $v_1, \ldots, v_n$ nennen wir Attributwerte. Diese können Konstanten oder Nullwerte sein. Wenn wir alle Relationenschemata einer Datenbank in einer Menge zusammenfassen, nennen wir diese Schema. In Abbildung 2.1 werden die Begriffe durch eine graphische Darstellung verdeutlicht.

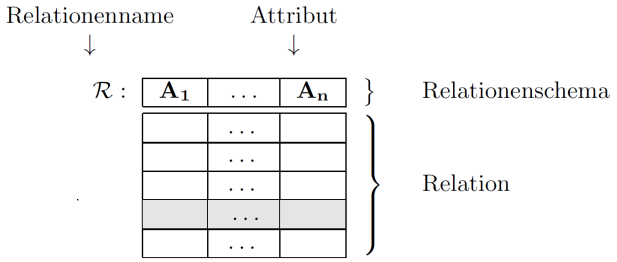

Abbildung 2.1.: Eine Relation mit Attributen A_i und einem grau hinterlegten Tupel[Aug22]

Nachdem wir uns die Begrifflichkeiten der relationalen Datenbanken noch einmal verdeutlicht haben, betrachten wir im folgenden Satz die Definition einer Schemaabbildung.

Definition 2.3 (Schemaabbildung nach [FKPT11b])**.** Eine Schemaabbildung ist ein Tripel $M = (S, T, \Sigma)$, wobei S das Quell- und T das Zielschema ist. Σ beschreibt die Verbindung Zwischen S und T durch eine Menge von Abhängigkeiten.

Im nächsten Abschnitt widmen wir uns den Abhängigkeiten, welche wir unter anderem für die Schemaabbildungen nutzen werden.

2.3. Abhängigkeiten

Integritätsbedingungen, wie z.B. funktionale Abhängigkeiten oder Verbundabhängigkeiten, können wir auch als logische Ausdrücke formulieren. Abhängigkeiten in einer allgemeineren Schreibweise darzustellen, hat den Vorteil, dass verschiedene Integritätsbedingungen und Schemaabbildungen beschrieben werden können. In diesem Abschnitt führen wir als Erstes Abhängigkeiten in der Logik erster Ordnung ein. Danach betrachten wir im Unterabschnitt 2.3.1 die Skolemisierung von Abhängigkeiten. Im zweiten Unterabschnitt 2.3.2 stellen wir die Inverse und Komposition von Abhängigkeiten vor. Abschließend stellen wir im Unterabschnitt 2.3.3 Abhängigkeiten der zweiten Ordnung vor.

Wir beginnen mit den Definitionen der Abhängigkeiten erster Ordnung. Integritätsbedingungen wie Funktionale Abhängigkeiten und Schlüsselbedingungen können durch eine *Equality Generating Dependency* beschrieben werden.

Definition 2.4 (*Equality Generating Dependency* (kurz EGD) nach [BKM$^+$17, AH18])**.** Ist $\phi(x)$ eine Konjunktion von Atomen aus x und $x_i, x_j \in x$, so ist eine EGD ein logischer Ausdruck der Form:

$$\forall x : (\phi(x) \rightarrow x_i = x_j).$$

Beispiel 2.2. *Wenn* `student_id` *ein Schlüsselattribut ist, heißt das, dass die Attribute der Studierenden-Relation funktional von dem Attribut* `student_id` *abhängen. Diese Abhängigkeit können wir durch folgende EGD darstellen:*

$$\text{STUDENT}(id, ln_1, fn_1, st_1) \wedge \text{STUDENT}(id, ln_2, fn_2, st_2) \rightarrow (ln_1 = ln_2) \wedge (fn_1 = fn_2) \wedge (st_1 = st_2)$$

In der Abhängigkeit beschreiben wir, dass wenn das Attribut `id` *in zwei Tupeln gleich ist, auch der Name und der Studiengang gleich sein sollen, also der Studierende eindeutig über die Id bestimmt werden kann.*

Abhängigkeiten wie Inklusions- oder Verbundabhängigkeiten, aber auch Schemaabbildungen können wir durch eine *Toupel Generating Dependency* beschreiben.

Definition 2.5 (*Tupel Generating Dependency* (kurz TGD) nach [BKM+17, AH18])**.** Seien $\phi(x)$ und $\psi(x, y)$ Konjunktionen von Atomen aus x bzw. x und y, so ist eine TGD ein logischer Ausdruck der Form:

$$\forall x : (\phi(x) \rightarrow \exists y : \psi(x, y)).$$

Wenn eine Abhängigkeit keine existenzquantifizierten Variablen enthält, wird diese als *voll* bezeichnet. Ist $\phi(x)$ eine Konjunktion über einem Quellschema S und $\psi(x, y))$ eine Konjunktion über einem Zielschema T, so wird die Abhängigkeit auch als *Source to Target Tuple Generating Dependency* (kurz ST-TGD) bezeichnet.

Beispiel 2.3. *Mit TGDs können verschiedene Arten von Abhängigkeiten dargestellt werden. Als Beispiel formulieren wir eine Inklusionsabhängigkeit als TGD. Um zu beschreiben, dass zu jeder Prüfung auch ein Studierender existieren muss, erstellen wir eine TGD, die aussagt, dass jede* `student_id` *aus der Grades-Relation auch in der Students-Relation vorkommen soll:*

$$\text{GRADES}(co, id, sem, gr) \rightarrow \exists LN, FN, ST : \text{STUDENT}(id, LN, FN, ST).$$

Sowohl TGDs als auch EGDs bestehen aus zwei Teilen, einem Rumpf ($\phi(x)$) und einem Kopf ($\exists y : \psi(x, y)$ bzw. $x_i = x_j$).

Definition 2.6 (GAV, LAV, GLAV nach [FKPT11b])**.** Die in der Definition 2.5 definierten ST-TGDs können in drei Arten unterteilt werden: *Global As View* (kurz GAV), *Local As View* (kurz LAV) und *Global Local As View* (kurz GLAV). GLAV ist die allgemeinste Form und beinhaltet alle ST-TGDs. GLAV enthält sowohl GAV als auch LAV.
Eine GAV ist eine ST-TGD mit der Eigenschaft, dass sie in ihrem Kopf nur eine atomare Aussage und keine existensquantifizierten Variablen hat und ist gegeben durch die Form:

$$\forall x : (\phi(x) \rightarrow T(x))$$

mit $T(x)$ ist eine atomare Aussage über dem Zielschema und $\phi(x)$ ist eine Konjunktion von Atomen über x. Eine LAV ist eine ST-TGD mit der Eigenschaft, dass sie in ihrem Rumpf nur eine atomare Aussage hat. Sie ist gegeben durch die Form:

$$\forall x : (Q(x) \rightarrow \exists y : \psi(x, y))$$

mit $Q(x)$ ist eine atomare Aussage über dem Quellschema und $\psi(x, y)$ ist eine Konjunktion von Atomen über x und y. Wiederholen sich in $Q(x)$ keine Variablen und kommen alle x_i aus x in $\psi(x, y)$ vor, so wird die LAV auch als *strickte* LAV bezeichnet. Eine ST-TGD kann sowohl in GAV als auch LAV liegen.

Beispiel 2.4. *Beispiele für die verschiedenen Arten von ST-TGDs sind:*

- *Um auf Basis der Tabellen* STUDENT *und* GRADES *eine Tabelle* STUDGRAD*, welche den Namen und die Note eines Studierenden enthält, zu erstellen, kann eine GAV-Abbildung verwendet werden:*

$$\text{STUDENT}(id, ln, fn, st) \wedge \text{GRADES}(co, id, sem, gr) \rightarrow \text{STUDGRAD}(ln, fn, gr).$$

- *Dass jedem Studierenden ein Fachschaftssprecher zugeordnet werden kann, können wird durch folgende ST-TGD in LAV ausdrücken. Die Id des Sprechers ist in der ST-TGD das existenzquantifizierte Attribut* $sprid$*:*

$$\text{STUDENT}(id, ln, fn, st) \rightarrow \exists Sprid : \text{VERTRETER}(id, Sprid).$$

- *Erstellen wir eine Tabelle, welche die Note, den Lehrstuhl und die Studienrichtung eines Studierenden einer Prüfung enthält, stellen wir dies durch eine GLAV-Abbildung dar:*

$$\text{STUDENT}(id, ln, fn, st) \wedge \text{GRADES}(co, id, sem, gr) \rightarrow \exists Lehr : \text{G}(Lehr, st, gr).$$

Da alle ST-TGDs GLAV-Abbildungen sind und alle GLAV-Abbildungen ST-TGDs sind, bedeuten diese Ausdrücke das Gleiche. Daher werden wir den Ausdruck GLAV-Abbildung nutzen, wenn alle ST-TGDs gemeint sind. Im nächsten Abschnitt 2.3.1 betrachten wir die Skolemisierung von Abhängigkeiten.

2.3.1. Skolemisierung von Abhängigkeiten

Bei der Skolemisierung von TGDs werden existenzquantifizierte Variablen durch eine neue Skolemfunktion ersetzt. Hierdurch kann bei mehrfacher Anwendung von TGDs verhindert werden, dass immer wieder neue Nullwerte entstehen.

Definition 2.7 (Skolemisierung von TGDs nach [BKM+17])**.** Sei r eine TGD nach Definition 2.5, so ist die skolemisierte Version vor r gegeben durch:

$$\forall x : (\phi(x) \rightarrow \psi(x, f(w))),$$

wobei jede existenzquantifizierte Variable y_i durch einen Skolemterm $f^r_{y_i}(w)$ ersetzt wird. Hierbei beinhaltet w alle Variablen, welche im Kopf und Rumpf der Abhängigkeit vorkommen. $f^r_{y_i}$ ist eine Skolemfunktion, wobei y_i für die ersetzte Variable und r für die Abhängigkeit steht.

Ein Skolemterm beschreibt, dass ein neuer Nullwert nur von dem Attributen x abhängt [BKM+17].

Beispiel 2.5. *Betrachten wir die TGD aus dem Beispiel 2.3:*

$$\text{GRADES}(co, id, sem, gr) \rightarrow \exists LN, FN, ST : \text{STUDENT}(id, LN, FN, ST).$$

Bei der Skolemisierung werden nun die Variablen ln, fn und st durch Skolemterme ersetzt. Die Eingabemenge für die Skolemfunktionen besteht nur aus dem Attribut id, da dieses als einziges im Kopf und Rumpf vorkommt. Die skolemisierte Version der TGD lautet:

$$\text{GRADES}(co, id, sem, gr) \rightarrow \text{STUDENT}(id, , f_{ln}(id), f_{fn}(id), f_{st}(id))$$

Bei den Skolem Ausdrücken $f^r_{y_i}(w)$ lassen wir in diesem Beispiel das r weg, da wir nur eine Abhängigkeit vorliegen haben. Die y_i sind die durch den jeweiligen Skolemterm ersetzten Attribute.

Nachdem wir die Skolemiesierung von Abhängigkeiten betrachtet haben, sehen wir im nächsten Abschnitt Inverse und Komposition von Abbildungen an.

2.3.2. Inverse und Komposition von GLAV-Abbildungen

In diesem Abschnitt stellen wir die Definitionen der Inverse einer GLAV-Abbildung und die der Komposition von zwei GLAV-Abbildungen vor. Für die Definition der Inversen nutzen wir den Chase, welcher im Abschnitt 2.4 vorgestellt wird. Als erstes betrachten wir die exakte Chase-Inverse.

Definition 2.8 (exakte Chase-Inverse nach [FKPT11b])**.** Sei M eine GLAV-Abbildung von einem Schema S_1 zu einem Schema S_2, so ist M' eine exakte Chase-Inverse für M, wenn M' eine GLAV-Abbildung von dem Schema S_2 zu dem Schema S_1 ist und für alle Instanzen I über S_1 $I = Chase_{M'}(Chase_M(I))$ gilt.

Als Nächstes werden wir die klassische Chase-Inverse vorstellen. Diese ist allgemeiner als die exakte Chase-Inverse und ist meist gemeint, wenn von der Chase-Inverse gesprochen wird.

Definition 2.9 ((klassische) Chase-Inverse nach [FKPT11b])**.** Sei M eine GLAV-Abbildung von einem Schema S_1 zu einem Schema S_2, so ist M' eine Chase-Inverse für M, wenn M' eine GLAV-Abbildung von dem Schema S_2 zu dem Schema S_1 ist und für alle Instanzen I über S_1 $I \leftrightarrow Chase_{M'}(Chase_M(I))$ gilt.

Nachdem wir die Inversen von Abbildungen definiert haben, werden wir nun die Komposition von zwei Schemaabbildungen definieren. Hierzu nutzen wir, dass wir eine Abbildung $M = (S, T, \Sigma)$ als eine binäre Relation $\text{Inst}(M)$ zwischen S und T darstellen können [FKPT05].

Definition 2.10 (Komposition nach [FKPT05])**.** Seien $M_{1,2} = (S_1, S_2, \Sigma_1)$ und $M_{2,3} = (S_2, S_3, \Sigma_2)$ zwei Schemaabbildungen, sodass die Schemata S_1, S_2, S_3 paarweise keine gleichen Relationssymbole haben, so ist $M_{1,3} = (S_1, S_3, \Sigma_3)$ eine Komposition von $M_{1,2}$ und $M_{2,3}$ wenn $\text{Inst}(M_{1,3}) = \{(I_1, I_2) |$ es existiert I_2, sodass $(I_1, I_2) \in \text{Inst}(M_{1,2})$ und $(I_2, I_3) \in \text{Inst}(M_{2,3})\}$ gilt.

Die Komposition von Abhängigkeiten lässt sich nicht immer in der Logik erster Ordnung darstellen. Aus diesem Grund führen wir im nächsten Abschnitt Abhängigkeiten in der Logik zweiter Ordnung ein.

2.3.3. TGDs zweiter Ordnung

Um die Komposition von mehreren GLAV-Abbildungen darzustellen, wurden in [FKPT05] TGDs zweiter Ordnung eingeführt. Diese erlauben im Gegensatz zu den in Definition 2.5 definierten TGDs nicht nur Ausdrücke der Logik erster Ordnung (Definition 2.1), sondern auch Ausdrücke der Logik zweiter Ordnung (Definition 2.2).

Definition 2.11 (*Second Order TGD* (kurz SO-TGD) nach [FKPT05])**.** Sei S ein Quellschema und T ein Zielschema, so hat eine SO-TGD die Form:

$$\exists f(\forall x_1(\phi_1 \rightarrow \psi_1) \wedge \ldots \wedge \forall x_n(\phi_n \rightarrow \psi_n)),$$

mit:

- Jedes Element aus f ist ein Funktionssymbol.
- Jedes ψ_i ist eine Konjunktion aus:
  - Atomaren Ausdrücken der Form $R_S(y_1, \ldots, y_k)$, wobei R_S ein Relationssymbol mit k Stellen über dem Schema S ist und $y_1, \ldots, y_k$ Elemente aus x_i sind,
  - Gleichungen der Form $t = t'$ mit t und t' aus x_i und f.
- Jedes ϕ_i ist eine Konjunktion aus atomaren Ausdrücken der Form $R_T(t_1, \ldots, t_l)$, wobei R_T ein Relationssymbol mit l Stellen über dem Schema T ist und $t_1, \ldots, t_l$ Elemente aus x_i und f sind.
- Jedes x_i kommt in mindesten einer atomaren Aussage von ϕ_i vor.

Beispiel 2.6. *Als Beispiel für eine SO-TGD übertragen wir das in [FKPT05] gegebene Beispiel auf unsere Studierenden-Datenbank. Das Beispiel geht von zwei Abbildungen aus. Die Abbildung $M_{1,2} = (S, S', \Sigma_1)$ bildet vom Schema S des Studierendenbeispiels auf das Schema S' ab. Die Abbildung $M_{2,3} = (S', S'', \Sigma_2)$ bildet vom Schema S' auf S'' ab.*
Durch die erste Abbildung $M_{1,2}$ entsteht die Relation VERTRETER*, welche beschreibt, durch welchen Fachschaftssprecher ein Studierender vertreten wird:*

$$\Sigma_1 = \{\text{STUDENT}(\mathit{id}, \mathit{ln}, \mathit{fn}, \mathit{st}) \rightarrow \exists \mathit{Sprid} : \text{VERTRETER}(\mathit{id}, \mathit{Sprid})\}.$$

In der zweiten Abbildung $M_{1,2}$ wird die Vertreter-Relation kopiert und eine Relation SELVERT *erstellt, welche alle Studierenden enthält, welche auch sich selbst vertreten:*

$$\begin{aligned}\Sigma_2 = \{&\text{VERTRETER}(\mathit{id}, \mathit{sprid}) \rightarrow \text{VERTRETER}(\mathit{id}, \mathit{sprid}),\\ &\text{VERTRETER}(\mathit{id}, \mathit{id}) \rightarrow \text{SELVERT}(\mathit{id})\}.\end{aligned}$$

Wenn die beiden Schemaabbildungen in Eine kombiniert werden sollen, so ist dies nur durch eine SO-TGD darstellbar. Dies ist der Fall, da in der Abbildung $M_{2,3}$ ein Vergleich über das, in der ersten Abbildung existensquantifizierte Attribut Sprid *durchgeführt wird. Die Schemaabbildung $M_{1,3} = M_{1,2} \circ M_{2,3}$ lautet:*

$$\begin{aligned}\exists f : (&\text{STUDENT}(\mathit{id}, \mathit{ln}, \mathit{fn}, \mathit{st}) \rightarrow \text{VERTRETER}(\mathit{id}, f(\mathit{id})),\\ &\text{STUDENT}(\mathit{id}, \mathit{ln}, \mathit{fn}, \mathit{st}) \wedge (\mathit{id} = f(\mathit{id})) \rightarrow \text{SELVERT}(\mathit{id})).\end{aligned}$$

Nachdem wir die Formulierung von verschiedenen Abhängigkeiten betrachtet haben, werden wir im nächsten Abschnitt den Chase vorstellen. Dieser ist ein Algorithmus zum Lösen mehrerer Probleme der Datenbanktheorie und kann die vorgestellten Abhängigkeiten verarbeiten.

2.4. Chase

Der Chase kann für verschiedene Probleme der Datenbanktheorie, wie z.B. Datenaustausch, Reformulieren von Anfragen unter Abhängigkeiten oder Bereinigen von Daten, genutzt werden. Zum Lösen der Probleme werden Abhängigkeiten schrittweise in ein Chase-Objekt eingearbeitet [BKM+17]. Im Rahmen dieser Arbeit beschränken wir uns auf Anwendungen, in denen der Chase auf eine Datenbank angewendet wird. Weitere Anwendungsmöglichkeiten, welche unter anderem von dem im Abschnitt 3.3 vorgestellten Chase-Tool ChaTEAU unterstützt werden, sind in der aus dem Paper [AH19] entnommenen Tabelle 2.1 dargestellt.

Parameter $\star$	**Objekt** $\bigcirc$	**Ergebnis** $\circledast$	**Ziel**
Abhängigkeiten	Datenbankschema	Datenbankschema mit Integritätsbedingungen	optimierter Datenbankentwurf
Abhängigkeiten	Anfrage	Anfrage	semantische Optimierung
Sichten	Anfrage	Anfrage über Sichten	AQuV
Operatoren	Anfrage	Anfrage mit gegebenen Operatoren	AQuO
ST-TGDs und EGDs	Quelldatenbank	Zieldatenbank	Datenaustausch, Datenintegration
TGDs und EGDs	Datenbank	modifizierte Datenbank	Bereinigung
TGDs und EGDs	unvollständige Datenbank	Anfrageergebnis	sichere Antworten
ST-TGDs, TGDs und EGDs	Datenbank	Anfrageergebnis	invertierbare Auswertung der Anfrage

Tabelle 2.1.: Anwendungen des Chases, mit der Art der übergebenen Abhängigkeiten, Art des Chase-Objektes und dem Ergebnis für ein jeweiliges Ziel, [AH19].

Die Abhängigkeiten werden in den im Abschnitt 2.3 besprochenen Formen formuliert und übergeben. Der Chase wird als Abfolge von Chase-Schritten definiert. In einem Chase-Schritt wird eine Abhängigkeit einmal angewendet. Um zu definieren, wie der Chase funktioniert, müssen wir noch klären, was ein Homomorphismus und ein Trigger ist.

Definition 2.12 (Homomorphismus nach [GMS12])**.** Seien K und J Instanzen über dem gleichen Datenbankschema. Ein Homomorphismus $h : K \to J$ ist eine Abbildung von $Konstanten(K) \cup Nullwerte(K)$ nach $Konstanten(J) \cup Nullwerte(J)$ mit:

1. Für alle c aus $Konstanten(K)$ gilt: $h(c) = c$.
2. Für jeden Fakt $R_i(t)$ aus K existiert in J ein Fakt $R_i(h(t))$.

Ähnlich zum Homomorphismus zwischen zwei Instanzen, ist auch der Homomorphismus von einer Formel $p(x)$ auf eine Instanz J möglich. Hierbei werden alle Werte aus x auf $Konstanten(J) \cup Nullwerte(J)$ abgebildet, sodass für jedes Atom $R(x_1, \ldots, x_n)$ aus $\phi(x)$ ein Fakt $R(h(x_1), \ldots, h(x_n))$ in J existiert.

Definition 2.13 (Trigger nach [BKM+17])**.** Sei eine Abhängigkeit d und eine Instanz I gegeben, so ist ein Trigger t für die Abhängigkeit d ein Homomorphismus von $\phi(x)$ nach I. Der Trigger t ist *aktiv*, wenn:

- d eine TGD der Form $\phi(x) \to \exists y : \psi(x, y)$ ist und für den Homomorphismus t keine Erweiterung existiert, die auch $\psi(x, y)$ auf I abbildet;
- d eine EGD der Form $\phi(x) \to x_i = x_j$ ist und $t(x_i) \neq t(x_j)$ gilt.

Definition 2.14 (Chase-Schritt nach [BKM+17])**.** Sei d eine TGD oder EGD, I eine Instanz und t ein aktiver Trigger für die Abhängigkeit d, so ist ein Chase-Schritt wie folgt definiert:

- Ist d eine TGD, wird I um die Fakten aus $h'(\psi(x, y))$ erweitert, wobei h' eine Substitution ist, sodass $\forall x_i \in x : h'(x_i) = t(x_i)$ und $h'(y)$ für alle $y_i \in y$ ein neuer Nullwert ist.
- Ist d eine EGD, scheitert der Chase-Schritt, wenn $t(x_i) \neq t(x_j)$ und $t(x_i), t(x_j)$ Konstanten sind. Andernfalls wird $\mu(I)$ wie folgt berechnet:
  - $t(x_j) \to t(x_i)$, wenn $t(x_i) \in Konstanten$,
  - $t(x_i) \to t(x_j)$, wenn $t(x_i) \notin Konstanten$.

Algorithmus 1 : Pseudocode vom Chase-Algorithmus nach [ASG+22].

```
Data : Menge von Abhängigkeiten Σ, Datenbankinstanz I_i
Result : Modifizierte Datenbankinstanz I_n
1  while I_j ≠ ⊥ and I_{j-1} ≠ I_j do
2      forall Trigger h für σ ∈ Σ do
3          if h ist ein aktiver Trigger then
4              if σ ist eine TGD then
5                  erweitere h und füge neue Toupel zu I_j hinzu
6              else if σ ist eine EGD then
7                  if verglichene Werte sind verschiedene Konstanten then
8                      I_j = ⊥
9                  else
10                     ersetze Nullwerte und Variabeln durch andere Nulwerten, Variabeln oder
                       Konstanten
```

Der Standard-Chase besteht aus einer Folge von Chase-Schritten und terminiert, wenn entweder alle Abhängigkeiten erfüllt sind oder der Chase scheitert. Die Reihenfolge der Chase-Schritte ist nicht festgelegt, sollte aber fair sein [BKM+17], was bedeutet, dass jeder mögliche Chase-Schritt irgendwann ausgeführt wird. Der Algorithmus 1 stellt dar, wie der Chase implementiert werden kann.

Im Algorithmus 1 beginnt in der dritten Zeile ein Chase-Schritt. Dieser beginnt mit dem Test, ob der gefundene Trigger aktiv ist. Wenn die zum Trigger zugehörige Abhängigkeit eine TGD ist, werden in der fünften Zeile neue Tupel erzeugt. Wenn die Abhängigkeit eine EGD ist und diese nicht erfüllt werden kann, wird in der achten Zeile der Algorithmus abgebrochen. Kann die EGD erfüllt werden, werden in der zehnten Zeile die Nullwerte oder Variablen ersetzt. Im folgenden Beispiel werden wir den Chase-Algorithmus nutzen, um Integritätsbedingungen in eine Datenbankinstanz einzuarbeiten.

Beispiel 2.7. *Als Beispiel für die Anwendung vom Standard-Chase werden wir zwei Abhängigkeiten in eine Datenbankinstanz einarbeiten. In der Instanz haben wir zwei Tupel aus der* STUDENT*-Relation und ein Tupel aus der* GRADES*-Relation:*

$$I_0 = \{\text{STUDENT}(1,\ \texttt{Moore}, \texttt{Donald}, \texttt{Computer Science for Teaching}),\\ \text{STUDENT}(1, \eta_1, \texttt{Donald}, \eta_2), \text{GRADES}(001, 2, SS\ 16, 1.7)\}.$$

Die Abhängigkeiten sind bereits in den Beispielen 2.3 und 2.2 erklärt worden. Die Menge der übergebenen Abhängigkeiten Σ *ist:*

$$\Sigma = \{\text{GRADES}(co, id, sem, gr) \to \exists LN, FN, ST : \text{STUDENT}(id, LN, FN, ST),\\ \text{STUDENT}(id, ln_1, fn_1, st_1) \wedge \text{STUDENT}(id, ln_2, fn_2, st_2) \to (ln_1 = ln_2) \wedge (fn_1 = fn_2) \wedge (st_1 = st_2).$$

Nun werden wir $Chase_\Sigma(I)$ berechnen. Wir finden für die beiden Abhängigkeiten jeweils einen aktiven Trigger und entscheiden uns zufällig für eine Reihenfolge, in der die Abhängigkeiten angewendet werden. Für die EGD finden wir die Abbildung t mit: $t(\texttt{id}) = 1$, $t(\texttt{ln}_1) = \texttt{Moore}$, $t(\texttt{ln}_2) = \eta_1$, $t(\texttt{fn}_1) = t(\texttt{fn}_2) = \texttt{Donald}$, $t(\texttt{st}_1) = \texttt{Computer Science for Teaching}$, $t(\texttt{st}_2) = \eta_2$. Aus dem Kopf der Abhängigkeit ergibt sich $\eta_1 \to \texttt{Moore}$ und $\eta_2 \to \texttt{Computer Science for Teaching}$. Die Instanz I_1 ist dann:

$$I_1 = \{\text{STUDENT}(1, \texttt{Moore}, \texttt{Donald}, \texttt{Computer Science for Teaching}),\\ \text{GRADES}(001, 2, \texttt{SS 16}, 1.7)\}.$$

Der zweite Trigger ist für die TGD und hat die Abbildung: $t'(\texttt{co}) = 001$, $t'(\texttt{id}) = 2$, $t'(\texttt{sem}) = \texttt{SS 16}$, $t'(\texttt{gr}) = 1.7$. Bei der Anwendung werden neue Nullwerte erzeugt. Die Instanz hat nun die Form:

$$I_2 = \{\text{STUDENT}(1, \texttt{Moore}, \texttt{Donald}, \texttt{Computer Science for Teaching}),\\ \text{GRADES}(001, 2, \texttt{SS 16}, 1.7), \text{STUDENT}(2, \eta_3, \eta_4, \eta_5)\}.$$

Da keine weiteren aktiven Trigger gefunden werden können, terminiert der Chase und I_2 ist unsere finale Instanz.

Skolem-Oblivious-Chase Der bis jetzt besprochene Chase ist der Standard-Chase. Es existieren aber noch weitere Varianten vom Chase [GMS12]. Eine der Varianten ist der Skolem-Oblivious-Chase. Dieser hat den Check auf einen aktiven Trigger in der dritten Zeile vom Algorithmus 1 nicht. Die Abhängigkeiten werden aber vor der Anwendung skolemisiert, wodurch bessere Terminierungseigenschaften erreicht werden.

Beispiel 2.8. *Im folgenden betrachten wir die Verhaltensweise des Skolem-Oblivious-Chase an einem Beispiel. Hierzu werden wir in eine Instanz eine skolemisierte TGD einarbeiten. Die Ausgangsinstanz enthält drei Einträge in der* GRADES*-Relation.*

$$I_0 = \{\text{GRADES}(001, 1, \texttt{SS 16}, 1.7),\ \text{GRADES}(002, 1, \texttt{WS 16/17}, 2.0),\\ \text{GRADES}(001, 2, \texttt{SS 16}, 3.0)\}.$$

Die übergebene TGD stellt dar, dass zu jedem Eintrag in der GRADES*-Relation ein Eintrag in der* STUDENT*-Relation vorhanden sein soll.*

$$\text{GRADES}(\texttt{co}, \texttt{id}, \texttt{sem}, \texttt{gr}) \to \text{STUDENT}(\texttt{id}, f_{ln}(\texttt{id}), f_{fn}(\texttt{id}), f_{st}(\texttt{id}))$$

Im ersten Chaseschritt finden wir für die Abhängigkeit drei Trigger. Da kein Check erfolgt, ob diese aktiv sind, führen wir die Abhängigkeit für jeden der Trigger aus und erhalten die Instanz

$$I_1 = \{\text{GRADES}(001, 1, \texttt{SS 16}, 1.7),\ \text{GRADES}(002, 1, \texttt{WS 16/17}, 2.0), \text{GRADES}(001, 2, \texttt{SS 16}, 3.0)\\ \text{STUDENT}(1, f_{ln}(1), f_{fn}(1), f_{st}(1)), \text{STUDENT}(1, f_{ln}(1), f_{fn}(1), f_{st}(1)),\\ \text{STUDENT}(2, f_{ln}(2), f_{fn}(2), f_{st}(2))\}.$$

Das rote Tupel in der Instanz entfällt, da der Chase mengenbasiert arbeitet und eine Menge keine Duplikate enthält. Es stellt aber dar, dass vor der Anwendung einer Abhängigkeit nicht getestet wird, ob diese aktiv ist. Im zweiten Durchlauf werden wieder die drei Trigger gefunden und die TGDs angewandt. Durch die Skolemfunktionen entstehen hierbei nur Duplikate, sodass wir die Instanz I_2 erhalten.

$$I_2 = \{\text{GRADES}(\texttt{001}, \texttt{1}, \texttt{SS 16}, \texttt{1.7}),\ \text{GRADES}(\texttt{002}, \texttt{1}, \texttt{WS 16/17}, \texttt{2.0}), \text{GRADES}(\texttt{001}, \texttt{2}, \texttt{SS 16}, \texttt{3.0})$$
$$\text{STUDENT}(\texttt{1}, f_{ln}(\texttt{1}), f_{fn}(\texttt{1}), f_{st}(\texttt{1})), \text{STUDENT}(\texttt{2}, f_{ln}(\texttt{2}), f_{fn}(\texttt{2}), f_{st}(\texttt{2}))\}.$$

Da die Instanzen I_1 und I_2 gleich sind, terminiert der Chase durch die Bedingung der While-Schleife in der ersten Zeile vom Algorithmus 1.

3. Aktueller Stand der Forschung und Technik

In diesem Kapitel werden wir andere Arbeiten betrachten, auf welche diese Arbeit aufbaut. Im ersten Abschnitt 3.1 werden wir betrachten, welche Erkenntnisse über die mögliche Entstehung von Abhängigkeiten zweiter Ordnung bereits vorhanden sind. Dabei betrachten wir die Komposition und die Invertierung von ST-TGDs. Im darauf folgenden Abschnitt 3.2 beschäftigen wir uns mit der Skolemfunktion, wobei wir besonders auf die Umsetzung von Skolemfunktionen achten. Als Letztes werden wir im Abschnitt 3.3 das in dieser Arbeit zu erweiternde Projekt ChaTEAU vorstellen.

3.1. Abhängigkeiten zweiter Ordnung im Chase

Um festzustellen, wann SO-TGDs in ChaTEAU auftreten können, werden wir betrachten, ob diese bei der Komposition und bei der Invertierung von ST-TGDs entstehen können. In den beiden folgenden Abschnitten werden Erkenntnisse zu dieser Fragestellung aus bestehenden Arbeiten gesammelt.

3.1.1. Komposition von Abbildungen

Die Komposition von Schemaabbildungen lässt sich nicht immer durch GLAV-Abbildungen darstellen, weshalb in [FKPT05] die SO-TGDs eingeführt wurden. Wir werden nun betrachten, welche Erkenntnisse es bereits zu der Fragestellung: „Wann wird eine SO-TGD benötigt und wann können andere Arten von Abbildungen genutzt werden, um das Ergebnis einer Komposition darzustellen?“ gibt. Die folgenden Sätze (Satz 3.1, 3.2, 3.3) beschreiben Fälle, in denen wir die Kompositionen durch eine GLAV-Abbildung darstellen können. Da eine GLAV-Abbildung das Gleiche wie eine allgemeine ST-TGD ist, werden wir im Folgenden nur den Ausdruck GLAV-Abbildung verwenden.

Satz 3.1 (Theorem 1 aus [FKPT11b]). *Seien M_1 und M_2 zwei aufeinander folgende Schemaabbildungen, so gilt:*

1. *Sind M_1 und M_2 GAV-Abbildungen, so kann $M_1 \circ M_2$ durch eine GAV-Abbildung dargestellt werden.*
2. *Ist M_1 eine GAV-Abbildung und M_2 eine GLAV-Abbildung, so kann $M_1 \circ M_2$ durch eine GLAV-Abbildung dargestellt werden.*

Satz 3.2 (nach Proposition 4.2 aus [FKPT05]). *Seien $M_1 = (S_1, S_2, \Sigma_1)$ und $(M_2 = S_2, S_3, \Sigma_2)$ zwei Schemaabbildungen, sodass Σ_1 eine endliche Menge von vollen GLAV-Abbildungen und Σ_2 eine endliche Menge von GLAV-Abbildungen ist, so lässt sich die Komposition $M_1 \circ M_2$ durch eine endliche Menge an GLAV-Abbildungen darstellen. Eine Anfrage, welche die Komposition darstellt, kann in polynomialer Zeit berechnet werden.*

Satz 3.3 (Nach Theorem 4.5 aus [AFM10]). *Existieren zwei Schemaabbildungen $M_1 = (S_1, S_2, \Sigma_1)$ und $M_2 = (S_2, S_3, \Sigma_2)$, wobei Σ_1 und Σ_2 Mengen von strikten LAV-Abbildungen sind, so kann die Komposition auch durch eine Menge strikter LAV-Abbildungen dargestellt werden.*

Nachdem wir einige Fälle betrachtet haben, in denen die Komposition sich durch eine GLAV-Abbildung darstellen lässt, stellen wir in den folgenden drei Sätzen (Satz 3.6, 3.4, 3.5) Fälle vor, in denen die Komposition nicht durch eine endliche Menge von GLAV-Abbildungen dargestellt werden kann.

Satz 3.4 (nach Proposition 4.4 aus [FKPT05]). *Es existieren zwei Schemaabbildungen $M_1 = (S_1, S_2, \Sigma_1)$ und $M_2 = (S_2, S_3, \Sigma_2)$, wobei Σ_1 eine endliche Menge von GLAV-Abbildungen und Σ_2 eine endliche Menge von vollen GLAV-Abbildungen ist und für die Komposition $M_1 \circ M_2$ folgendes gilt:*

1. *Die Komposition kann nicht durch eine endliche Menge, sondern nur durch eine unendliche Menge an GLAV-Abbildungen dargestellt werden.*

2. *Die Komposition kann durch einen Ausdruck erster Ordnung beschrieben werden und eine Anfrage, welche die Komposition darstellt, kann in polynomialer Zeit berechnet werden.*

Satz 3.5 (nach Proposition 4.5 aus [FKPT05]). *Es existieren zwei Schemaabbildungen $M_1 = (S_1, S_2, \Sigma_1)$ und $M_2 = (S_2, S_3, \Sigma_2)$, wobei Σ_1 aus einer einzigen GLAV-Abbildung besteht und Σ_2 eine endliche Menge von vollen GLAV-Abbildungen ist und die Komposition $M_1 \circ M_2$ nicht durch eine endliche oder unendliche Menge an GLAV-Abbildungen dargestellt werden kann.*

Satz 3.6 (nach Theorem 4.6 aus [FKPT05]). *Es existieren zwei Schemaabbildungen $M_1 = (S_1, S_2, \Sigma_1)$ und $M_2 = (S_2, S_3, \Sigma_2)$, wobei Σ_1 eine endliche Menge von GLAV-Abbildungen mit höchstens einer existenzquantifizierten Variable ist, Σ_2 aus einer einzelnen vollen GLAV-Abbildung besteht und die Komposition $M_1 \circ M_2$ folgende Eigenschaften hat:*

1. *Eine Anfrage, welche die Komposition darstellt, liegt in der Komplexitätsklasse NP-vollständig.*

2. *Die Komposition kann nicht in der "finite-variable infinitary logic" ausgedrückt werden und benötigt mindestens die Fix-Punkt-Logik.*

Die Betrachtung der bisherigen Ergebnisse zur Komposition von GLAV-Abbildungen schließen wir mit dem Satz 3.7 ab. Dieser beschreibt, dass die Komposition von GLAV-Abbildungen immer durch eine SO-TGD dargestellt werden kann.

Satz 3.7 (nach Proposition 4.8 aus [FKPT05]). *Seien $M_1 = (S_1, S_2, \Sigma_1)$ und $M_2 = (S_2, S_3, \Sigma_2)$ zwei Schemaabbildungen, sodass Σ_1 und Σ_2 endliche Mengen von GLAV-Abbildungen sind, so lässt sich die Komposition $M_1 \circ M_2$ durch einen Ausdruck der Logik zweiter Ordnung ohne Allquantifizierung darstellen. Eine Anfrage, welche die Komposition darstellt, ist in der Komplexitätsklasse NP.*

3.1.2. Invertierung von Abbildungen

Nachdem wir in Abschnitt 3.1.1 gesehen haben, dass bei der Komposition von Abhängigkeiten SO-TGDs auftreten können, werden wir nun die Invertierung betrachten. Auch bei der Invertierung stellen wir uns die Frage, welche Art von Abhängigkeit wann entsteht. Als Inverse betrachten wir im Folgenden die Chase-Inverse. In [FKPT11b] wurde für die Chase-Inversen folgender Satz aufgestellt:

Satz 3.8 (Theorem 2 aus [FKPT11b]). *Sei M ein GLAV-Schemamapping. Wenn M eine Chase-Inverse hat, dann hat M eine GAV Chase-Inverse.*

In den Abschnitten 4.1.1 und 4.1.2 des Konzepts werden wir diesen Satz verwenden, um zu zeigen, dass für die Darstellung von Chase-Inversen keine SO-TGDs notwendig sind.

3.2. Skolemisierung

In diesem Abschnitt werden wir betrachten, wie die Skolemisierung beziehungsweise Skolemfunktionen bereits umgesetzt wurden. In [BKM+17] wird eine Beispiel für die Umsetzung einer skolemisierten Abhängigkeit in SQL gegeben. So kann zum Beispiel die Abhängigkeit $R(a, b) \rightarrow R(a, f(a))$ durch die SQL-Anfrage 3.1 dargestellt werden. Hierbei wird der Skolemterm als neuer Wert eingetragen. In dem in [HY90] vorgestellten System werden Skolemterme aufgelöst, indem diese durch einen neuen Identifikator ersetzt werden.

```
INSERT INTO R(a,b)
SELECT DISTINCT R.a, append('_Sk_f(',R.a,')') FROM R
```

Listing 3.1: Die Abhängigkeit $R(a, b) \rightarrow R(a, f(a))$ dargestellt als SQL-Anfrage.

Obwohl zu Skolemfunktionen weitere Literatur existiert, beschäftigt sich diese mit anderen Bereichen und ist nicht für diese Arbeit relevant. In [Kom19] wird zum Beispiel die Eliminierung von Skolemfunktionen untersucht. Mögliche Umsetzungen von Skolemfunktionen werden in Kontexten beschrieben, welche sich nicht auf diese Arbeit übertragen lassen. Ein Beispiel hierfür ist [JSC+15]. In der Dokumentation von bestehenden Implementationen vom Skolem-Chase, wie zum Beispiel Llunatik[1], ist die Umsetzung der Skolemfunktionen nicht genau dokumentiert. Auch die zu Llunatik gehörende Literatur bietet keine Informationen über die genaue Umsetzung der Skolemfunktionen.

3.3. ChaTEAU

ChaTEAU steht für **Cha**se for **T**ransforming, **E**volving, and **A**dapting databases and queries, **U**niversal approach und ist eine an der Universität Rostock entwickelte Implementation des Standard-Chases. ChaTEAU ist ein universelles Chase-Tool, ist also im Gegensatz zu anderen Chase-Tools nicht für einen speziellen Anwendungsfall optimiert, sondern ist für verschiedene Anwendungsfälle anwendbar. An der Universität Rostock wurde ChaTEAU beispielsweise für das Answering Queries using Views Problem (siehe [Tre22]) und zur Optimierung von Anfragen (siehe [Zim20]) genutzt. Zusätzlich zu diesen Arbeiten wird ChaTEAU in den Forschungsprojekten PArADISE und ProSA genutzt. In dem Projekt PArADISE (siehe [GH16]) wird die Reformulierung von Anfragen genutzt, um diese datensparsam auszuwerten. Hierzu soll die Anfrage möglichst nah am Sensor (vor)ausgewertet werden [AH19]. Das Projekt ProSA beschäftigt sich mit dem Forschungsdatenmanagement. Eine Anwendung hierbei ist das Bestimmen einer

[1] Git-Hub der Llunatik Implementation: `https://github.com/donatellosantoro/Llunatic`

minimalen Teildatenbank, welche für das Ergebnis einer Anfrage relevant ist[AH19]. Zusätzlich zur reinen Chase-Implementation kann ChaTEAU vor der Ausführung vom Chase mehrere Terminierungskriterien testen und so eine Terminierung vor der Ausführung sicherstellen. ChaTEAU kann über die Konsole oder eine GUI verwendet werden. Außerdem bietet ChaTEAU über eine API die Möglichkeit in verschiedenen Anwendungen verwendet zu werden[ASG+22].

XML-Eingabeformat In diesem Abschnitt wollen wir das XML-Eingabeformat von ChaTEAU vorstellen. Hierzu stellen wir die einzelnen Elemente vor und geben kleine Beispiele. Im Anhang A geben wir vollständige Beispieldokumente an. ChaTEAU erhält die Eingabe in einem XML-Format. Das Root-Element des XML-Dokuments ist Input. Dieses enthält die Elemente Schema und Instanz. Das Schema-Element enthält die Relationsschemata und die Abhängigkeiten. Der grundlegende Aufbau der Eingabedatei ist im Listing 3.2 dargestellt. In dem Element RELATIONS werden die verschiedenen Relationen aufgelistet. Diese können als Ziel- oder Quellrelation markiert werden. Die Abhängigkeiten werden im Element DEPENDENCIES übergeben. Die Instanz über welche der Chase ausgeführt werden soll, wird in dem Element INSTANCE angegeben.

```
<imput>
        <schema>
                <relations>
                        ⋮
                </relations>
                <dependencies>
                        ⋮
                </dependencies>
        </schema>
        <instance>
                ⋮
        </instance>
</input>
```

Listing 3.2: Aufbau des XML-Eingabeformats für ChaTEAU.

Das Element RELATIONS enthält alle Relationen. Diese werden als RELATION-Elemente angegeben. Ein RELATION-Element hat das Attribut `Name` und das optionale Attribut `Tag`. Das Attribut `Name` gibt den Namen der Relation an. Das Attribut `Tag` wird benötigt, wenn ST-TGDs verwendet werden. Für Relationen über dem Quellschema setzen wir den `Tag` auf "S". Für Relationen über dem Zielschema verwenden wir den `Tag` "T". Mit den ATTRIBUTE-Elementen werden die Attribute der Relation angegeben. Diese Elemente haben zwei Attribute: `name` für den Attributnamen und `type` für den Attributtyp. In dieser Arbeit verwenden wir die Attributtypen "String" und "Int". Im Listing 3.3 geben wir ein Minimalbeispiel für zwei Tabellen.

```
<relations>
        <relation name="A" tag="S">
                <attribute name="a" type="int" />
        </relation>
        <relation name="B" tag="T">
                <attribute name="b" type="string" />
        </relation>
</relations>
```

Listing 3.3: Angabe der Relationen für ChaTEAU.

Als Abhängigkeiten können wir ChaTEAU TGDs, EGDs und ST-TGDs übergeben. Hierzu nutzen wir die Elemente TGD, EGD und STTGD. Diese haben die Unterelemente Body und Head, welche wiederum Atom-Elemente beinhalten. Ein Atom-Element hat das Attribut `Name`, um zu zeigen über welcher Relation die Bedingung formuliert ist. Im Body von TGDs und ST-TGDs können wir außerdem noch das Attribut `negation` verwenden und mit "true" belegen, um ein Atom zu negieren. Im Kopf einer EGD fällt das Attribut `Name` weg. Die Atom-Elemente beinhalten Variable-Elemente zum Darstellen der einzelnen Variablen. Diese haben die Attribute `Name`, `Type` und `Index`. Wenn eine Variable ausgezeichnet ist, setzen wir `Type` auf "V". Handelt es sich um eine existenzquantifizierte Variable, setzen wir den `Type` auf "E". In dem Listing 3.4 stellen wir als Beispiel eine ST-TGD dar.

```
<dependencies>
        <sttgd>
                <body>
                        <atom name="A">
                                <variable name="a" type="V" index="1" />
                        </atom>
                </body>
                <head>
                        <atom name="B">
                                <variable name="b" type="E" index="1" />
                        </atom>
                </head>
        </sttgd>
</dependencies>
```

Listing 3.4: Angabe einer ST-TGD in ChaTEAU.

Zum Abschluss stellen wir dar, wie das letzte Element, die Instanz, angegeben werden kann. Das Instance-Element enthält Atom-Elemente, welche die übergebenen Fakten darstellen. Ein Atom-Element hat das Attribut `Name`, welches angibt zu welcher Relation der Fakt gehört. In der Instanz enthält ein Atom-Element Constant- oder Null-Elemente. Ein Null-Element stellt einen Nullwert dar und hat als Attribute `Name` und `Index`. Ein Constant-Element hat die Attribute `Name` und `value`. In dem Listing 3.5 geben wir eine Beispielinstanz an.

```
<instance>
        <atom name="A">
                <constant name="a" value="123" />
        </atom>
        <atom name="B">
                <null name="b" index="1" />
        </atom>
</instance>
```

Listing 3.5: Angabe einer Instanz in ChaTEAU.

4. Eigene Konzeptentwicklung

Nachdem wir in Kapitel 3 bereits bestehende Forschung betrachtet haben, ziehen wir nun Rückschlüsse auf die Gegebenheiten in ChaTEAU und fassen unsere Erkenntnisse in Korollaren zusammen. Als Erstes betrachten wir im Abschnitt 4.1 ob SO-TGDs in ChaTEAU entstehen können und wie wir in ChaTEAU mit diesen umgehen können. Hierzu betrachten wir im Abschnitt 4.1.1, ob SO-TGDs bei der Invertierung von ST-TGDs entstehen können. Im darauf folgenden Abschnitt 4.1.2 betrachten wir die Invertierung von SO-TGDs. Danach betrachten wir im Abschnitt 4.1.3, wann bei der Komposition SO-TGDs entstehen können. Den Themenbereich SO-TGDs schließen wir mit dem Abschnitt 4.1.4 ab, in dem wir betrachten, wie SO-TGDs sich in ChaTEAU umsetzen lassen. Abschließend werden wir im Abschnitt 4.2 Skolemfunktionen in ChaTEAU umsetzen.

4.1. SO-TGDs in ChaTEAU

In [FKPT05] wurden SO-TGDs eingeführt, um die Komposition von Abbildungen immer darzustellen. In diesem Abschnitt werden wir die Fragen klären:

- Können SO-TGDs auch bei der Invertierung von ST-TGDs auftreten? (Abschnitt 4.1.1)
- Welche Form hat die Chase-Inverse einer SO-TGD? (Abschnitt 4.1.2)
- Bei welchen Kompositionen können SO-TGDs auftreten? (Abschnitt 4.1.3)
- Können wir SO-TGDs in ChaTEAU darstellen? (Abschnitt 4.1.4)

4.1.1. Invertierung von ST-TGDs

Um zu entscheiden, ob bei der Invertierung von Abbildungen SO-TGDs entstehen können, schauen wir uns an, welche Art von Inversen bei der Invertierung von GAV-, LAV- und GLAV-Abbildungen entstehen können. Im Folgenden betrachten wir nur die Chase-Inversen:

- **Invertierung einer GLAV:** Die Invertierung von GLAV-Abbildungen wurde bereits in [FKPT11b] beschrieben. Satz 3.8 besagt, dass, wenn für eine GLAV Abhängigkeit eine Inverse existiert, diese als GAV-Abbildung ausgedrückt werden kann.
- **Invertierung einer GAV:** Da nach Definition 2.6 alle ST-TGDs GLAV-Abbildungen sind und eine GAV-Abbildung nur eine spezielle Form einer ST-TGD darstellt, lässt sich Satz 3.8 auch auf GAV-Abbildungen übertragen. Anhand dessen leiten wir das Korollar 4.1 her.

  Korollar 4.1. *Sei M eine GAV-Schemaabbildung. Wenn M eine Chase-Inverse hat, dann hat M eine GAV Chase-Inverse.*
- **Invertierung einer LAV:** Da nach Definition 2.6 alle ST-TGDs auch GLAV-Abbildungen sind und eine LAV nur eine spezielle Form einer ST-TGD darstellt, lässt sich Satz 3.8 auch auf LAV-Abbildungen übertragen. Hieraus leiten wir das Korollar 4.2 her.

Korollar 4.2. *Sei M eine LAV-Schemaabbildung. Wenn M eine Chase-Inverse hat, dann hat M eine GAV Chase-Inverse.*

Nachdem wir die möglichen Fälle betrachtet haben, können wir feststellen, dass, falls eine Chase-Inverse existiert, diese sich als eine GAV-Abbildung darstellen lässt. Hieraus leiten wir folgendes Korollar her:

Korollar 4.3. *Sei M eine LAV-, GAV- oder GLAV-Schemaabbildung. Wenn M eine Chase-Inverse hat, dann hat M eine GAV Chase-Inverse.*

Aus dieser Erkenntnis schließen wir, dass bei der Erstellung von Chase-Inversen in ChaTEAU keine SO-TGDs entstehen können. Abschließend geben wir ein Beispiel für die Invertierung einer GLAV-Abbildung.

Beispiel 4.1 (Beispiel für die Invertierung eine GLAV-Abbildung)**.** *In unserem Beispiel existiert eine Abbildung $M = (S_1, S_2, \Sigma)$, welche die* GRADES*-Relation aufspaltet, sodass die Informationen eines Studierenden und die der Prüfung in getrennten Relationen sind. Damit weiterhin festgestellt werden kann, welche Note zu welchem Fach gehört, wird das gemeinsame Attribut* `pid` *eingeführt:*

$$\Sigma = \{\text{GRADES}(\texttt{co}, \texttt{id}, \texttt{sem}, \texttt{gr}) \rightarrow \exists \texttt{Pid} : \text{STUDGRAD}(\texttt{Pid}, \texttt{id}, \texttt{gr}) \wedge \text{EXAM}(\texttt{Pid}, \texttt{co}, \texttt{sem})\}.$$

Die Ausgangsinstanz unseres Beispieles ist gegeben durch:

$$I_1 = \{\text{GRADES}(\texttt{001}, \texttt{2}, \texttt{SS16}, \texttt{1.7}), \text{GRADES}(\texttt{001}, \texttt{5}, \texttt{SS16}, \texttt{3.0}),$$
$$\text{GRADES}(\texttt{002}, \texttt{1}, \texttt{WS15/16}, \texttt{3.7})\}.$$

Nachdem die Abbildung angewendet wurde, befinden sich in der Zielinstanz I_2 folgende Werte:

$$I_2 = \{\text{STUDGRAD}(\eta_1, \texttt{2}, \texttt{1.7}), \text{EXAM}(\eta_1, \texttt{001}, \texttt{SS16}),$$
$$\text{STUDGRAD}(\eta_2\ \texttt{1}, \texttt{3.7}), \text{EXAM}(\eta_2, \texttt{002}, \texttt{WS15/16}),$$
$$\text{STUDGRAD}(\eta_3, \texttt{5}, \texttt{3.0}), \text{EXAM}(\eta_3, \texttt{001}, \texttt{SS16})\}.$$

$M' = (S_2, S_1, \Sigma')$ ist eine Chase-Inverse für M und Σ' hat die Form:

$$\Sigma' = \{\text{STUDGRAD}(\texttt{pid}, \texttt{id}, \texttt{gr}) \wedge \text{EXAM}(\texttt{pid}, \texttt{co}, \texttt{sem}) \rightarrow \text{GRADES}(\texttt{co}, \texttt{id}, \texttt{sem}, \texttt{gr})\}.$$

Da M' sogar eine exakte Chase-Inverse ist, wird bei der Anwendung von M' auf I_2 die Instanz I_1 vollständig wieder hergestellt und enthält keine neuen Nullwerte. Dies wollen wir durch eine Rückabbildung zeigen. Wenn wir die Abbildung M' auf die Instanz I_2 anwenden, finden wir für diese folgende aktive Trigger:

$$\text{STUDGRAD}(\eta_1, \texttt{2}, \texttt{1.7}) \wedge \text{EXAM}(\eta_1, \texttt{001}, \texttt{SS16}) \rightarrow \text{GRADES}(\texttt{001}, \texttt{2}, \texttt{SS16}, \texttt{1.7}),$$
$$\text{STUDGRAD}(\eta_2\ \texttt{1}, \texttt{3.7}) \wedge \text{EXAM}(\eta_2, \texttt{002}, \texttt{WS15/16}) \rightarrow \text{GRADES}(\texttt{001}, \texttt{5}, \texttt{SS16}, \texttt{3.0}),$$
$$\text{STUDGRAD}(\eta_3, \texttt{5}, \texttt{3.0}) \wedge \text{EXAM}(\eta_3, \texttt{001}, \texttt{SS16}) \rightarrow \text{GRADES}(\texttt{002}, \texttt{1}, \texttt{WS15/16}, \texttt{3.7}).$$

Da keine weiteren Trigger existieren und durch die Anwendung der bereits gefundenen Trigger alle Tupel aus I_1 rekonstruiert wurden, gilt: $I_1 = Chaes_{M'}(I_2)$. Was bedeutet, dass M' eine exakte Chase-Inverse von M ist.

4.1.2. Invertierung von SO-TGDs

Nachdem wir im Abschnitt 4.1.1 die Invertierung von GLAV-Abbildungen betrachtet haben, betrachten wir in diesem Abschnitt darauf aufbauend die Chase-Inverse von SO-TGDs. In [FKMP05] wird beschrieben, dass jede SO-TGD äquivalent zu einer Komposition einer endlichen Menge von ST-TGDs ist. Aus dieser Erkenntnis ergibt sich die Frage, ob die Chase-Inverse einer SO-TGD die Komposition der Chase-Inversen der ST-TGDs ist. Die Inverse $(S \circ R)'$ ist nach [MM15] äquivalent zu $R' \circ S'$. Im Folgenden betrachten wir erst die exakte und danach die klassische Chase-Inverse.

Exakte Chase-Inverse Die exakte Chase-Inverse gibt im Gegensatz zur klassischen Chase-Inverse die Instanz exakt zurück. In Abbildung 4.1 wird dargestellt, wie eine Invertierung der Komposition $M_1 \circ M_2$ aussieht, wenn M_1' und M_2' exakte Chase-Inversen sind. Da die Inverse M_2' wieder auf I_2 abbildet, ist die Inverse von $M_1 \circ M_2$ durch $M_2' \circ M_1'$ gegeben. Bei der exakten Chase-Inverse ist einfach zu erkennen,

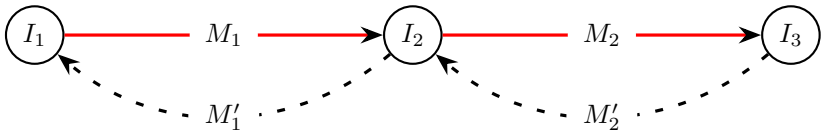

Abbildung 4.1.: Exakte Chase-Inverse für die Komposition von zwei Abbildungen.

dass die Inverse der Komposition $M_1 \circ \ldots \circ M_n$ durch die Komposition der exakten Chase-Inversen $M_n' \circ \ldots \circ M_1'$ gegeben ist. Da nach Korollar 4.3 eine (exakte) Chase-Inverse einer ST-TGD immer durch eine GAV-Abbildung dargestellt werden kann und die Komposition von GAV-Abbildungen durch eine GAV-Abbildung dargestellt werden kann, ist auch die exakte Chase-Inverse der Komposition eine GAV-Abbildung. Unsere Erkenntnisse fassen wir im Korollar 4.4 zusammen:

Korollar 4.4. *Sei eine Komposition $M_1 \circ \ldots \circ M_n$ von Abbildungen bestehend aus ST-TGDs gegeben und existiere für alle Abbildungen M_i mit $i \in \{1, \ldots, n\}$ eine exakte Chase-Inverse M_i', dann kann die exakte Chase-Inverse durch eine GAV-Abbildung M' mit $M' = M_n' \circ \ldots \circ M_1'$ beschrieben werden.*

Klassische Chase-Inverse Die klassische Chase-Inverse bildet den allgemeineren Fall der exakten Chase-Inversen. Im Gegensatz zur exakten Chase-Inversen wird bei der klassischen Chase-Inversen die Quelle nur auf homomorphe Äquivalenz rekonstruiert. Im Folgenden zeigen wir, dass auch für die klassische Chase-Inverse die Komposition von Abbildungen durch die Komposition der Inversen der Abbildungen dargestellt werden kann. Hierfür betrachten wir als Erstes den Fall, dass die SO-TGD durch eine Komposition von zwei Abbildungen dargestellt werden kann. Danach betrachten wir den allgemeineren Fall, dass eine Komposition von n Abbildungen invertiert wird.

In der Abbildung 4.2 ist die Invertierung der Komposition von zwei durch ST-TGDs gegebenen Abbildungen M_1 und M_2 dargestellt. M_1 ist eine Abbildung von der Instanz I_1 zur Instanz I_2 und M_2 eine Abbildung von der Instanz I_2 zur Instanz I_3. Die Chase-Inversen der Abbildungen sind M_1' und M_2'. Die durch die Anwendung von $M_1 \circ M_1'$ auf die Instanz I_1 entstandene Instanz bezeichnen wir mit I_1'. Diese ist nach der Definition der klassischen Chase-Inverse äquivalent zur Instanz I_1, dargestellt durch den Homomorphismus h_1. Der Homomorphismus h_2 beschreibt die Äquivalenz zwischen I_2 und I_2', welche durch die Anwendung von $M_1 \circ M_1'$ auf die Instanz I_1 entsteht. Die Instanz I_1'' entsteht, indem erst die Abbildungen $M_1 \circ M_2$ und danach die Inversen $I_2' \circ I_1'$ angewandt werden, also die Instanz, welche wir bei der Invertierung der Komposition erhalten. Um zu zeigen, dass diese äquivalent zu I_1 ist, zeigen wir, dass die Homomorphismen h_3 und h_4 existieren.

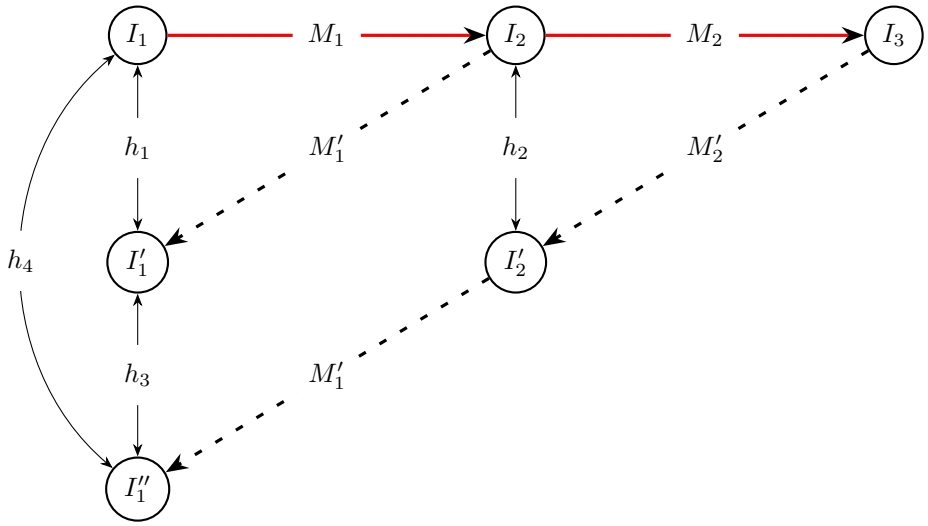

Abbildung 4.2.: Klassische Chase-Inverse für die Komposition von zwei Abbildungen.

In [FKPT11a] wird beschrieben, dass wenn ein Homomorphismus zwischen I und I' besteht und Σ nur ST-TGDs enthält, auch ein Homomorphismus zwischen $Chase_\Sigma(I)$ und $Chase_\Sigma(I')$ besteht. Da zwischen den Instanzen I_2 und I_2' ein Homomorphismus in beide Richtungen existiert und für die Instanzen $I_1' = Chase_{M_1'}(I_2)$ und $I_1'' = Chase_{M_1'}(I_2')$ gilt, können wir mit dieser Aussage schlussfolgern, dass auch zwischen I_1' und I_1'' ein Homomorphismus in beide Richtungen besteht, den wir mit h_3 bezeichnen.
Nun können wir zeigen, dass zwischen I_1 und I_1'' in beide Richtungen ein Homomorphismus besteht. Hierzu stellen wir den Homomorphismus als Menge von Tupeln dar. Ein Tupel stellt dar, wie ein Wert durch den Homomorphismus abgebildet wird. Wenn ein Homomorphismus h zum Beispiel den Nullwert η_3 auf das Semester `WS15/16` abbildet, so kann dies durch $(\eta_3, \texttt{WS15/16}) \in h$ dargestellt werden. Wenn wir diese Schreibweise nutzen, können wir den Homomorphismus h_4 von I_1'' nach I_1 durch den Ausdruck

$$h_4 = \{(A, C) | \exists (A, B) \in h_3 \wedge \exists (B, C) \in h_1\}$$

beschreiben. Mit demselben Vorgehen lässt sich auch der Homomorphismus von I_1 nach I_1'' beschreiben. Aus unseren Beobachtungen folgern wir das Korollar 4.5.

Korollar 4.5. *Seien $M_1 = (S_1, S_2, \Sigma_1)$ und $M_2 = (S_2, S_3, \Sigma_2)$ zwei aufeinanderfolgende Schemaabbildungen mit Σ_1 und Σ_2 Mengen von ST-TGDs, so lässt sich die Chase-Inverse der Komposition darstellen durch:*

$$(M_1 \circ M_2)' = M_2' \circ M_1'.$$

Da M_1' und M_2' nach Korollar 4.3 GAV-Abbildungen sind, ist nach Satz 3.1 auch $(M_1 \circ M_2)'$ eine GAV-Abbildung.

Das Korollar 4.5 lässt sich erweitern, sodass wir eine Aussage über die Invertierung der Komposition von n Abbildungen treffen können. Haben wir die Komposition $M_1 \circ \ldots \circ M_n$ gegeben, können wir ihre Inverse durch $M_n' \circ (M_1 \circ \ldots \circ M_{n-1})'$ darstellen. Dadurch, dass das Korollar den Fall $n = 2$ beschreibt, können wir folgende Aussage treffen:

$$(M_1 \circ \ldots \circ M_n)' = M_n' \circ \ldots \circ M_1'$$

Da nach Korollar 4.3 die Chase-Inverse einer Abbildung immer als GAV dargestellt werden kann und nach Satz 3.1 die Komposition von GAV-Abbildungen sich durch eine GAV-Abbildung darstellen lässt, ist auch die Komposition der Inversen immer eine GAV-Abbildung. In Korollar 4.6 fassen wir unsere Beobachtung über die Chase-Inverse für eine SO-TGD zusammen.

Korollar 4.6. *Sei eine SO-TGD M gegeben, so existiert Komposition $M_1 \circ \ldots \circ M_n$ von Abbildungen, bestehend aus ST-TGDs, welche zu dieser äquivalent sind. Existiert für alle Abbildungen M_i mit $i \in \{1, \ldots, n\}$ eine Chase-Inverse M_i', so kann die Inverse der SO-TGD durch eine GAV-Abbildung $M' = M_n' \circ \ldots \circ M_1'$ beschrieben werden.*

Beispiel 4.2. *Als Beispiel für die Invertierung einer SO-TGD wollen wir die SO-TGD aus dem Beispiel 2.6 invertieren. Hierbei schränken wir die* STUDENT*-Relation auf das Attribut* id *ein, da die erste Abbildung sonst keine Chase-Inverse hat. Mit dieser Einschränkung erhalten wir die SO-TGD*

$$\begin{aligned}\exists f : (&\mathrm{STUDENT}(id) \rightarrow \mathrm{VERTRETER}(id, f(id)),\\ &\mathrm{STUDENT}(id) \wedge (id = f(id)) \rightarrow \mathrm{SELVERT}(id)).\end{aligned}$$

Diese können wir durch die beiden Abbildungen $M_1 = (S_1, S_2, \Sigma_1)$ und $M_2 = (S_2, S_3, \Sigma_2)$ darstellen. Σ_1 und Σ_2 sind gegeben durch:

$$\begin{aligned}\Sigma_1 &= \{\mathrm{STUDENT}(id) \rightarrow \exists Sprid : \mathrm{VERTRETER}(id, Sprid)\}\\ \Sigma_2 &= \{\mathrm{VERTRETER}(id, sprid) \rightarrow \mathrm{VERTRETER}(id, sprid),\\ &\qquad \mathrm{VERTRETER}(id, id) \rightarrow \mathrm{SELVERT}(id)\}.\end{aligned}$$

Um die Chase-Inverse der SO-TGD zu bestimmen, bestimmen wir als erstes die Inverse der Abbildungen M_1 und M_2. Die Inversen bezeichnen wir mit $M_1' = (S_2, S_1, \Sigma_1')$ und $M_2' = (S_3, S_2, \Sigma_2')$. Σ_1' und Σ_2' sind gegeben durch:

$$\begin{aligned}\Sigma_1' &= \{\mathrm{VERTRETER}(id, sprid) \rightarrow \mathrm{STUDENT}(id)\}\\ \Sigma_2' &= \{\mathrm{VERTRETER}(id, sprid) \rightarrow \mathrm{VERTRETER}(id, sprid)\}.\end{aligned}$$

Um die Inverse der SO-TGD zu bestimmen, führen wir als letzten Schritt die Komposition $M_2' \circ M_1'$ durch. Hierdurch erhalten wir die Inverse $M_2' \circ M_1' = (S_3, S_1, \Sigma')$. Σ' ist gegeben durch:

$$\Sigma' = \{\mathrm{VERTRETER}(id, sprid) \rightarrow \mathrm{STUDENT}(id)\}.$$

4.1.3. Komposition von Abbildungen

Die Komposition von zwei Schemaabbildungen lässt sich nicht immer durch GLAV-Abbildungen darstellen. Um die Komposition trotzdem immer darstellen zu können, wurden in [FKPT05] die SO-TGDs eingeführt. In welchen Fällen SO-TGDs benötigt werden und wann eine Komposition durch GLAV-Abbildungen dargestellt werden kann, wurde für bestimmte Fälle bereits betrachtet. Im Folgenden schauen wir uns die möglichen Kombinationen an und stellen dar, welche Art von Abbildung benötigt wird, um die Komposition darzustellen. Hierzu nutzen wir Sätze aus Abschnitt 3.1.1 und geben für manche Fälle Beispiele. Für die Fälle, die nicht bereits durch einen Satz beschrieben sind, erstellen wir jeweils ein Korollar. In Tabelle 4.1 sind die Ergebnisse der einzelnen Fälle dargestellt, welche wir im Folgenden betrachten.

$\circ$	GAV	LAV	GLAV
GAV	GAV	GLAV	GLAV
LAV	SO-TGD	SO-TGD	SO-TGD
GLAV	SO-TGD	SO-TGD	SO-TGD

Tabelle 4.1.: Ergebnisse bei der Komposition von verschiedenen Arten von Abbildungen.

GAV ∘ GAV

Die Komposition von zwei GAV-Abbildungen lässt sich nach Satz 3.1 durch eine GAV-Abbildung darstellen. Im Folgenden geben wir ein kurzes Beispiel für die Komposition von zwei GAV-Abbildungen.

Beispiel 4.3 (Komposition GAV ∘ GAV)**.** *In der ersten Abbildung wird eine Projektion auf der* STUDENT- *und der* GRADES-*Relation durchgeführt. In der zweiten Abbildung werden die beiden Relationen verbunden, sodass eine Relation mit Namen und Noten von Studierenden entsteht:*

$$\Sigma_1 = \{\text{STUDENT}(id, ln, fn, st) \rightarrow \text{STU}(id, ln, fn), \text{GRADES}(co, id, sem, gr) \rightarrow \text{GRA}(id, gr)\}$$
$$\Sigma_2 = \{\text{GRA}(id, gr) \wedge \text{STU}(id, ln, fn) \rightarrow \text{STUGRA}(ln, fn, gr)\}.$$

Die Komposition der Schemaabbildungen $M_{1,2}$ und $M_{2,3}$ kann durch die GAV-Abbildung $M_{1,3}$ mit den Abhängigkeiten Σ dargestellt werden:

$$\Sigma = \{\text{STUDENT}(id, ln, fn, st) \wedge \text{GRADES}(co, id, sem, gr) \rightarrow \text{STUGRA}(ln, fn, gr)\}.$$

GAV ∘ LAV

Für den Fall der Komposition, dass die erste Abbildung eine GAV-Abbildung ist, wird in Satz 3.2 beschrieben, dass diese durch eine GLAV-Abbildung dargestellt werden kann. Der Satz beschreibt aber nur die Komposition GAV ∘ GLAV, also den allgemeinsten Fall an ST-TGDs als zweite Abbildung. Im Folgenden werden wir am Beispiel zeigen, dass auch bei dem speziellen Fall der Komposition von GAV- mit LAV-Abbildungen eine GLAV-Abbildung entstehen kann und somit keine weitere Einschränkung auf LAV oder GAV möglich ist.

Beispiel 4.4 (Komposition GAV ∘ LAV)**.** *Die erste Abbildung des Beispiels ist eine GAV, welche eine Tabelle* STUGRA *mit Vor- und Nachname eines Studierenden und seiner Note erzeugt. In der zweiten Abbildung wird diese in zwei Tabellen aufgespalten und eine neue Studenten-Id hinzugefügt:*

$$\Sigma_1 = \{\text{STUDENT}(id, ln, fn, st) \wedge \text{GRADES}(co, id, sem, gr) \rightarrow \text{STUGRA}(ln, fn, gr)\}$$
$$\Sigma_2 = \{\text{STUGRA}(ln, fn, gr) \rightarrow \exists Sid : \text{STU}(Sid, ln, fn) \wedge \text{GRA}(Sid, gr)\}.$$

Die Komposition der Abbildungen enthält die Abhängigkeiten Σ, mit:

$$\Sigma = \{\text{STUDENT}(id, ln, fn, st) \wedge \text{GRADES}(co, id, sem, gr)$$
$$\rightarrow \exists Sid : \text{STU}(Sid, ln, fn) \wedge \text{GRA}(Sid, gr)\}.$$

Im Beispiel 4.4 ist zu erkennen, dass eine Einschränkung darauf, dass die Komposition sich durch eine GAV-Abbildung darstellen lässt, nicht möglich ist, da das in der zweiten Abbildung hinzugefügte Attribut in der Komposition existenzquantifiziert auftaucht. Eine Einschränkung darauf, dass die Komposition sich durch eine LAV-Abbildung darstellen lässt, ist auch nicht möglich, da im Rumpf der GAV-Abbildung mehrere Prädikate vorkommen können. Diese können, wie in unserem Beispiel, in der Komposition wieder auftauchen. Aus dieser Beobachtung leiten wir das Korollar 4.7 her.

Korollar 4.7. *Seien $M_1 = (S_1, S_2, \Sigma_1)$ und $M_2 = (S_2, S_3, \Sigma_2)$ zwei Schemaabbildungen. Sei weiter Σ_1 eine Menge von GAV-Abbildungen und Σ_2 eine Menge von LAV-Abbildungen, so kann die Komposition $M_1 \circ M_2$ im allgemeinen durch eine GLAV-Abbildung dargestellt werden.*

GAV ∘ GLAV

Auch das Ergebnis der Komposition von einer GAV- und einer GLAV-Abbildung wurde bereits beschrieben. Die Sätze 3.1 und 3.2 besagen, dass die Komposition GAV ∘ GLAV durch eine GLAV-Abbildung dargestellt werden kann. Im Folgenden geben wir ein Beispiel für die Komposition, welches eine Erweiterung des Beispiels 4.3 ist.

Beispiel 4.5 (Komposition GAV ∘ GLAV)**.** *Die Abbildungen ähneln denen aus dem Beispiel 4.3. Der einzige Unterschied ist, dass die durch die zweite Abbildung entstandene Relation außerdem noch ein neues Attribut* `Pid` *enthält:*

$$\Sigma_1 = \{\text{STUDENT}(id, ln, fn, st) \rightarrow \text{STU}(id, ln, fn), \text{GRADES}(co, id, sem, gr) \rightarrow \text{GRA}(id, gr)\}$$
$$\Sigma_2 = \{\text{GRA}(id, gr) \wedge \text{STU}(id, ln, fn) \rightarrow \exists Pid : \text{STUGRA}(Pid, ln, fn, gr)\}.$$

Die Komposition der Abbildungen enthält die Abhängigkeiten Σ*, mit:*

$$\Sigma = \{\text{STUDENT}(id, ln, fn, st) \wedge \text{GRADES}(co, id, sem, gr) \rightarrow \exists Pid : \text{STUGRA}(Pid, ln, fn, gr)\}.$$

LAV ∘ GAV

Die Komposition einer LAV- und GAV-Abbildung kann nach Satz 3.7 auf jeden Fall durch eine SO-TGD dargestellt werden. Wenn für die Darstellung der Komposition LAV ∘ GAV im allgemeinen eine SO-TGD benötigt wird, können wir ein Beispiel geben in dem dies der Fall ist.

Beispiel 4.6. *Für das Beispiel orientieren wir uns an dem in [FKMP05] gegebenen Beispiel. Hier entsteht die Notwendigkeit für eine SO-TGD durch das mehrfache Auftreten einer Variable im Rumpf der zweiten Schemaabbildung. Übertragen wir dies nun auf unser Studierenden-Beispiel.*
In der ersten Abbildung wird jedem Studierenden ein Vertreter zugeordnet. In der zweiten Abbildung wird eine Tabelle erstellt mit allen Studierenden, welche sich selber vertreten:

$$\Sigma_1 = \{\text{STUDENT}(id, ln, fn, st) \rightarrow \exists Sprid : \text{VERTRETER}(id, Sprid)\}$$
$$\Sigma_2 = \{\text{VERTRETER}(id, id) \rightarrow \text{SELVERT}(id)\}.$$

Die Komposition der beiden Schemaabbildungen kann durch folgende SO-TGD dargestellt werden:

$$\exists f : \text{STUDENT}(id, ln, fn, st) \wedge id = f(id) \rightarrow \text{SELVERT}(id).$$

Aus Satz 3.7 und Beispiel 4.6 leiten wir das Korollar 4.8 her.

Korollar 4.8. *Seien* $M_1 = (S_1, S_2, \Sigma_1)$ *und* $M_2 = (S_2, S_3, \Sigma_2)$ *zwei Schemaabbildungen. Sei weiter* Σ_1 *eine Menge von LAV-Abbildungen und* Σ_2 *eine Menge von GAV-Abbildungen, so kann die Komposition* $M_1 \circ M_2$ *im Allgemeinen nur durch eine SO-TGD dargestellt werden.*

LAV ∘ LAV

Für die Komposition von zwei strikten LAV-Abbildungen wurde in [AFM10] gezeigt, dass sich diese durch eine LAV-Abbildung darstellen lässt. Welche Art von Abhängigkeit bei der Komposition von zwei nicht strikten LAV-Abbildungen entsteht, wurde noch nicht genauer beschrieben. Satz 3.7 besagt, dass bei der Komposition von GLAV-Abbildungen höchstens eine SO-TGD entstehen kann. In dem für die Komposition LAV ∘ GAV genutzten Beispiel 4.6 ist die zweite Abbildung $M_{2,3}$ sowohl eine LAV als auch eine GAV-Abbildung. Deswegen können wir das Beispiel auch für die Komposition von zwei LAV-Abbildungen nutzen. Aus Satz 3.7 und Beispiel 4.6 leiten wir folgendes Korollar her:

Korollar 4.9. *Seien zwei Schemaabbildungen $M_1 = (S_1, S_2, \Sigma_1)$ und $M_2 = (S_2, S_3, \Sigma_2)$ gegeben. Seien weiter Σ_1 und Σ_2 Mengen von LAV-Abbildungen, so kann die Komposition $M_1 \circ M_2$ im Allgemeinen nur durch eine SO-TGD dargestellt werden.*

LAV ∘ GLAV

Da jede LAV- und GAV-Abbildung auch eine GLAV-Abbildung ist, lässt sich aus den Korollaren 4.9 und 4.8 sowie Satz 3.7 herleiten, dass im allgemeinen Fall die Komposition LAV ∘ GLAV nur durch eine SO-TGD dargestellt werden kann. Diese Eigenschaft formulieren wir im folgenden Korollar.

Korollar 4.10. *Seien zwei Schemaabbildungen $M_1 = (S_1, S_2, \Sigma_1)$ und $M_2 = (S_2, S_3, \Sigma_2)$ gegeben. Sei weiter Σ_1 eine Menge von LAV-Abbildungen und Σ_2 eine Menge von GLAV-Abbildungen, so kann die Komposition $M_1 \circ M_2$ im allgemeinen nur durch eine SO-TGD dargestellt werden.*

GLAV ∘ GAV

Die Komposition einer GLAV- und einer GAV-Abbildung bildet einen allgemeineren Fall der LAV ∘ GAV-Komposition. Diese lässt sich nach Korollar 4.8 nur durch eine SO-TGD darstellen. Aus dieser Eigenschaft und Satz 3.7 leiten wir her, dass auch für die Darstellung der Komposition GLAV ∘ GAV eine SO-TGD notwendig ist und formulieren dies in Korollar 4.11

Korollar 4.11. *Seien zwei Schemaabbildungen $M_1 = (S_1, S_2, \Sigma_1)$ und $M_2 = (S_2, S_3, \Sigma_2)$ gegeben. Sei weiter Σ_1 eine Menge von GLAV-Abbildungen und Σ_2 eine Menge von GAV-Abbildungen, so ist im Allgemeinen für die Darstellung der Komposition $M_1 \circ M_2$ eine SO-TGD notwendig.*

Da die Komposition GLAV ∘ GAV ein allgemeinerer Fall von bereits besprochenen Fällen ist, sind die Beispiele 4.6 und 4.4 auch für diesen Fall exemplarisch.

GLAV ∘ LAV

Die Komposition GLAV ∘ LAV bildet eine allgemeinere Variante von GAV ∘ LAV und LAV ∘ LAV. Da die Komposition von zwei LAV-Abbildungen im Allgemeinen sich nur durch eine SO-TGD darstellen lässt, kann auch die Komposition GLAV ∘ LAV im Allgemeinen nur durch eine SO-TGD dargestellt werden. Dies formulieren wir im folgenden Korollar.

Korollar 4.12. *Seien zwei Schemaabbildungen $M_1 = (S_1, S_2, \Sigma_1)$ und $M_2 = (S_2, S_3, \Sigma_2)$ gegeben. Sei weiter Σ_1 eine Menge von GLAV-Abbildungen und Σ_2 eine Menge von LAV-Abbildungen gegeben, so ist im Allgemeinen für die Darstellung der Komposition $M_1 \circ M_2$ eine SO-TGD notwendig.*

Die oben bereits gegebenen Beispiele 4.6 und 4.4 sind auch Beispiele für die Komposition GLAV $\circ$ LAV, wobei nur im Beispiel 4.6 eine SO-TGD benötigt wird.

GLAV $\circ$ GLAV

Die Komposition von zwei GLAV-Abbildungen bildet den allgemeinsten Fall der Komposition, welcher in Satz 3.7 beschrieben wird. Da bereits SO-TGDs bei der Komposition von speziellen ST-TGDs auftreten können, ist dies im allgemeinen Fall auch möglich.
Da es sich um den allgemeinsten Fall handelt, sind alle bisher gegebenen Beispiele auch ein Beispiel für die Komposition GLAV $\circ$ GLAV.

Zusammenfassung der Ergebnisse

Nachdem wir die einzelnen Fälle betrachtet haben, fassen wir in diesem Abschnitt die Ergebnisse zusammen. In Tabelle 4.1 geben wir eine Übersicht über die Aussagen unserer Korollare und der bereits vorhandenen Sätze zum Ergebnis der Komposition. Welche Art von Abhängigkeit durch die Komposition $f \circ g$ entsteht, formulieren wir außerdem in der folgenden Aussage. Die Komposition $f \circ g$ ist darstellbar durch eine Abhängigkeit des Typs

- GAV, wenn f und g GAV-Abhängigkeiten sind;
- GLAV, wenn f eine GAV und $g \in \{\text{LAV}, \text{GLAV}\}$ ist;
- SO-TGD, wenn $f \in \{\text{LAV}, \text{GLAV}\}$ und $g \in \{\text{LAV}, \text{GAV}, \text{GLAV}\}$ ist.

4.1.4. Umsetzung in ChaTEAU

Wie wir in Abschnitt 4.1.3 gesehen haben, können SO-TGDs bei der Komposition von Abbildungen entstehen. Hieraus ergibt sich die Frage, wie die Komposition und SO-TGDs in ChaTEAU dargestellt werden sollen. Eine Möglichkeit wäre ChaTEAU um SO-TGDs zu erweitern. Die zweite Variante ist, die bereits bestehenden Möglichkeiten in ChaTEAU zu nutzen. Da alle SO-TGDs einer endlichen Folge von Kompositionen entsprechen [FKMP05], entscheiden wir uns dafür, SO-TGDs über die Komposition von GLAV-Abbildungen darzustellen und diese mit den bestehenden Möglichkeiten in ChaTEAU zu realisieren.
Da das Eingabeformat von ChaTEAU verlangt, dass eine Relation im Ziel- oder Quellschema liegt, können momentan keine ST-TGDs hintereinander ausgeführt werden. Eine Möglichkeit ist, das Eingabeformat so zu erweitern, dass "Zwischenschemata" möglich sind. Eine zweite Möglichkeit ist, die bestehenden Funktionen von Chateau möglichst gut auszunutzen. Hierzu stellen wir die Relationen der Zwischenschemata als Hilfstabellen im Quellschema dar und übergeben die ST-TGDs als TGDs. Dies ist möglich, da sich ST-TGDs von TGDs nur durch die Eigenschaft unterscheiden, dass bei ST-TGDs Ziel- und Quellschema nicht gleich sind. Hierbei ist es wichtig, dass die Namen der Relationen eindeutig sind, da sonst die Terminierung gefährdet werden kann. Wir entscheiden uns für die zweite Variante. Im Folgenden beschreiben wir, wie wir eine Folge von ST-TGDs ChaTEAU übergeben können, ohne das Eingabeformat zu erweitern.

Folge von ST-TGDs in ChaTEAU Wir stellen eine Folge von Abbildungen $M_1 \circ \ldots \circ M_n$ dar. Jede Abbildung M_i mit $i \in 1, \ldots, n$ kann aus mehreren ST-TGDs bestehen und bildet von der Instanz I_{i-1} auf die Instanz I_i ab. Das erste Schema bezeichnen wir also mit I_0 und das Zielschema mit I_n. Zur Übergabe der ST-TGD führen wir folgende Schritte durch:

1. Als erstes stellen wir sicher, dass die Bezeichnungen der Relationen eindeutig sind. Wenn ein Name in zwei verschiedenen Schemata vorkommt, ändern wir den Namen so, dass dieser wieder eindeutig ist.

2. Die Relationen aus dem Zielschema I_n übergeben wir als Zielrelationen, indem wir den Tag auf "T" setzen. Die Relationen der Schemata I_i mit $i < n$ übergeben wir als Quellrelationen, indem wir den Tag auf "S" setzen.

3. Als nächsten Schritt übergeben wir die Abbildungen. Die ST-TGDs der Abbildung M_n übergeben wir als ST-TGD. Die ST-TGDs aus den Abbildungen M_i mit $i < 1$ übergeben wir als TGD. Hierzu ändern wir den Namen des XML-Elementes von "sttgd" zu "tgd".

Die Relationen der Ziel- und Zwischenschemata sollten leer übergeben werden. Außerdem sollte beachtet werden, dass über dem Quellschema formulierte Integritätsbedingungen gegebenenfalls nicht vor der Abbildung ausgeführt werden. Wenn vor der Abbildung Integritätsbedingungen eingearbeitet werden sollen, können wir das realisieren, indem wir ChaTEAU zweimal aufrufen: das erste Mal um die Integritätsbedingungen einzuarbeiten und ein zweites Mal um die Abbildung auszuführen.

Beispiel 4.7. *Als Beispiel für die Übergabe einer Folge von ST-TGDs übergeben wir die Abbildungen aus dem Beispiel 2.6 an ChaTEAU. Wir wählen dieses Beispiel, da die Komposition der ST-TGDs eine SO-TGD ergeben. Die zu übergebenden Abbildungen sind:*

$$M_{1,2} = \{\text{STUDENT}(id, ln, fn, st) \rightarrow \exists Sprid : \text{VERTRETER\_Z}(id, Sprid)\}$$
$$M_{2,3} = \{\text{VERTRETER\_Z}(id, sprid) \rightarrow \text{VERTRETER}(id, sprid),$$
$$\text{VERTRETER\_Z}(id, id) \rightarrow \text{SELVERT}(id)\}.$$

Um den ersten Punkt zu erfüllen, haben wir in der Auflistung der Abhängigkeiten die Relation VERTRETER *bereits in* VERTRETER_Z *umbenannt, wodurch jede Relation einen eindeutigen Namen hat. Als zweiten Schritt tragen wir die Relationen im XML-Dokument ein.*

```
<relations>
        <relation name="STUDENT" tag="S">
                <attribute name="student_id" type="int" />
                <attribute name="lastname" type="string" />
                <attribute name="firstname" type="string" />
                <attribute name="study_course" type="string" />
        </relation>
        <relation name="Vertreter_Z" tag="S">
                <attribute name="student_id" type="int" />
                <attribute name="sprecher_id" type="int" />
        </relation>
        <relation name="VERTRETER_Z" tag="T">
                <attribute name="student_id" type="int" />
                <attribute name="sprecher_id" type="int" />
        </relation>
        <relation name="SELVERT" tag="T">
                <attribute name="id" type="int" />
        </relation>
</relations>
```

Im dritten Schritt geben wir die ST-TGDs an. Die ST-TGDs der ersten Abbildung geben wir als TGD an. Die ST-TGDs der zweiten Abbildung geben wir als ST-TGDs an. Beispielhaft geben wir hier die erste ST-TGD der zweiten Abbildung an.

```
<dependencies>
        <tgd>
                <body>
                        <atom name="STUDENT">
                                <variable name="student_id" type="V" index="1" />
                                <variable name="lastname" type="V" index="1" />
                                <variable name="firstname" type="V" index="1" />
                                <variable name="study_course" type="V" index="1" />
                        </atom>
                </body>
                <head>
                        <atom name="VERTRETER_Z">
                                <variable name="student_id" type="V" index="1" />
                                <variable name="sprecher_id" type="E" index="1" />
                        </atom>
                </head>
        </tgd>
        <sttgd>
                <body>
                        <atom name="VERTRETER_Z">
                                <variable name="student_id" type="V" index="1" />
                                <variable name="student_id" type="V" index="1" />
                        </atom>
                </body>
                <head>
                        <atom name="SELVERT">
                                <variable name="student_id" type="V" index="1" />
                        </atom>
                </head>
        </sttgd>
</dependencies>
```

Nachdem die Relationsschemata und die Abhängigkeiten angegeben wurden, kann im letzten Schritt die Instanz angegeben werden. Das vollständige XML-Dokument mit einer Beispielinstanz befindet sich im Anhang A.1.

4.2. Skolemfunktionen für ChaTEAU

In Abschnitt 2.3.1 wurde die Skolemisierung von Abhängigkeiten vorgestellt. Um skolemisierte Abhängigkeiten in ChaTEAU verwenden zu können, werden Skolemfunktionen benötigt. Für die Umsetzung von Skolemfunktionen gibt es zwei Möglichkeiten. Die erste Möglichkeit ist die Implementation von ChaTEAU um Skolemfunktionen zu erweitern. Die zweite Möglichkeit ist, Skolemfunktionen mit der bestehenden Implementation umzusetzen. Hierbei ergeben sich drei Versionen. Die Nutzung von Hilfstabellen, die Nutzung von EGDs und die Nutzung von EGDs über Hilfstabellen zur Darstellung von Skolemfunktionen. Im Folgenden betrachten wir die beiden Umsetzungen mit der bestehenden Implementation.

4.2.1. Darstellung in Hilfstabelle

Die Idee bei der Darstellung von Skolemfunktionen als Hilfstabelle ist, dass in dieser für jede Eingabe eine Ausgabe hinterlegt wird. Die möglichen Eingabeparameter lassen sich aus der Instanz auslesen. Als Ausgabe nutzen wir Nullwerte, welche sich durch eine existensquantifizierte Variable erzeugen lassen. Diese Darstellung ähnelt der aus [HY90], da wir jeden Skolemterm auf einen Nullwert abbilden. Wir unterteilen unser Vorgehen in zwei Schritte. Im ersten Schritt wollen wir die Hilfstabelle erzeugen. Im zweiten Schritt nutzen wir diese um die skolemisierte TGD anzuwenden.
Soll ChaTEAU eine skolemisierte TGD t (Definition 2.7) übergeben werden, so wird im ersten Schritt eine Hilfstabelle für jede Skolemfunktion erstellt. Die Hilfstabelle einer Skolemfunktion $f(w)$ muss alle Attribute w und einen Nullwert als Ausgabe enthalten. Hat die Skolemfunktion $w_1, \ldots, w_n$ als Eingabeparameter, so hat die Hilfstabelle die Form $\mathrm{H}(w_1, \ldots, w_n, \eta)$, wobei η den ausgegebenen Nullwert darstellt. Für jede in der Abhängigkeit vorkommende Skolemfunktion $f_i(w)$ mit $i \in \{1 \ldots j\}$ erzeugen wir eine TGD der Form

$$\phi(x) \wedge \neg H_i(w, c) \rightarrow \exists N : H_i(w, N)),$$

wobei $\phi(x)$ der Rumpf der skolemisierten TGD ist. Um zu verhindern, dass Eingabeparameter mehrfach eingetragen werden, befindet sich zusätzlich $\neg H_i(w, c)$ im Rumpf der TGD.
Im zweiten Schritt nutzen wir die Hilfstabellen, um die Skolemfunktionen zu ersetzen. Hierzu übergeben wir ChaTEAU die TGD

$$\phi(x) \wedge H_1(w_1, y_1) \wedge \ldots \wedge H_n(w_n, y_n) \rightarrow \psi(x, y).$$

Für jede Skolemfunktion f_i in t haben wir im Rumpf der TGD die dazugehörige Hilfstabelle $H_i(w_i, y_i)$ hinzugefügt. Im Kopf der TGD haben wir den Skolemterm $f_i(w)$ durch y_i ersetzt. Durch das Einsetzen der entsprechenden Nullwerte erreichen wir das gleiche Ergebnis wie bei der Verwendung von Skolemfunktionen. Im Algorithmus 2 ist das Vorgehen noch einmal als Pseudocode dargestellt.

Algorithmus 2 : Pseudocode zur Darstellung einer skolemisierten Abhängigkeit mit Hilfstabellen in ChaTEAU

Data : Skolemisierte Abhängigkeiten $t = \phi(x) \rightarrow \psi(x, f(w))$
Result : Modifizierte Abhängigkeit t', Menge an Abhängigkeiten A

```
1 A = ∅
2 t' = t
3 forall Skolemfunktion f_i(x) in t do
4     ersetze f_i(x) durch y_i
5     füge φ(x) ∧ ¬H_i(w, c) → ∃y_i : H_i(w, y_i)) zu A hinzu
6     füge H_i(w, y_i) zum Rumpf von t' hinzu
```

Um die Darstellung von Skolemfunktionen mit Hilfstabellen in ChaTEAU zu verdeutlichen, betrachten wir diese im Folgenden am Beispiel der im Beispiel 2.5 skolemisierten TGD.

Beispiel 4.8. *Für dieses Beispiel nutzen wir die TGD aus dem Beispiel 2.5 und schränken die Relation* STUDENT *auf die Attribute* `id` *und Studiengang* `st` *ein, sodass wir folgende Abhängigkeit erreichen:*

$$\mathrm{GRADES}(co, id, sem, gr) \rightarrow \mathrm{STUDENT}(id, f_{st}(id)).$$

Im ersten Schritt erzeugen wir zur Darstellung der Skolemfunktion f_{st} die Hilfstabelle HST*:*

$$\mathrm{GRADES}(co, id, sem, gr) \wedge \neg \mathrm{HST}(id, c) \rightarrow \exists N : \mathrm{HST}(id, N).$$

Nachdem die Hilfstabelle erstellt ist, nutzen wir diese, um die Skolemfunktion in der TGD darzustellen:

$$\text{GRADES}(co, id, sem, gr) \wedge \text{HST}(id, st) \rightarrow \text{STUDENT}(id, st).$$

Zum Abschluss des Beispiels betrachten wir, wie das Ergebnis bei einer Anwendung auf die Instanz

$$I_0 = \{\text{GRADES}(001, 1, SS16, 1.7), \text{GRADES}(001, 5, SS16, 3.0),\\ \text{GRADES}(002, 1, WS15/16, 3.7)\}$$

aussieht. Im ersten Schritt werden drei aktive Trigger für die erste Abhängigkeit gefunden. Nach der Anwendung des ersten Triggers erhalten wir die Instanz

$$I_1 = \{\text{GRADES}(001, 1, SS16, 1.7), \text{GRADES}(001, 5, SS16, 3.0),\\ \text{GRADES}(002, 1, WS15/16, 3.7), \text{HST}(1, \eta_1)\}.$$

Hiernach existiert für die erste TGD nur noch ein aktiver Trigger. Für die zweite TGD existieren zwei aktive Trigger. Im zweiten Schritt nutzen wir den letzten aktiven Trigger der ersten Abhängigkeit und erhalten die neue Instanz

$$I_2 = \{\text{GRADES}(001, 1, SS16, 1.7), GRADES(001, 5, SS16, 3.0),\\ \text{GRADES}(002, 1, WS15/16, 3.7), \text{HST}(1, \eta_1), \text{HST}(5, \eta_2)\}.$$

Auf dieser Instanz existieren nur noch aktive Trigger für die zweite TGD. Nach Anwendung der Trigger erhalten wir die Instanz

$$I_3 = \{\text{GRADES}(001, 1, SS16, 1.7), \text{GRADES}(001, 5, SS16, 3.0), \text{GRADES}(002, 1, WS15/16, 3.7),\\ \text{HST}(1, \eta_1), \text{HST}(5, \eta_2)\\ \text{STUDENT}(1, \eta_1), \text{STUDENT}(5, \eta_2)\}.$$

Darstellung in ChaTEAU Wollen wir eine skolemisierte Abhängigkeit in der Darstellung als Hilfstabelle an ChaTEAU übergeben, müssen wir dies im XML-Dokument passend darstellen. Hierzu müssen wir die Hilfstabellen im Schema hinzufügen und die zusätzlichen TGDs angeben. Als erstes fügen wir für jede verwendete Hilfstabelle $H_i(w, c)$ im Schema eine entsprechende Relation hinzu. Im zweiten Schritt übergeben wir die TGDs zum Erstellen der Hilfstabellen und die TGD mit ersetzten Skolemfunktionen. Um uns die Vorgehensweise zu verdeutlichen, wollen wir im Folgenden beispielhaft das Beispiel 4.8 in dem XML-Format formulieren.

Beispiel 4.9. *Im Beispiel 4.8 stellen wir eine TGD mit Skolemfunktionen so dar, dass diese ChaTEAU übergeben werden kann. Die TGD stellen wir durch folgende TGDs dar:*

$$\text{GRADES}(co, id, sem, gr) \wedge \neg\text{HST}(id, c) \rightarrow \exists N : \text{HST}(id, N)$$
$$\text{GRADES}(co, id, sem, gr) \wedge \text{HST}(id, st) \rightarrow \text{STUDENT}(id, st).$$

Da wir die Tabelle HST *zur Darstellung der Skolemfunktion nutzen, müssen wir diese zusätzlich zur* STUDENT- *und* GRADES-*Relation im XML angeben.

```
<relations>
        <relation name="GRADES">
                <attribute name="course_nr" type="int" />
                <attribute name="student_id" type="int" />
                <attribute name="semester" type="string" />
                <attribute name="grade" type="string" />
        </relation>
        <relation name="HST" >
                <attribute name="student_id" type="int" />
                <attribute name="study_course" type="string" />
        </relation>
        <relation name="STUDENT">
                <attribute name="student_id" type="int" />
                <attribute name="study_course" type="string" />
        </relation>
</relations>
```

Nachdem wir die Relationen definiert haben, übergeben wir die TGD zur Erzeugung der Hilfstabelle.

```
<tgd>
        <body>
                <atom name="GRADES">
                        <variable name="course_nr" type="V" index="1" />
                        <variable name="student_id" type="V" index="1" />
                        <variable name="semester" type="V" index="1" />
                        <variable name="grade" type="V" index="1" />
                </atom>
                <atom name="HST" negation="true">
                        <variable name="student_id" type="V" index="1" />
                        <variable name="study_course" type="V" index="1" />
                </atom>
        </body>
        <head>
                <atom name="HST">
                        <variable name="student_id" type="V" index="1" />
                        <variable name="study_course" type="E" index="1" />
                </atom>
        </head>
</tgd>
```

Als zweite Abhängigkeit übergeben wir die TGD, welche die Hilfstabelle nutzt um die Skolemfunktion darzustellen.

```
<tgd>
        <body>
                <atom name="GRADES">
                        <variable name="course_nr" type="V" index="1" />
                        <variable name="student_id" type="V" index="1" />
                        <variable name="semester" type="V" index="1" />
                        <variable name="grade" type="V" index="1" />
                </atom>
                <atom name="HST">
                        <variable name="student_id" type="V" index="1" />
                        <variable name="study_course" type="V" index="1" />
                </atom>
        </body>
```

```
        <head>
                <atom name="STUDENT">
                        <variable name="student_id" type="V" index="1" />
                        <variable name="study_course" type="V" index="1" />
                </atom>
        </head>
</tgd>
```

Im letzten Schritt kann noch eine Instanz angegeben werden. Das vollständige XML-Dokument mit der Instanz aus dem Beispiel 4.8 befindet sich im Anhang A.2

4.2.2. Darstellung als EGD

Da ein Skolemterm $f_{y_i}(x)$ beschreibt, dass ein Wert y_i nur von den Werten x abhängt [BKM+17], scheint es logisch, diese als EGD darzustellen. Die Idee ist, statt der skolemisierten TGD, die einfache TGD auszuführen und im zweiten Schritt durch eine EGD die erzeugten Nullwerte so gleichzusetzen, dass diese dem Ergebnis der skolemisierten TGD entsprechen. Hierzu erstellen wir für jede atomare Aussage $A(x, f(x), z)$, welche eine Skolemfunktion enthält, eine EGD der Form: $A(x, y_1, z_1) \wedge A(x, y_2, z_2) \rightarrow y_1 = y_2$ und ersetzen in der TGD die Skolemfunktion durch die existensquantifizierte Variable y. Dieses Vorgehen formulieren wir in dem Algorithmus 3.

Algorithmus 3 : Pseudocode zur Darstellung einer skolemisierten Abhängigkeit mit EGDs in ChaTEAU

Data : Skolemisierte Abhängigkeiten t
Result : Modifizierte Abhängigkeit t, Menge an EGDs E

```
1 E = ∅
2 forall Skolemfunktion f_i(x) in t do
3     ersetze f_i(x) durch y_i
4     forall Atome A(x, y_i, z) do
5         füge EGD A(x, y_{i,1}, z_1) ∧ A(x, y_{i,2}, z_2) → y_{i,1} = y_{i,2} zu E hinzu
```

Da wir bei der Darstellung von Skolemfunktionen in ChaTEAU zusätzlich zu den Änderungen an der TGD auch EGDs erzeugen, gibt der Algorithmus die veränderte TGD t und eine Menge von EGDs E aus. Wir initialisieren in der ersten Zeile E als leere Menge, bevor wir in der zweiten Zeile anfangen alle in der Abhängigkeit vorkommenden Skolemfunktionen zu durchlaufen. In der dritten Zeile ersetzen wir die Skolemfunktion f_i durch eine neue Variable y_i. In der vierten und fünften Zeile erstellen wir für jedes Atom in dem y_i vorkommt eine EGD, welche die Abhängigkeit der Variable von der Variablen x beschreibt.

Beispiel 4.10. *Damit die beiden Ansätze sich im Beispiel besser vergleichen lassen, wollen wir die gleiche TGD wie in 4.8 darstellen und auf eine Instanz anwenden. Die TGD ist:*

$$\text{GRADES}(co, id, sem, gr) \rightarrow \text{STUDENT}(id, f_{st}(id)).$$

Nachdem wir die Skolemfunktion durch eine neue Variable ersetzt haben, hat die TGD die Form:

$$\text{GRADES}(co, id, sem, gr) \rightarrow \exists ST : \text{STUDENT}(id, ST).$$

Die EGD zur Darstellung der Skolemfunktion hat die Form:

$$\text{STUDENT}(id, st_1) \wedge \text{STUDENT}(id, st_2) \rightarrow st_1 = st_2.$$

Nun wenden wir die Abhängigkeiten auf eine Instanz an. Als Instanz nutzen wir

$$I_0 = \{\text{GRADES}(001, 1, SS16, 1.7), \text{GRADES}(001, 5, SS16, 3.0), \\ \text{GRADES}(002, 1, WS15/16, 3.7)\}.$$

Bei der Instanz existieren drei aktive Trigger für die TGD. Durch Anwendung der TGD erhalten wir die Instanz

$$I_1 = \{\text{GRADES}(001, 1, SS16, 1.7), \text{GRADES}(001, 5, SS16, 3.0), \\ \text{GRADES}(002, 1, WS15/16, 3.7), \text{STUDENT}(1, \eta_1), \\ \text{STUDENT}(5, \eta_2), \text{STUDENT}(1, \eta_3)\}.$$

Auf dieser Instanz finden wir einen aktiven Trigger für die EGD. Durch die Anwendung der EGD erhalten wir die Instanz

$$I_2 = \{\text{GRADES}(001, 1, SS16, 1.7), \text{GRADES}(001, 5, SS16, 3.0), \\ \text{GRADES}(002, 1, WS15/16, 3.7), \text{STUDENT}(1, \eta_1), \\ \text{STUDENT}(1, \eta_1), \text{STUDENT}(5, \eta_2)\}.$$

Diese Instanz ist die gleiche wie im Beispiel 4.8 und entspricht der Anwendung der skolemisierten TGD.

Darstellung in ChaTEAU Im Gegensatz zur Darstellung mit einer Hilfstabelle, müssen wir bei der Darstellung als EGD nur die EGD zusätzlich angeben und die Skolemfunktionen durch existenzquantifizierte Variablen ersetzen. Um die Formulierung zu verdeutlichen, wollen wir das Beispiel 4.10 im XML-Dokument darstellen.

Beispiel 4.11. *Im Beispiel 4.10 haben wir eine Skolemfunktion durch eine EGD dargestellt. Da hier keine weitere Relation benötigt wird, zeigen wir nur, wie die EGD und die TGD im XML-Dokument dargestellt werden. Als erstes wollen wir die TGD darstellen.*

```
<tgd>
        <body>
                <atom name="GRADES">
                        <variable name="course_nr" type="V" index="1" />
                        <variable name="student_id" type="V" index="1" />
                        <variable name="semester" type="V" index="1" />
                        <variable name="grade" type="V" index="1" />
                </atom>
        </body>
        <head>
                <atom name="STUDENT">
                        <variable name="student_id" type="V" index="1" />
                        <variable name="study_course" type="E" index="1" />
                </atom>
        </head>
</tgd>
```

Es ist zu erkennen, dass dadurch, dass keine Hilfstabelle benötigt wird, die TGD deutlich übersichtlicher ist. Als zweites geben wir die EGD an.

```
<egd>
        <body>
                <atom name="STUDENT">
                        <variable name="student_id" type="V" index="1" />
                        <variable name="study_course" type="V" index="1" />
                </atom>
                <atom name="STUDENT">
                        <variable name="student_id" type="V" index="1" />
                        <variable name="study_course" type="V" index="2" />
                </atom>
        </body>
        <head>
                <atom>
                        <variable name="study_course" type="V" index="1" />
                        <variable name="study_course" type="V" index="2" />
                </atom>
        </head>
</egd>
```

Im Anhang A.3 befindet sich das vollständige XML-Dokument mit der Darstellung der Relationen und Instanz.

4.2.3. Darstellung mit Hilfstabelle und EGD

Beide Ansätze bringen Nachteile mit sich (siehe Abschnitt 4.2.4). So benötigt die Darstellung als Hilfstabelle die Negation von Relationen im Rumpf einer TGD und mit der Darstellung als EGD lassen sich Skolemfunktionen nur darstellen, wenn alle Eingabeparameter in der Relation vorhanden sind. Eine Möglichkeit, um diese Nachteile zu vermeiden ist, die beiden Ansätze zu kombinieren. Hierzu lassen wir bei der Erzeugung der Hilfstabellen die Negation weg und formulieren über diese die EGD. Hierdurch können wir alle Skolemfunktionen darstellen, ohne die Negation zu benötigen. Im Algorithmus 4 geben wir den Pseudocode zum Erstellen der Abhängigkeiten an.

Algorithmus 4 : Pseudocode zur Darstellung einer skolemisierten Abhängigkeit mit EGDs und Hilfstabellen in ChaTEAU

Data : Skolemisierte Abhängigkeiten $t = \phi(x) \rightarrow \psi(x, f(w))$
Result : Modifizierte Abhängigkeit t', Menge an Abhängigkeiten A

1 $A = \emptyset$
2 $t' = t$
3 **forall** *Skolemfunktion* $f_i(x)$ *in* t **do**
4 ersetze $f_i(x)$ durch y_i
5 füge $\phi(x) \rightarrow \exists y_i : H_i(w, y_i))$ zu A hinzu
6 füge EGD $H_i(w, y_1) \wedge H_i(w, y_2)) \rightarrow y_{i,1} = y_{i,2}$ zu A hinzu
7 füge $H_i(w, y_i)$ zum rumpf von t' hinzu

Die Darstellung wollen wir uns nun an einem kleinem Beispiel veranschaulichen. Im Beispiel verwenden wir die gleiche Abhängigkeit wie in den Beispielen 4.8 und 4.10 zu den anderen Ansätzen.

Beispiel 4.12. *Für dieses Beispiel nutzen wir wieder die TGD aus dem Beispiel 2.5 und schränken die Relation* STUDENT *auf die Attribute* `id` *und Studiengang* `st` *ein, sodass wir folgende Abhängigkeit erreichen:*

$$\text{GRADES}(co, id, sem, gr) \rightarrow \text{STUDENT}(id, f_{st}(id)).$$

Im ersten Schritt erzeugen wir zur Darstellung der Skolemfunktion f_{st} *die Hilfstabelle* HST*:*

$$\text{GRADES}(co, id, sem, gr) \rightarrow \exists N : \text{HST}(id, N).$$

Nachdem die Hilfstabelle erstellt ist, formulieren wir die EGD:

$$\mathrm{HST}(\mathit{id}, \mathit{st}_1) \wedge \mathrm{HST}(\mathit{id}, \mathit{st}_2) \rightarrow \mathit{st}_1 = \mathit{st}_2.$$

Nachdem die Hilfstabelle erstellt ist, nutzen wir diese, um die Skolemfunktion in der TGD darzustellen:

$$\mathrm{GRADES}(\mathit{co}, \mathit{id}, \mathit{sem}, \mathit{gr}) \wedge \mathrm{HST}(\mathit{id}, \mathit{st}) \rightarrow \mathrm{STUDENT}(\mathit{id}, \mathit{st}).$$

Zum Abschluss des Beispiels betrachten wir, wie das Ergebnis bei einer Anwendung auf die Instanz

$$I_0 = \{\mathrm{GRADES}(\mathit{001}, \mathit{1}, \mathit{SS16}, \mathit{1.7}), \mathrm{GRADES}(\mathit{001}, \mathit{5}, \mathit{SS16}, \mathit{3.0}),\\ \mathrm{GRADES}(\mathit{002}, \mathit{1}, \mathit{WS15/16}, \mathit{3.7})\}$$

aussieht. Im ersten Schritt werden drei aktive Trigger für die erste Abhängigkeit gefunden. Nach der Anwendung der Abhängigkeiten erhalten wir die Instanz

$$I_1 = \{\mathrm{GRADES}(\mathit{001}, \mathit{1}, \mathit{SS16}, \mathit{1.7}), \mathrm{GRADES}(\mathit{001}, \mathit{5}, \mathit{SS16}, \mathit{3.0}), \mathrm{GRADES}(\mathit{002}, \mathit{1}, \mathit{WS15/16}, \mathit{3.7}),\\ \mathrm{HST}(\mathit{1}, \eta_1), \mathrm{HST}(\mathit{1}, \eta_2), \mathrm{HST}(\mathit{5}, \eta_3)\}.$$

Auf dieser Instanz existieren ein aktiver Trigger für die EGD und drei aktive Trigger für die zweite TGD. Wir wählen zufallsbasiert aus und führen als erstes die TGD aus, wodurch die Instanz

$$I_2 = \{\mathrm{GRADES}(\mathit{001}, \mathit{1}, \mathit{SS16}, \mathit{1.7}), \mathrm{GRADES}(\mathit{001}, \mathit{5}, \mathit{SS16}, \mathit{3.0}), \mathrm{GRADES}(\mathit{002}, \mathit{1}, \mathit{WS15/16}, \mathit{3.7}),\\ \mathrm{HST}(\mathit{1}, \eta_1), \mathrm{HST}(\mathit{1}, \eta_2), \mathrm{HST}(\mathit{5}, \eta_3)\\ \mathrm{STUDENT}(\mathit{1}, \eta_1), \mathrm{STUDENT}(\mathit{1}, \eta_2), \mathrm{STUDENT}(\mathit{5}, \eta_3)\}$$

entsteht. Hiernach existiert nur noch ein aktiver Trigger für die EGD. Nach Anwendung der EGD erhalten wir die finale Instanz:

$$I_3 = \{\mathrm{GRADES}(\mathit{001}, \mathit{1}, \mathit{SS16}, \mathit{1.7}), \mathrm{GRADES}(\mathit{001}, \mathit{5}, \mathit{SS16}, \mathit{3.0}), \mathrm{GRADES}(\mathit{002}, \mathit{1}, \mathit{WS15/16}, \mathit{3.7}),\\ \mathrm{HST}(\mathit{1}, \eta_1), \mathrm{HST}(\mathit{5}, \eta_3)\\ \mathrm{STUDENT}(\mathit{1}, \eta_1), \mathrm{STUDENT}(\mathit{5}, \eta_3)\}.$$

Darstellung in ChaTEAU Den Ansatz schließen wir mit einem Blick auf die Darstellung im XML-Eingabeformat von ChaTEAU. Im Schemabereich der Eingabe müssen wir, wie im ersten Ansatz (Abschnitt 4.2.1), die Hilfstabelle zum Schema hinzufügen. Die Abhängigkeiten können wie gewohnt im XML-Format dargestellt werden. Als Beispiel für eine Darstellung im XML-Format, befindet sich im Anhang A.4 die XML-Eingabedatei zum Beispiel 4.12.

4.2.4. Vergleich der Ansätze

Abschließend vergleichen wir unsere drei Ansätze. Der erste Ansatz verwendet nur eine Hilfstabelle zur Darstellung der Skolemfunktion und wurde im Abschnitt 4.2.1 vorgestellt. Der zweite Ansatz verwendet eine EGD über der Zielrelation zur Darstellung und wurde im Abschnitt 4.2.2 vorgestellt. In Abschnitt 4.2.3 wird der dritte Ansatz dargestellt, in welchem zur Umsetzung eine EGD über einer Hilfstabelle vorformuliert wird. Um die Abhängigkeit $\mathrm{GRADES}(\mathtt{co}, \mathtt{id}, \mathtt{sem}, \mathtt{gr}) \rightarrow \mathrm{STUDENT}(\mathtt{id}, f_{st}(\mathtt{id}))$ darzustellen, nutzen die Ansätze folgende Abhängigkeiten:

Ansatz 1: $\text{GRADES}(\texttt{co}, \texttt{id}, \texttt{sem}, \texttt{gr}) \wedge \neg\text{HST}(\texttt{id}, \texttt{c}) \rightarrow \exists \texttt{N} : \text{HST}(\texttt{id}, \texttt{N})$
$\text{GRADES}(\texttt{co}, \texttt{id}, \texttt{sem}, \texttt{gr}) \wedge \text{HST}(\texttt{id}, \texttt{st}) \rightarrow \text{STUDENT}(\texttt{id}, \texttt{st});$

Ansatz 2: $\text{GRADES}(\texttt{co}, \texttt{id}, \texttt{sem}, \texttt{gr}) \rightarrow \exists \texttt{ST} : \text{STUDENT}(\texttt{id}, \texttt{ST})$
$\text{STUDENT}(\texttt{id}, \texttt{st}_1) \wedge \text{STUDENT}(\texttt{id}, \texttt{st}_2) \rightarrow \texttt{st}_1 = \texttt{st}_2;$

Ansatz 3: $\text{GRADES}(\texttt{co}, \texttt{id}, \texttt{sem}, \texttt{gr}) \rightarrow \exists \texttt{N} : \text{HST}(\texttt{id}, \texttt{N})$
$\text{HST}(\texttt{id}, \texttt{st}_1) \wedge \text{HST}(\texttt{id}, \texttt{st}_2) \rightarrow \texttt{st}_1 = \texttt{st}_2$
$\text{GRADES}(\texttt{co}, \texttt{id}, \texttt{sem}, \texttt{gr}) \wedge \text{HST}(\texttt{id}, \texttt{st}) \rightarrow \text{STUDENT}(\texttt{id}, \texttt{st}).$

Für den Vergleich wollen wir auf folgende Aspekte eingehen: Als Erstes betrachten wir den Unterschied in den Ansprüchen an die Chase-Implementation, als Zweites die Ausdruckskraft der Ansätze, als Drittes die benötigten Änderungen am Schema und zum Abschluss die voraussichtlichen Laufzeitunterschiede. Nachdem wir uns diese Aspekte angeschaut haben, wollen wir eine Empfehlung geben, welcher Ansatz sich für eine mögliche Umsetzung in einem Parser eignet.

- Die Ansätze zwei und drei nutzen nur TGDs und EGDs ohne Negation, welche grundlegende Eingaben für den Chase sind. Da der erste Ansatz die Negation benötigt, hat dieser höhere Anforderungen an die Chase-Implementation und kann nicht mit jeder Implementation verwendet werden. Im Fall von ChaTEAU heißt das, dass dieser Ansatz erst ab Version 1.3 funktioniert.

- Wenn wir die Skolemfunktionen als EGD darstellen, müssen die Eingabeparameter zusammen mit der Ausgabe der Skolemfunktion in einer Tabelle vorhanden ein. Wenn der Kopf einer TGD aus mehreren atomaren Aussagen besteht, ist dies für den zweiten Ansatz nicht unbedingt der Fall. Ein praktisches Beispiel hierfür ist eine Abbildung, welche eine Relation aufspaltet. So kann zum Beispiel die TGD

  $$\text{AB}(\texttt{a}, \texttt{b}) \rightarrow \text{A}(f(\texttt{a}, \texttt{b}), a) \wedge \text{B}(f(\texttt{a}, \texttt{b}), b)$$

  nicht mit dieser Variante dargestellt werden, da in den Tabellen A und B jeweils nur einer der beiden Eingabeparameter der Skolemfunktion vorhanden ist und wir eine EGD immer über eine Relation formulieren. Hierdurch können wir nur eine Abhängigkeit von einem der beiden Werte, nicht von beiden beschreiben. Eine Darstellung mit einer Hilfstabelle ist hingegen immer möglich.

- Die Umsetzung der Skolemfunktionen ohne Hilfstabellen hat den großen Vorteil, dass für diese Variante keine Änderungen am Quellschema vorgenommen werden müssen. Dies hat den Vorteil, dass nur auf dem Quellschema gearbeitet wird und nach Anwendung der Abhängigkeiten keine zusätzlichen Relationen im Schema vorhanden sind. Dadurch wird dieses übersichtlicher. Der Vorteil der Hilfstabellen ist, dass diese die gleichen Informationen wie ein Skolemterm enthalten. Sie beschreiben, durch welche Eingabeparameter ein Nullwert entstanden ist. Hilfstabellen können in einem zweiten Schritt durch die Anwendung von ST-TGDs entfernt werden.

- Wir betrachten als erstes die Anzahl der Chase-Schritte, wenn in einer TGD eine Skolemfunktion dargestellt wird. Da der zweite Ansatz als Erstes mehrere Ausgaben für die gleiche Eingabe für eine Skolemfunktion berechnet, braucht dieser auch mehr Chase-Schritte. Kommt ein Eingabewert n mal vor, werden in n Schritten Ausgabewerte erzeugt und diese werden dann mit der EGD in $n-1$ Schritten reduziert, sodass wir auf $2n-1$ Chase-Schritte kommen.
  In Ansatz drei ist die EGD über der Hilfstabelle formuliert. Hierdurch kommt pro Vorkommen der Eingabe ein weiterer Schritt für die Erzeugung der Hilfstabelle hinzu. Somit benötigt der dritte Ansatz bei n gleichen Eingaben $3n-1$ Schritte.
  Nutzen wir nur Hilfstabellen für die Darstellung, wird nur ein Eintrag in der Hilfstabelle erstellt, auch wenn die gleichen Eingabewerte mehrfach vorkommen. Bei der Anwendung der Hilfstabelle als Ersatz für die Skolemfunktion entsteht ein weiterer Schritt. Hierdurch kommen wir unabhängig

von der Häufigkeit des Eingabewertes auf zwei Chase-Schritte.
Wenn Eingabewerte mehrfach vorkommen, ist die Darstellung vom Skolemfunktionen nur mit Hilfstabellen somit die effektivste. Wenn mehrere Skolemfunktionen in einer TGD vorkommen, muss bei den Varianten zwei und drei damit gerechnet werden, dass im schlechtesten Fall alle Kombinationen vor der Reduzierung durch die EGD erstellt werden. In diesen Fällen sollte der erste Ansatz also deutlich effektiver sein.

Da man mit dem zweiten Ansatz nicht alle Skolemfunktionen darstellen kann, können wir diesen Ansatz nicht empfehlen. Wenn das zu Grunde liegende Chase-Tool keine Negation unterstützt, bleibt somit nur noch die dritte Variante übrig. Für Chase-Tools wie ChaTEAU, welche die Negation unterstützen, ist jedoch der erste Ansatz zu empfehlen, da dieser weniger Abhängigkeiten verwendet und nur wenige Chase-Schritte braucht.

5. Fazit

Zusammenfassung: In dieser Arbeit haben wir zwei Themenbereiche behandelt. Im ersten Themenbereich haben wir uns mit SO-TGDs in dem Chase-Tool ChaTEAU beschäftigt. Für die Komposition haben wir festgestellt, dass, sobald die erste Abbildung keine GAV-Abbildung ist, für die Darstellung eine SO-TGD benötigt wird. Außerdem haben wir die Chase-Inverse von Abhängigkeiten betrachtet. Hierbei haben wir festgestellt, dass sich die Chase-Inverse immer als GAV-Abbildung darstellen lässt. Außerdem konnten wir zeigen, wie die Chase-Inverse einer SO-TGD ermittelt werden kann. Hierzu nutzen wir, dass eine SO-TGD als eine Folge von ST-TGDs dargestellt werden kann. Die Inverse bestimmen wir durch eine Komposition der Inversen der ST-TGDs. Den Themenbereich der SO-TGDs schließen wir dadurch ab, dass wir zeigen das SO-TGDs in ChaTEAU dargestellt werden können. Hierzu stellen wir die SO-TGD als eine Folge von ST-TGDs dar.
Im zweiten Themenbereich haben wir Möglichkeiten vorgestellt, wie Skolemfunktionen in ChaTEAU umgesetzt werden können. Wir haben drei mögliche Lösungen vorgestellt, wie Skolemfunktionen umgesetzt werden können. Die erste Lösung nutzt lediglich eine Hilfstabelle, benötigt zur Erzeugung dieser aber die Negation einer Relation in einer TGD. Die zweite Lösung nutzt eine EGD über der erzeugten Relation, um die Skolemfunktion darzustellen. In der dritten Lösung wird eine Hilfstabelle verwendet und über dieser eine EGD formuliert, um eine Hilfstabelle wie in der ersten Lösung zu erstellen, ohne die Negation zu nutzen. Diese haben wir unter verschiedenen Aspekten verglichen und für eine mögliche Umsetzung in ChaTEAU die erste Lösung, also die Darstellung als Hilfstabelle empfohlen.

Ausblick In der vorliegenden Arbeit haben wir gezeigt, wie wir Skolemfunktionen, eine Folge von ST-TGDs und SO-TGDs in ChaTEAU darstellen können. In zukünftigen Arbeiten kann darauf aufbauend ein Parser entwickelt werden. In dem Projekt ProSA existiert bereits ein Parser [Kav22], welchen wir entsprechend erweitern können. Da im ProSA-Projekt mit Anfragen an eine Datenbank gearbeitet wird und Anfragen keine Skolemfunktionen enthalten, ist die Darstellung der Skolemfunktionen für ProSA nicht unbedingt relevant. Interessant wäre hier, ob sich der Ansatz für die Skolemfunktionen auf allgemeinere Funktionen, wie zum Beispiel der Multiplikation von Werten oder der Konkatenation von Zeichenketten übertragen lässt.
Wir haben zudem gezeigt, dass eine SO-TGD als Folge von ST-TGDs dargestellt an ChaTEAU übergeben werden kann. Die Darstellung der Skolemfunktionen als Hilfstabelle könnte hier aber eine weitere und direktere Darstellung von SO-TGDs in ChaTEAU bieten. Direkt beschreibt in diesem Fall, dass die SO-TGD nicht erst in eine äquivalente Folge von ST-TGDs umgewandelt werden muss. Um zu verdeutlichen wie eine direktere Darstellung aussehen könnte, wollen wir kurz die in unserer Arbeit als Beispiel genutzte SO-TGD

$$\begin{aligned}\exists f : (&\mathrm{STUDENT}(\mathtt{id}, \mathtt{ln}, \mathtt{fn}, \mathtt{st}) \rightarrow \mathrm{VERTRETER}(\mathtt{id}, f(\mathtt{id})),\\ &\mathrm{STUDENT}(\mathtt{id}, \mathtt{ln}, \mathtt{fn}, \mathtt{st}) \wedge (\mathtt{id} = f(\mathtt{id})) \rightarrow \mathrm{SELVERT}(\mathtt{id}))\end{aligned}$$

betrachten. Diese scheint sich auch durch die drei Abhängigkeiten

$$\text{STUDENT}(\texttt{id}, \texttt{ln}, \texttt{fn}, \texttt{st}) \land \neg\text{HST}(\texttt{id}, \texttt{c}) \rightarrow \exists \texttt{N} : \text{HST}(\texttt{id}, \texttt{N}),$$
$$\text{STUDENT}(\texttt{id}, \texttt{ln}, \texttt{fn}, \texttt{st}) \land \text{HST}(\texttt{id}, \texttt{ver}) \rightarrow \text{VERTRETER}(\texttt{id}, \texttt{ver}),$$
$$\text{STUDENT}(\texttt{id}, \texttt{ln}, \texttt{fn}, \texttt{st}) \land \text{HST}(\texttt{id}, \texttt{id}) \rightarrow \text{SELVERT}(\texttt{id}))$$

darstellen zu lassen. Für diese Art der Darstellung müsste gezeigt werden, dass ein durch diese Darstellung erzeugtes Ergebnis immer korrekt ist. Außerdem müsste betrachtet werden, ob eine solche Darstellung für alle SO-TGDs oder nur in bestimmten Fällen möglich ist.

Literaturverzeichnis

[AFM10] Arocena, Patricia C. ; Fuxman, Ariel ; Miller, Reneé J.: Composing Local-as-View Mappings: Closure and Applications. In: *Proceedings of the 13th International Conference on Database Theory*. New York, NY, USA : Association for Computing Machinery, 2010 (ICDT '10). – ISBN 9781605589473, 209?218

[AH18] Auge, Tanja ; Heuer, Andreas: Inverses in Research Data Management: Combining Provenance Management, Schema and Data Evolution (Inverse im Forschungsdatenmanagement). In: Klassen, Gerhard (Hrsg.) ; Conrad, Stefan (Hrsg.): *Proceedings of the 30th GI-Workshop Grundlagen von Datenbanken, Wuppertal, Germany, May 22-25, 2018* Bd. 2126, CEUR-WS.org, 2018 (CEUR Workshop Proceedings), 108–113

[AH19] Auge, Tanja ; Heuer, Andreas: ProSA - Using the CHASE for Provenance Management. In: Welzer, Tatjana (Hrsg.) ; Eder, Johann (Hrsg.) ; Podgorelec, Vili (Hrsg.) ; Latific, Aida K. (Hrsg.): *Advances in Databases and Information Systems - 23rd European Conference, ADBIS 2019, Bled, Slovenia, September 8-11, 2019, Proceedings* Bd. 11695, Springer, 2019 (Lecture Notes in Computer Science), 357–372

[ASG+22] Auge, Tanja ; Scharlau, Nic ; Görres, Andreas ; Zimmer, Jakob ; Heuer, Andreas: ChaTEAU: A Universal Toolkit for Applying the Chase. In: *CoRR* abs/2206.01643 (2022). http://dx.doi.org/10.48550/arXiv.2206.01643. – DOI 10.48550/arXiv.2206.01643

[Aug22] Auge, Tanja: *Provenance Management unter Verwendung von Schemaabbildungen mit Annotationen (in Bearbeitung)*. 2022

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

[EFT18] Ebbinghaus, Heinz-Dieter ; Flum, Jörg ; Thomas, Wolfgang: *Einführung in die mathematische Logik*. 6., überarbeitete und erweiterte Auflage. Berlin : Springer Spektrum, 2018 (SpringerLink Bücher). http://dx.doi.org/10.1007/978-3-662-58029-5. http://dx.doi.org/10.1007/978-3-662-58029-5

[FKMP05] Fagin, Ronald ; Kolaitis, Phokion G. ; Miller, Renée J. ; Popa, Lucian: Data exchange: semantics and query answering. In: *Theor. Comput. Sci.* 336 (2005), Nr. 1, 89–124. http://dx.doi.org/10.1016/j.tcs.2004.10.033. – DOI 10.1016/j.tcs.2004.10.033

[FKPT05] Fagin, Ronald ; Kolaitis, Phokion G. ; Popa, Lucian ; Tan, Wang C.: Composing schema mappings: Second-order dependencies to the rescue. In: *ACM Trans. Database Syst.* 30 (2005), Nr. 4, 994–1055. http://dx.doi.org/10.1145/1114244.1114249. – DOI 10.1145/1114244.1114249

[FKPT11a] Fagin, Ronald ; Kolaitis, Phokion G. ; Popa, Lucian ; Tan, Wang-Chiew: Reverse Data Exchange: Coping with Nulls. In: *ACM Trans. Database Syst.* 36 (2011), jun, 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: Bellahsene, Zohra (Hrsg.) ; Bonifati, Angela (Hrsg.) ; Rahm, Erhard (Hrsg.): *Schema Matching and Mapping*. Springer, 2011 (Data-Centric Systems and Applications). – DOI 10.1007/978–3–642–16518–4_7, 191–222

[GH16] Grunert, Hannes ; Heuer, Andreas: Privacy Protection through Query Rewriting in Smart Environments. In: Pitoura, Evaggelia (Hrsg.) ; Maabout, Sofian (Hrsg.) ; Koutrika, Georgia (Hrsg.) ; Marian, Amélie (Hrsg.) ; Tanca, Letizia (Hrsg.) ; Manolescu, Ioana (Hrsg.) ; Stefanidis, Kostas (Hrsg.): *Proceedings of the 19th International Conference on Extending Database Technology, EDBT 2016, Bordeaux, France, March 15-16, 2016, Bordeaux, France, March 15-16, 2016*, OpenProceedings.org, 2016, 708–709

[GMS12] Greco, Sergio ; Molinaro, Cristian ; Spezzano, Francesca: *Incomplete Data and Data Dependencies in Relational Databases*. Morgan & Claypool Publishers, 2012 (Synthesis Lectures on Data Management). http://dx.doi.org/10.2200/S00435ED1V01Y201207DTM029. http://dx.doi.org/10.2200/S00435ED1V01Y201207DTM029

[HY90] Hull, Richard ; Yoshikawa, Masatoshi: ILOG: Declarative Creation and Manipulation of Object Identifiers. In: McLeod, Dennis (Hrsg.) ; Sacks-Davis, Ron (Hrsg.) ; Schek, Hans-Jörg (Hrsg.): *16th International Conference on Very Large Data Bases, August 13-16, 1990, Brisbane, Queensland, Australia, Proceedings*, Morgan Kaufmann, 1990, 455–468

[JSC+15] John, Ajith K. ; Shah, Shetal ; Chakraborty, Supratik ; Trivedi, Ashutosh ; Akshay, S.: Skolem Functions for Factored Formulas. In: Kaivola, Roope (Hrsg.) ; Wahl, Thomas (Hrsg.): *Formal Methods in Computer-Aided Design, FMCAD 2015, Austin, Texas, USA, September 27-30, 2015*, IEEE, 2015, S. 73–80

[Kav22] Kavisanczki, Ivo: *Erweiterung des ProSA-Parsers um Aggregatfunktionen*, Universität Rostock, Bachelorarbeit, 2022

[KK06] Kreuzer, Martin ; Kühling, Stefan: *Logik für Informatiker*. Pearson Studium, 2006 http://www.pearson-studium.de/main/main.asp?page=bookdetails&ProductID=111263. – ISBN 978–3–8273–7215–4

[Kom19] Komara, Ján: Efficient elimination of Skolem functions in first-order logic without equality. In: *CoRR* abs/1909.01697 (2019). http://arxiv.org/abs/1909.01697

[MM15] Meinel, Christoph ; Mundhenk, Martin: *Mathematische Grundlagen der Informatik: Mathematisches Denken und Beweisen Eine Einführung*. Springer Fachmedien Wiesbaden, 2015. – 51–76 S. http://dx.doi.org/10.1007/978-3-658-09886-5_5. http://dx.doi.org/10.1007/978-3-658-09886-5_5

[SRH+20] Strelnikov, Artur ; Röhrs, Chris ; Herrmann, Leon ; Kaseler, Max ; Flach, Rocco: *Provenance-Tools - Technischer Bericht*, Universität Rostock, Technischer Bericht, 2020

[Tre22] Trebbin, Nico: *Nutzung von ChaTEAU für das Answering-Queries-using-Views-Problem (AQuV)*, Universität Rostock, Bachelorarbeit, 2022

[Zim20] Zimmer, Jakob: *Vereinheitlichung des CHASE auf Instanzen und Anfragen am Beispiel ChaTEAU*, Universität Rostock, Bachelorarbeit, 2020

Anfragenverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

A. XML-Dokumente zu den Beispielen

Die vollständigen XML-Dokumente zu den Beispielen.

A.1. XML-Dokument zum Beispiel 4.7

```
<input>
        <schema>
                <relations>
                        <relation name="STUDENT" tag="S">
                                <attribute name="student_id" type="int" />
                                <attribute name="lastname" type="string" />
                                <attribute name="firstname" type="string" />
                                <attribute name="study_course" type="string" />
                        </relation>
                                <relation name="VERTRETER_Z" tag="S">
                                <attribute name="student_id" type="int" />
                                <attribute name="sprecher_id" type="int" />
                        </relation>
                        <relation name="VERTRETER" tag="T">
                                <attribute name="student_id" type="int" />
                                <attribute name="sprecher_id" type="int" />
                        </relation>
                        <relation name="SELVERT" tag="T">
                                <attribute name="student_id" type="int" />
                        </relation>
                </relations>
                <dependencies>
                        <tgd>
                                <body>
                                        <atom name="STUDENT">
                                                <variable name="student_id" type="V"
                                                    index="1" />
                                                <variable name="lastname" type="V" index=
                                                    "1" />
                                                <variable name="firstname" type="V" index
                                                    ="1" />
                                                <variable name="study_course" type="V"
                                                    index="1" />
                                        </atom>
                                </body>
                                <head>
                                        <atom name="VERTRETER_Z">
                                                <variable name="student_id" type="V"
                                                    index="1" />
                                                <variable name="sprecher_id" type="E"
                                                    index="1" />
                                        </atom>
                                </head>
                        </tgd>
                        <sttgd>
                                <body>
```

```
                                        <atom name="VERTRETER_Z">
                                                <variable name="student_id" type="V"
                                                    index="1" />
                                                <variable name="sprecher_id" type="V"
                                                    index="1" />
                                        </atom>
                                </body>
                                <head>
                                        <atom name="Vertreter">
                                                <variable name="student_id" type="V"
                                                    index="1" />
                                                <variable name="sprecher_id" type="V"
                                                    index="1" />
                                        </atom>
                                </head>
                        </sttgd>
                        <sttgd>
                                <body>
                                        <atom name="VERTRETER_Z">
                                                <variable name="student_id" type="V"
                                                    index="1" />
                                                <variable name="student_id" type="V"
                                                    index="1" />
                                        </atom>
                                </body>
                                <head>
                                        <atom name="SELVERT">
                                                <variable name="student_id" type="V"
                                                    index="1" />
                                        </atom>
                                </head>
                        </sttgd>
                </dependencies>
        </schema>
        <instance>
                <atom name="STUDENT">
                        <constant name="student_id" value="1" />
                        <constant name="lastname" value="Moore" />
                        <constant name="firstname" value="Donald" />
                        <constant name="study_course" value="Computer␣Science␣for␣
                            Teaching" />
                </atom>
                <atom name="STUDENT">
                        <constant name="student_id" value="2" />
                        <constant name="lastname" value="Morgan" />
                        <constant name="firstname" value="Sarah" />
                        <constant name="study_course" value="Mathematics" />
                </atom>
        </instance>
</input>
```

A.2. XML-Dokument zum Beispiel 4.9

```
<input>
        <schema>
                <relations>
                        <relation name="GRADES">
                                <attribute name="course_nr" type="int" />
                                <attribute name="student_id" type="int" />
                                <attribute name="semester" type="string" />
                                <attribute name="grade" type="string" />
                        </relation>
                        <relation name="HST" >
                                <attribute name="student_id" type="int" />
                                <attribute name="study_course" type="string" />
                        </relation>
                        <relation name="STUDENT">
                                <attribute name="student_id" type="int" />
                                <attribute name="study_course" type="string" />
                        </relation>
                </relations>
                <dependencies>
                        <tgd>
                                <body>
                                        <atom name="GRADES">
                                                <variable name="course_nr" type="V" index
                                                    ="1" />
                                                <variable name="student_id" type="V"
                                                    index="1" />
                                                <variable name="semester" type="V" index=
                                                    "1" />
                                                <variable name="grade" type="V" index="1"
                                                     />
                                        </atom>
                                        <atom name="HST" negation="true">
                                                <variable name="student_id" type="V"
                                                    index="1" />
                                                <variable name="study_course" type="V"
                                                    index="1" />
                                        </atom>
                                </body>
                                <head>
                                        <atom name="HST">
                                                <variable name="student_id" type="V"
                                                    index="1" />
                                                <variable name="study_course" type="E"
                                                    index="1" />
                                        </atom>
                                </head>
                        </tgd>
                        <tgd>
                                <body>
                                        <atom name="GRADES">
                                                <variable name="course_nr" type="V" index
                                                    ="1" />
                                                <variable name="student_id" type="V"
                                                    index="1" />
                                                <variable name="semester" type="V" index=
                                                    "1" />
                                                <variable name="grade" type="V" index="1"
                                                     />
                                        </atom>
```

```
                                        <atom name="HST">
                                                <variable name="student_id" type="V"
                                                    index="1" />
                                                <variable name="study_course" type="V"
                                                    index="1" />
                                        </atom>
                                </body>
                                <head>
                                <atom name="STUDENT">
                                        <variable name="student_id" type="V" index="1" />
                                        <variable name="study_course" type="V" index="1"
                                            />
                                </atom>
                                </head>
                        </tgd>
                </dependencies>
        </schema>
        <instance>
                <atom name="GRADES">
                        <constant name="course_nr" value="1" />
                        <constant name="student_id" value="1" />
                        <constant name="semester" value="SS16" />
                        <constant name="grade" value="1.7" />
                </atom>
                <atom name="GRADES">
                        <constant name="course_nr" value="1" />
                        <constant name="student_id" value="5" />
                        <constant name="semester" value="SS16" />
                        <constant name="grade" value="3.0" />
                </atom>
                <atom name="GRADES">
                        <constant name="course_nr" value="2" />
                        <constant name="student_id" value="1" />
                        <constant name="semester" value="WS15/16" />
                        <constant name="grade" value="3.7" />
                </atom>
        </instance>
</input>
```

A.3. XML-Dokument zum Beispiel 4.11

```
<input>
        <schema>
                <relations>
                        <relation name="GRADES">
                                <attribute name="course_nr" type="int" />
                                <attribute name="student_id" type="int" />
                                <attribute name="semester" type="string" />
                                <attribute name="grade" type="string" />
                        </relation>
                        <relation name="STUDENT">
                                <attribute name="student_id" type="int" />
                                <attribute name="study_course" type="string" />
                        </relation>
                </relations>
                <dependencies>
                        <tgd>
                                <body>
                                        <atom name="GRADES">
                                                <variable name="course_nr" type="V" index
                                                    ="1" />
                                                <variable name="student_id" type="V"
                                                    index="1" />
                                                <variable name="semester" type="V" index=
                                                    "1" />
                                                <variable name="grade" type="V" index="1"
                                                     />
                                        </atom>
                                </body>
                                <head>
                                        <atom name="STUDENT">
                                                <variable name="student_id" type="V"
                                                    index="1" />
                                                <variable name="study_course" type="E"
                                                    index="1" />
                                        </atom>
                                </head>
                        </tgd>
                        <egd>
                                <body>
                                        <atom name="STUDENT">
                                                <variable name="student_id" type="V"
                                                    index="1" />
                                                <variable name="study_course" type="V"
                                                    index="1" />
                                        </atom>
                                        <atom name="STUDENT">
                                                <variable name="student_id" type="V"
                                                    index="1" />
                                                <variable name="study_course" type="V"
                                                    index="2" />
                                        </atom>
                                </body>
                                <head>
                                        <atom>
                                                <variable name="study_course" type="V"
                                                    index="1" />
                                                <variable name="study_course" type="V"
                                                    index="2" />
                                        </atom>
```

```
                        </head>
                    </egd>
                </dependencies>
        </schema>
        <instance>
                <atom name="GRADES">
                        <constant name="course_nr" value="1" />
                        <constant name="student_id" value="1" />
                        <constant name="semester" value="SS16" />
                        <constant name="grade" value="1.7" />
                </atom>
                <atom name="GRADES">
                        <constant name="course_nr" value="1" />
                        <constant name="student_id" value="5" />
                        <constant name="semester" value="SS16" />
                        <constant name="grade" value="3.0" />
                </atom>
                <atom name="GRADES">
                        <constant name="course_nr" value="2" />
                        <constant name="student_id" value="1" />
                        <constant name="semester" value="WS15/16" />
                        <constant name="grade" value="3.7" />
                </atom>
        </instance>
</input>
```

A.4. XML-Dokument zum Beispiel 4.12

```
<input>
        <schema>
                <relations>
                        <relation name="GRADES">
                                <attribute name="course_nr" type="int" />
                                <attribute name="student_id" type="int" />
                                <attribute name="semester" type="string" />
                                <attribute name="grade" type="string" />
                        </relation>
                        <relation name="HST" >
                                <attribute name="student_id" type="int" />
                                <attribute name="study_course" type="string" />
                        </relation>
                        <relation name="STUDENT">
                                <attribute name="student_id" type="int" />
                                <attribute name="study_course" type="string" />
                        </relation>
                </relations>
                <dependencies>
                        <tgd>
                                <body>
                                        <atom name="GRADES">
                                                <variable name="course_nr" type="V" index
                                                    ="1" />
                                                <variable name="student_id" type="V"
                                                    index="1" />
                                                <variable name="semester" type="V" index=
                                                    "1" />
                                                <variable name="grade" type="V" index="1"
                                                     />
                                        </atom>
                                </body>
                                <head>
                                        <atom name="HST">
                                                <variable name="student_id" type="V"
                                                    index="1" />
                                                <variable name="study_course" type="E"
                                                    index="1" />
                                        </atom>
                                </head>
                        </tgd>
                <egd>
                        <body>
                                <atom name="HST">
                                        <variable name="student_id" type="V" index="1" />
                                        <variable name="study_course" type="V" index="1"
                                            />
                                </atom>
                                <atom name="HST">
                                        <variable name="student_id" type="V" index="1" />
                                        <variable name="study_course" type="V" index="2"
                                            />
                                </atom>
                        </body>
                        <head>
                                <atom>
                                        <variable name="study_course" type="V" index="1"
                                            />
                                        <variable name="study_course" type="V" index="2"
```

```
                                        />
                                </atom>
                        </head>
                </egd>
                <tgd>
                        <body>
                                <atom name="GRADES">
                                        <variable name="course_nr" type="V" index="1" />
                                        <variable name="student_id" type="V" index="1" />
                                        <variable name="semester" type="V" index="1" />
                                        <variable name="grade" type="V" index="1" />
                                </atom>
                                <atom name="HST">
                                        <variable name="student_id" type="V" index="1" />
                                        <variable name="study_course" type="V" index="1"
                                            />
                                </atom>
                        </body>
                        <head>
                                <atom name="STUDENT">
                                        <variable name="student_id" type="V" index="1" />
                                        <variable name="study_course" type="V" index="1"
                                            />
                                </atom>
                        </head>
                </tgd>
                </dependencies>
        </schema>
        <instance>
                <atom name="GRADES">
                        <constant name="course_nr" value="1" />
                        <constant name="student_id" value="1" />
                        <constant name="semester" value="SS16" />
                        <constant name="grade" value="1.7" />
                </atom>
                <atom name="GRADES">
                        <constant name="course_nr" value="1" />
                        <constant name="student_id" value="5" />
                        <constant name="semester" value="SS16" />
                        <constant name="grade" value="3.0" />
                </atom>
                <atom name="GRADES">
                        <constant name="course_nr" value="2" />
                        <constant name="student_id" value="1" />
                        <constant name="semester" value="WS15/16" />
                        <constant name="grade" value="3.7" />
                </atom>
        </instance>
</input>
```

B. Datenträger

Aufgrund der digitalen Abgabe dieser Arbeit existiert kein Datenträger mit der verwendeten Literatur und ChaTEAU Version. Sowohl die verwendete Literatur als auch die ChaTEAU Version stehen online zur Verfügung.
Die verwendete Literatur ist in der StudIP-Veranstaltung Projekt: MA: So-tgds und Skolemisierung für ChaTEAU[1] hochgeladen. Die verwendete ChaTEAU-Version ist 1.3 und ist im ChaTEAU-GitLab-Projekt im Branch chateau-1.3[2] verfügbar.

[1] Link: `https://studip.uni-rostock.de/dispatch.php/course/details?sem_id=ab595c3a5128901694932705408c05a3`
[2] Link `https://git.informatik.uni-rostock.de/ta093/chateau/-/tree/chateau-1.3`

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 übernommen 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 30.8.2022