Technical Report CS-01-21
Ausblick auf einen erweiterten CHASE-Algorithmus

Andreas Görres und Andreas Heuer

Universität Rostock
Fakultät für Informatik und Elektrotechnik
Institut für Informatik
Lehrstuhl für Datenbank- und Informationssysteme

Long version of a 6-page paper for
32nd GI-Workshop on Foundations of Databases 2021,

erhältlich unter:
www.ls-dbis.de/digbib/dbis-tr-cs-01-21.pdf

Ausblick auf einen erweiterten CHASE-Algorithmus

Andreas Görres
Lehrstuhl für Datenbank- und Informationssysteme
Institut für Informatik
Universität Rostock
andreas.goerres@uni-rostock.de

Andreas Heuer
Lehrstuhl für Datenbank- und Informationssysteme
Institut für Informatik
Universität Rostock
andreas.heuer@uni-rostock.de

KURZFASSUNG

Der CHASE ist ein grundlegender Algorithmus der Datenbanktheorie, der über noch ungenutztes Potential hinsichtlich seiner praktischen Umsetzung in Systemen verfügt. In den von uns untersuchten Anwendungsbereichen wurden mehrere Problemfälle identifiziert, für die der CHASE interessant sein könnte. Allerdings sind diese Anwendungsbereiche mit zahlreichen Datenbankkonzepten wie komplexeren Anfrageoperationen verbunden, die bisher nicht (gemeinsam) in den CHASE integriert wurden.

Die Erweiterung des CHASE um diese Anfrageoperationen, wie Negation, allgemeine Vergleiche oder Aggregatfunktionen, muss jedoch kontrolliert erfolgen, um Effektivität, Konfluenz und Terminierung des CHASE nicht zu gefährden und so eine effektive Anwendung des Algorithmus auf praktisch relevante Problemfälle zu ermöglichen. Um Auswirkungen von CHASE-Erweiterungen einschätzen zu können, müssen zunächst Effizienz, Konfluenz, Terminierung und Effektivität des bisher etablierten CHASE verstanden und gegebenenfalls näher untersucht werden. In dieser Arbeit stellen wir erste Erkenntnisse hinsichtlich Terminierung und Konfluenz des Standard-CHASE zusammen.

Stichworte

CHASE-Algorithmus, Konfluenz, Terminierung, Privacy, Provenance

1. MOTIVATION

Die Auswertung großer Datenmengen, wie sie beispielsweise im Rahmen des *Internet of Things* generiert werden, stellt eine besondere Herausforderung für Datenbanksysteme dar. Am Lehrstuhl für Datenbank- und Informationssysteme der Universität Rostock wird diese Problematik hauptsächlich anhand der strukturierten Daten zweier Anwendungsfälle untersucht: Forschungsdatenmanagement [3] und intelligente Assistenzsysteme [16].

32^{nd} GI-Workshop on Foundations of Databases (Grundlagen von Datenbanken), 25.05.2021 - 28.05.2021, Grimma, Germany.

Anwendungsbereich Forschungsdatenmanagement

Unser Ansatzpunkt für das Auswerten und Managen von Forschungsdaten wissenschaftlicher Institute ist die langjährige Zusammenarbeit des Lehrstuhls mit dem Institut für Ostseeforschung (IOW). Einerseits liefert diese Kooperation Einsichten darüber, welche (z.B. statistische) Operationen praktisch relevant für ein Datenbankmanagementsystem sind, andererseits lässt sich anhand historischer Daten die Schemaevolution von Forschungsdatenbanken untersuchen. Veröffentlichte Auswertungen sollen sich trotz struktureller Veränderungen auf die beteiligten Originaldaten zurückführen lassen, wodurch die Reproduzierbarkeit der Ergebnisse gewährleistet ist. Dabei sind aber auch ein gewisser Schutz von Rechten an den Forschungsdaten (intellectual property preservation) sicherzustellen: Die für die Reproduzierbarkeit nötigen Originaldaten sollen etwa zur Begutachtung veröffentlicht werden, ohne den gesamten Datenbestand an Forschungsdaten veröffentlichen zu müssen. Damit müssen die Ziele Reproduzierbarkeit (Provenance) und Datensparsamkeit in Übereinstimmung gebracht werden.

Anwendungsbereich Assistenzsysteme

Smarte Assistenzsysteme sollen ihren Nutzern gewisse Assistenzfunktionalitäten bieten. Um dieses Ziel erreichen zu können, müssen die Nutzer mit Hilfe diverser Sensoren beobachtet werden, um ihre derzeitigen Aktivitäten zu ermitteln und ihre Intention (was wollen sie in Kürze tun?) ableiten zu können. Im Zusammenhang mit der Aktivitäts- und Intentionserkennung smarter Assistenzsysteme werden insbesondere die Aspekte Privacy und Effizienz in unseren Projekten näher beleuchtet. Wenn durchzuführende Berechnungen bereits bei der Datenerfassung erfolgen, wird unter Umständen eine ineffizient zentrale Zusammenführung und Speicherung der Daten, welche das Prinzip der Datensparsamkeit verletzt, nicht mehr benötigt. Wird die Datenerfassung dann von einer Vielzahl parallel arbeitender Sensoren / Prozessoren vorgenommen, so kann durch die hochgradige Parallelisierung der Datenanalyse auch die Effizienz des Gesamtsystems gesteigert werden. Eine Forschungsfrage ist in diesem Zusammenhang, wie man in einer Form der semantischen Anfragetransformation bzw. Anfrageoptimierung die zentralisierte Datenanalyse auf integrierten Datenbeständen in eine parallelisierte Analyse nahe an den Datenquellen automatisch transformieren kann. Dies führt dann nicht nur zu einer effizienteren Anfragebearbeitung, sondern auch durch eine datensparsame Realisierung der gewünschten Auswertung (Privacy-by-Design).

Realisierung von Rahmenbedingungen

Für die genannten Data-Science-Anwendungen lassen sich also die drei Rahmenbedingungen Privacy, Provenance und Optimierung definieren:

- Privacy: Wie kann man automatisiert eine zentrale Datenauswertung in eine vollständig dezentrale Datenauswertung transformieren, um Datensparsamkeit und Effizienz zu erreichen?
- Provenance: Wie kann man automatisch eine abstrahierte, anonymisierte Sicht auf die minimal erforderlichen Originaldaten generieren?
- Optimierung der Datenauswertung: Wie kann man aufwändige Basisoperationen in den Auswertungsverfahren identifizieren und dann insofern minimieren, dass die komplexe Auswertung automatisch (bei garantiert identischem Auswertungsergebnis) in einen weit effizienteren, möglichst hochgradig parallelisierten, Algorithmus transformiert wird?

Es wäre wünschenswert, wenn Privacy und Provenance in das Datenauswertungsverfahren integriert und dann automatisch optimiert werden könnten.

Grundsätzlich stellen Privacy und Provenance jedoch gegensätzliche Anforderungen an das Auswertungsverfahren und sind daher nicht leicht kombinierbar.

Sollen die Rahmenbedingungen unabhängig voneinander erfüllt werden, existieren bereits Speziallösungen. Prinzipiell können die Teilprobleme aber auch mit dem CHASE, einem universellen Algorithmus der Datenbanktheorie, bearbeitet werden. In dieser Arbeit sollen die entsprechenden Teilprobleme für den CHASE formalisiert werden, um sie dann gemeinsam mit einem in sich geschlossenen Algorithmus zu lösen.

Die Grundidee des CHASE

Der CHASE integriert CHASE-Parameter, dargestellt als spezielle prädikatenlogische Formeln, in CHASE-Objekte, diese auch dargestellt als spezielle, aber andersartige, prädikatenlogische Formeln. So kann man Integritätsbedingungen als CHASE-Parameter in Anfragen integrieren, um die Anfragen semantisch (unter Berücksichtigung der Integritätsbedingungen) optimieren zu können. Auf diese Weise kann beispielsweise die Anzahl der Verbundoperationen reduziert werden, was die Effizienz der Anfrage steigert. Man kann als CHASE-Parameter aber auch Sichtdefinitionen verwenden, um diese in Anfragen integrieren zu können, um die Anfragen statt über den Relationen der Datenbank nur noch über den Sichten auszuführen. Auf diese Weise kann beispielsweise Privacy garantiert werden, wenn der direkte Zugriff auf die Basisrelationen der Datenbank aus Datenschutzgründen nicht erlaubt ist. Statt Anfragen als CHASE-Objekte zu verwenden, kann man aber auch eine Datenbank als CHASE-Objekt verstehen. Die CHASE-Parameter sind dann Transformationen der Datenbank, etwa auch Datenanalysen oder klassische Anfragen. Der CHASE kann dann als Anfragemechanismus verstanden werden. Anwendungsfälle für den CHASE auf Datenbanken ist etwa die Integration mehrerer lokalen Datenbanken zu einer zentralen Datenbank oder die gemeinsame Evolution eines Datenbankschemas und den jeweiligen Datenbankanfragen unter Berücksichtigung der Provenance. In Unterabschnitt 3.1 werden wir, nach Besprechung der theoretische Grundlagen des CHASE und ihrer Demonstration anhand eines praktischen Beispiels, auf die unterschiedlichen Anwendungszwecke des CHASE zurückkommen.

Um die oben genannten Teilprobleme der Privacy, Provenance und Anfrageoptimierung gemeinsam lösen zu können, müssen die bisher getrennten Anwendungen des CHASE auf verschiedene CHASE-Objekte und mit verschiedenen CHASE-Parametern in eine gemeinsame Technik integriert werden.

2. PROBLEMSTELLUNG

Die Rahmenbedingungen Privacy, Provenance und Effizenzoptimierung sollen in Datenauswertungsverfahren integriert werden, indem über die einheitliche Grundtechnik des CHASE ursprünglich getrennte Problemlösungen zu einer Gesamtlösung kombiniert werden.

Hieraus leiten sich folgende zwei Herausforderungen ab:

- Vereinheitlichung der CHASE-Techniken
- Kontrollierte Erweiterung des CHASE

Vereinheitlichung der CHASE-Techniken: Obwohl der CHASE – von einem abstrakten Gesichtspunkt betrachtet – ein universeller Algorithmus ist, der unterschiedliche Arten von Parametern verarbeiten und so in einer Vielzahl von Aufgabengebieten Verwendung finden könnte [4], gehen praktische Implementierungen des Algorithmus bisher in der Regel von einzelnen Anwendungsfällen aus (z.B. Datenbereinigung oder Anfrageoptimierung). Voraussetzung für die folgenden Arbeitsschritte ist, die unterschiedlichen konkreten Ausführungen des CHASE in einem Gesamtmodell des Algorithmus zu kombinieren, damit die oben genannten Fragestellungen der Privacy, Provenance und Anfrageoptimierung mit derselben Grundtechnik einheitlich und in abgestimmter Kombination zueinander gelöst werden können.

Erweiterung des CHASE: Obwohl der CHASE in der Datenbanktheorie ein ungeheuer mächtiges Werkzeug darstellt, ist er bisher für praktische Erfordernisse von Datenbankanwendungen inadäquat. Tatsächlich beschränkt sich der klassische Standard-CHASE auf die Abbildung von Konjunktionen positiver Selektions-Projektions-Verbund-Anfragen, wobei die Selektion lediglich auf dem Testen von Gleichheit beruht. Um Effektivität in den zuvor genannten Anwendungsgebieten zu erreichen, wird eine Erweiterung um weitere Vergleichsoperatoren, Negation und Funktionen benötigt. Diese Erweiterung sollte jedoch kontrolliert erfolgen, um Terminierung, Konfluenz und Effizienz des Verfahrens nicht zu gefährden. Tatsächlich handelt es sich beim in dieser Arbeit betrachteten CHASE-Algorithmus bereits um eine Erweiterung des ursprünglichen Tableau-Verfahrens – auch diese Erweiterungen gingen mit Einschränkungen der ursprünglich garantierten Terminierung und Konfluenz einher.

3. STAND DER FORSCHUNG

3.1 Der CHASE

Zahlreiche grundlegende Anwendungsfälle der Datenbankforschung haben gemeinsam, dass sie mit Hilfe eines Universalmodells gelöst werden können [9]. Soll beispielsweise eine Quelldatenbank unter Berücksichtigung von Integritätsbedingungen in eine Zieldatenbank überführt werden, so kann

es zahlreiche mögliche Lösungen des Problems geben, welche die Integritätsbedingungen erfüllen. Aus einigen dieser Zieldatenbanken – den sogenannten universellen Lösungen – lassen sich alle anderen Lösungen des Problems durch homomorphe Abbildungen gewinnen. Stellt man (boolesche) Datenbankanfragen an diese Universallösung, so erhält man genau für die Anfragen positive Antworten (die sogenannten sicheren Antworten), die auch für jede andere mögliche Variante der Zieldatenbank eine positive Antwort geliefert hätten. Es liegt also nahe, eine Universallösung als Lösung des Datenaustauschproblems zu wählen und zu materialisieren. Universallösungen lassen sich direkt aus dem Universalmodell ableiten, welche durch den CHASE berechnet werden können. Allerdings handelt es sich bei der Standard-Variante des CHASE (im Gegensatz zu einer komplizierteren Spezialform, dem Core-CHASE) um keinen vollständigen Algorithmus, um Universalmodelle zu erzeugen. In der weiteren Arbeit werden wir jedoch über diese Unvollkommenheit des Standard-CHASE (also die Existenz von Universalmodellen, die dieser CHASE nicht finden kann) hinwegsehen.

Der CHASE-Algorithmus

Im Folgenden soll der grundlegende Ablauf eines CHASE-Schrittes betrachtet werden. Der grundlegende Ablauf ist in Algorithmus 1 (mit CHASE-Parametern Σ, CHASE-Objekt I und neu erzeugten Nullwerten bzw. existenzquantifizierten Variablen FRESH) gegeben. Zunächst müssen wir einige Begrifflichkeiten klären. Der CHASE arbeitet, allgemein gesehen, Parameter in Objekte ein. Sowohl Parameter als auch Objekte sollten als prädikatenlogische Formeln erster Ordnung (ohne Funktionssymbole oder Negationen) darstellbar sein. Ohne Verlust der Allgemeinheit werden wir die Parameter als Integritätsbedingungen und das Objekt als Datenbankinstanz bezeichnen – tatsächlich könnte die Instanz jedoch z.B. auch den Körper einer Datenbankanfrage darstellen (die sogenannte „kanonische“ Instanz der Anfrage). Integritätsbedingungen lassen sich in tupelerzeugende Abhängigkeiten (TGDs) und gleichheiterzeugende Abhängigkeiten (EGDs) unterteilen:

$$TGD : \forall X, Y : \phi(X, Y) \rightarrow \exists Z : \psi(X, Z)$$
$$EGD : \forall X : \phi(X) \rightarrow x_1 = x_2.$$

ϕ und ψ sind hierbei Mengen relationaler Atome, X, Y und Z sind jeweils Mengen allquantifizierter Variablen und Konstanten, Z ist eine Menge existenzquantifizierter Variablen, und x_1 sowie x_2 sind Variablen oder Konstanten aus X. Kommen keine existenzquantifizierten Variablen im Kopf der TGD vor, so sprechen wir von einer vollen Abhängigkeit, ansonsten von einer eingebetteten Abhängigkeit.

Durch den Rumpf der Integritätsbedingung wird ein Muster definiert. Erfüllt ein Tupel der Instanz (oder ein Atom eines Anfragerumpfes) das Muster, lässt sich ein Homomorphismus von den Variablen der Bedingung zu den Konstanten und Nullwerten der Instanz (bzw. zu den all- und existenzquantifizierten Variablen des Anfragerumpfes) finden. In diesem Fall sprechen wir auch vom Vorliegen eines *Triggers*. Der Trigger ist aktiv (Zeile 3), wenn die Bedingung noch nicht erfüllt ist. Für TGDs testen wir hierfür, ob der Homomorphismus so für die existenzquantifizierten Variablen des TGD-Kopfes erweitert werden kann, dass das Bild des TGD-Kopfes unter dem erweiterten Homomorphismus bereits in der Datenbank existiert. Ist dies nicht der Fall, wird der Homomorphismus stattdessen so erweitert, dass die existenzquantifizierten Variablen auf neu erzeugte markierte Nullwerte abgebildet sind (Zeile 5). Die so definierten neuen Tupel werden in der Datenbankinstanz materialisiert (Zeile 6). Für EGDs wird überprüft, ob das Gleichheitsatom $x_1 = x_2$ bereits erfüllt ist, also ob das Bild beider Terme unter dem Homomorphismus identisch ist. Ist dem nicht der Fall, unterscheiden wir die folgenden drei Fälle:

Algorithmus 1 STANDARD-CHASE(Σ, I)

```
1: while fixpoint not reached do
2:   choose nondeterministically τ ∈ Σ
3:   for each  active trigger h for τ in I do
4:     if τ is a TGD with head ∃Z : ψ(X, Z) then
5:       h := h ∪ {Z ↦ N} where N ⊆ FRESH
6:       I := I ∪ h(ψ(X, Z))
7:     else if τ is an EGD with head x_i = x_j then
8:       if {h(x_i) ≠ h(x_j)} ⊆ CONSTANT then
9:         fail
10:      ω := {h(x_i) → h(x_j)} (h(x_j) > h(x_i))
11:      I := ω(I)
```

- Das Bild beider Terme sind unterschiedliche Konstanten (wobei das Bild einer Konstanten die Konstante selbst ist). Ist dies der Fall, scheitert der CHASE (Zeile 9).
- Das Bild beider Terme sind unterschiedliche Nullwerte. In der gesamten (Ziel-) Datenbankinstanz wird einer der Nullwerte (gewöhnlich der mit höherem Index) durch den anderen ersetzt.
- Das Bild des einen Terms ist eine Konstante, während das Bild des anderen Terms ein Nullwert ist. In der gesamten (Ziel-) Datenbankinstanz wird der Nullwert durch die Konstante ersetzt.

Für Instanzen lässt sich also eine Hierarchie zwischen Konstanten und Nullwerten definieren (siehe Größenvergleich in Zeile 10 des Algorithmus). Handelt es sich beim CHASE-Objekt um eine Datenbankanfrage, besteht diese Hierarchie zwischen Konstanten, allquantifizierten und existenzquantifizierten Variablen. Wird die Anfrage in Form eines Tableaus dargestellt, treten an die Stelle der allquantifizierten und existenzquantifizierten Variablen ausgezeichnete und nichtausgezeichnete Variablen, die im Folgenden durch die Buchstaben a und b gekennzeichnet werden.

Beispiel: CHASE im Datenbankentwurf

Exemplarisch soll die Wirkungsweise des CHASE anhand eines in Tableauform codierten Datenbankschemas dargestellt werden (Tabelle 1). Jede Zeile des Tableaus steht für eine Tabelle, wobei die ausgezeichneten Variablen (die a_i-Variablen) die in der jeweiligen Relation vorkommenden Attribute kennzeichnen. Die Datenbank besteht also aus zwei Relationen (im Folgenden Bachelor und Master genannt), welche den Matrikelnummern von Studierenden die jeweilige Bachelor- (Nb) oder Master-Abschlussnote (Nm) zuordnen. Die in der jeweiligen Relation nicht vorkommenden Attribute werden durch nichtausgezeichnete Variablen (die b_j-Variablen) repräsentiert.

Berücksichtigen wir, dass funktionale Abhängigkeiten zwischen Matrikelnummer und Note_Bachelor sowie zwischen

Matrikelnummer	Note_Bachelor	Note_Master
a_1	a_2	b_1
a_1	b_2	a_3

Tabelle 1: Tableau eines Datenbankschemas

Matrikelnummer und Note_Master bestehen, können wir folgende EGD [1] in das Tableau einarbeiten:

$$\text{Abschlussnoten}(ma, nb_1, nm_1),$$
$$\text{Abschlussnoten}(ma, nb_2, nm_2) \rightarrow nb_1 = nb_2, nm_1 = nm_2.$$

Bei der hier genannten Abschlussnoten-Relation handelt es sich um die (fiktive) universelle Relation, aus der die Relationen Bachelor und Master durch Projektion entstanden sind. Bei Anwendung des CHASE ersetzen wir die nichtausgezeichneten Variablen b_1 und b_2 durch die ausgezeichneten Variablen a_1 und a_2, wodurch die beiden Zeilen des Tableaus (Tabelle 2) verschmelzen. Da das entstehende Tableau aus einer einzigen Zeile besteht, die nur ausgezeichnete Variablen enthält, ist das ursprüngliche Schema hinsichtlich der gegebenen EGDs verbundtreu (d.h. es ist unter den gegebenen Abhängigkeiten äquivalent zum Schema der universellen Abschlussnoten-Relation).

Beispiel: CHASE in der semantischen Anfrageoptimierung

Wir hätten das Schema auch als eine Datenbankanfrage an die universelle Abschlussnoten-Relation auffassen können, welche durch zwei Projektionen auf die Bachelor- bzw. Masterabschlussnoten jedes Studenten einer Zerlegung in zwei Relationen vornimmt und durch den anschließenden Verbund der zerlegten Relationen eine Relation über allen Attributen ermittelt. Im Allgemeinen liefert diese Anfrage eine Obermenge der Original-Abschlussnoten-Relation. Somit kann der Anfrageoptimierer eines relationalen Datenbanksystems diese Anfrage nicht weiter – etwa durch Entfernen überflüssiger Operationen – optimieren. Gelten aber die beiden EGDs wie oben, so entfallen durch die Reduktion dieses Anfrage-Tableaus auf eine Zeile sowohl die beiden Projektionen als auch der anschließende Verbund. Die Anfrage kann durch eine simple Ausgabe der Abschlussnoten-Relation realisiert werden. Diese Art der Anfrageoptimierung, die das Vorhandensein von Integritätsbedingungen zur Eliminierung überflüssiger Operationen ausnutzt, wird semantische Anfrageoptimierung genannt. Der CHASE ist in diesem Fall das Werkzeug, die Integritätsbedingungen (hier: die EGDs) in die Anfrage (hier: das zugehörige Tableau) einzuarbeiten.

Für die Definition eines Tableaus notwendige Einschränkungen (z.B. das Verbot, Variablen in verschiedenen Attributen zu wiederholen) entfallen bei der Optimierung allgemeiner Anfragen durch den CHASE.

Matrikelnummer	Note_Bachelor)	Note_Master
a_1	a_2	a_3

Tabelle 2: Da eine Reduktion des Schemas aus Tabelle 1 auf eine Zeile mit ausschließlich ausgezeichneten Variablen möglich war, ist das Schema verbundtreu.

[1] Abschnitt 5 geht näher auf die verwendete Notation ein.

Weitere CHASE-Anwendungen

Der CHASE-Algorithmus ist über vierzig Jahre alt [17]. Ursprünglich diente er lediglich der Absicherung eines guten Datenbankentwurfs, seitdem sind jedoch zahlreiche Anwendungsgebiete hinzugekommen, wie beispielsweise Integration heterogener Datenbanken [11], Anfragetransformation mit Beschränkung auf bestimmte Nutzersichten (AQuV; Answering Queries using Views) [10], Data Cleaning [1], Anfrageoptimierung unter Integritätsbedingungen [2] und Datenbanktransformationen [11].

Weitere Anwendungsgebiete, wie etwa die Invertierung von Datenbanktransformationen und Provenance-Management, sind gegenwärtiger Stand der Forschung am Lehrstuhl für Datenbank- und Informationssysteme [3].

Aus den oben genannten Anwendungsgebieten ergibt sich bereits eine deutliche Variabilität der möglichen CHASE-Parameter und -Objekte. Für Datenbankintegration und Datenbanktransformation werden Transformationsregeln (source-to-target TGDs) verwendet, für AQuV kommen Sichtdefinitionen (TGDs) zum Einsatz, und Data Cleaning sowie Anfrageoptimierung erfolgt unter Verwendung allgemeiner Integritätsbedingungen (EGDs und TGDs). Wie bereits zuvor erwähnt, handelt es sich beim Objekt des CHASE entweder um eine Datenbankanfrage (Anfrageoptimierung und AQuV) oder um eine Datenbankinstanz (alle anderen Anwendungsfälle).

Während die meisten der oben genannten Probleme bei Wahl geeigneter CHASE-Parameter und -Objekte durch den Standard-CHASE direkt gelöst werden können, setzt die Anfrageoptimierung und -transformation unter Integritätsbedingungen durch den CHASE mehrere Hilfsalgorithmen voraus. Im Wesentlichen handelt es sich hierbei um das unter dem Namen Backchase bekannte Verfahren, welches das Ergebnis des vorherigen CHASE als Ausgangspunkt eines erneuten CHASE in umgekehrter Richtung verwendet und anschließend durch Finden eines homomorphen Kerns Redundanzen aus dem Ergebnis entfernt.

3.2 Konfluenz, Terminierung und Effizienz des universellen CHASE

Bisher wird der CHASE-Algorithmus zwar noch nicht in kommerziellen Lösungen verwendet (Ausnahme: aus dem Forschungsprototyp Clio entwickelte Teile des IBM Information Servers), es gibt jedoch eine Reihe prototypischer Implementierungen, mit denen einige Anwendungsfälle des CHASE näher untersucht werden können. Allerdings kann bisher keiner dieser Prototypen alle in Unterabschnitt 3.1 genannten Anwendungsbereiche bearbeiten. So sind in der CHASE-Software Llunatic [13] effiziente Techniken des Datenbankbereinigung implementiert. Dies betrifft nicht nur das Ersetzen von Nullwerten durch Konstanten, sondern auch das Auflösen von Konflikten unterschiedlicher Konstanten, indem das System auf nutzerdefinierte Präferenzregeln, Mastertabellen oder Nutzereingaben zurückgreift. Im Gegensatz hierzu kann PDQ [6] Anfrageoptimierungen durchführen, die selbst kommerziellen Systemen überlegen sind. PDQ berücksichtigt neben Integritätsbedingungen auch Zugriffsmuster und Kostenfunktionen. Aus diesen erzeugt das Programm nicht nur optimierte Anfragen, sondern auch die hierzu passenden Anfragepläne. Die von Llunatic und PDQ berücksichtigten CHASE-Objekte sind also unterschiedlich und nur in der Theorie äquivalent.

Die Verarbeitung der unterschiedlichen CHASE-Parame-

ter und -Objekte ist tatsächlich in keinem uns bekannten Werkzeug vereinheitlicht, stattdessen kommen immer getrennte Techniken zum Einsatz. Die CHASE-Software Graal [5] zeigt Ansätze für die analoge Behandlung von Instanzen und Anfragen, jedoch sind auch hier die berücksichtigten Anwendungsfälle sehr speziell (eine zu PDQ äquivalente Anfrageoptimierung ist beispielsweise nicht möglich). Zum einen ermöglicht es Graal, Anfragen an eine (möglicherweise unvollständige) Datenbankinstanz zu stellen, welche durch Anwendung von TGDs ergänzt wurde. Zum anderen formuliert Graal Anfragen unter Verwendung von TGDs um und wendet diese auf Datenbankinstanzen an.

An dieser Stelle sei erwähnt, dass einige CHASE-Prototypen bereits jetzt über Erweiterungen des zuvor beschriebenen Standard-CHASE verfügen. VLog integriert beispielsweise Negationen in den CHASE [7]. Ursprünglich handelte es sich bei VLog um eine prototypische Umsetzung der logischen Programmiersprache Datalog, welche vor allem im Datenbankumfeld eingesetzt wird. Datalog ermöglicht die Definition voller TGDs (in VLog auf eingebettete TGDs erweitert), die keine Funktionssymbole, jedoch unter Umständen Negationen enthalten dürfen. In VLog sind diese Negationen auf stratifizierte Mengen von TGDs [2] beschränkt, eine Einschränkung, die auf die Datalog-Theorie zurückgeht. Wenn wir den Test der Triggeraktivität als Negation im CHASE interpretieren (siehe Abschnitt 5), verfügt jedoch selbst der klassische Standard-CHASE über Negationen – und zwar ohne eine Beschränkung auf stratifizierte TGDs.

Die ursprüngliche Variante des CHASE – d.h. die Berücksichtigung von Verbund-Abhängigkeiten (JDs) und Funktionalen Abhängigkeiten (FDs) beim Datenbankentwurf – ist sowohl terminierend, als auch konfluent. Der CHASE erzeugt in diesem Fall also unabhängig von der Reihenfolge der Regelanwendungen stets das gleiche (oder zumindest ein isomorphes) Ergebnis. Die Erweiterung des CHASE um existenzquantifizierte Variablen erlaubt es zwar, z.B. Inklusionsabhängigkeiten und damit auch Fremdschlüsselbeziehungen zu berücksichtigen, jedoch unter Verlust der sicheren Terminierung des Algorithmus. Tatsächlich kann diese Terminierung des CHASE sogar abhängig von der Reihenfolge der Regelanwendungen sein. Obwohl das allgemeine Problem der CHASE-Terminierung unentscheidbar ist, wurden inzwischen eine ganze Reihen von Testverfahren entwickelt, welche eine sichere Terminierung des CHASE allein auf Basis der Integritätsbedingungen zusichern können. Einige dieser Kriterien werden in Abschnitt 4 vorgestellt. Häufig beziehen sich CHASE-Terminierungskriterien jedoch auf den Skolem-Oblivious-CHASE, nicht den Standard-CHASE (dessen Terminierung jedoch durch die Terminierung des Skolem-Oblivious-CHASE garantiert wird). Das Kriterium der *WA-Stratifizierung* [15] erkennt zwar, ob der Standard-CHASE eine TGD aus Gründen der Triggerdeaktivierung nicht auf die von einer anderen TGD erzeugten Tupel anwenden kann, allerdings werden bei derartigen Betrachtungen lediglich diese beiden Integritätsbedingungen berücksichtigt. Die Triggerdeaktivierung könnte jedoch durch eine dritte Integritätsbedingung erfolgt sein. Das Kriterium der *Restricted Model Faithful Acyclicity* [8] berücksichtigt ein derartiges Zusammenspiel mehrerer Regeln, geht jedoch von einer bestimmten Strategie der Regelanwendungsreihenfolge aus. Die Terminierung des CHASE – und sogar allgemein sein Ergebnis – sind von dieser Reihenfolge abhängig.

Während die Terminierung des CHASE also recht gut verstanden ist, wurde die erwähnte Reihenfolgeabhängigkeit, also die Konfluenz des CHASE, bisher kaum untersucht. Wie zuvor erwähnt ist der CHASE auf Funktionalen Abhängigkeiten konfluent. Für Erweiterungen der Funktionalen Abhängigkeit ist der CHASE entweder konfluent (Graph-CHASE in [12]) oder Fälle von Inkonfluenz treten auf, die dem Standard-CHASE völlig fremd sind (Conditional FDs in [18]). Interessanterweise sind wichtige Spezialformen des CHASE, wie der Oblivious-CHASE, der Skolem-Oblivious-CHASE und der Core-CHASE, konfluent, wenn wir lediglich eingebettete TGDs, aber keine EGDs, berücksichtigen.

Obwohl zum CHASE bereits zahlreiche Komplexitätsuntersuchungen existieren, beziehen sich diese bisher meist auf Worst-Case-Abschätzungen. In dieser Arbeit soll dies durch neuere Techniken erweitert werden. So erlaubt die Untersuchung der parametrischen Komplexität, relativ konstant bleibende Parameter (z.B. die Anzahl der Attribute) zu isolieren. Durch die kostenbasierte Laufzeitabschätzung werden präzisere Aussagen zur Effizienz des CHASE möglich, die über ein Worst-Case Szenario hinaus gehen.

4. VORARBEITEN ZUR TERMINIERUNG

Die Thematik der CHASE-Terminierung wurde von uns bereits in vorbereitenden Arbeiten untersucht [14]. Hierbei bezogen wir uns konkret auf die Anwendungsfälle der Datenbanktransformation und des Data Cleanings. Aufgrund dieser Spezialisierung beschränkten wir uns auf das CHASE-Objekt der Datenbankinstanz. Da sich nahezu alle bisherigen Untersuchungen zur CHASE-Terminierung auf abstrakte Eigenschaften der CHASE-Parameter (also der Integritätsbedingungen) beziehen und unabhängig vom betrachteten CHASE-Objekt sind, können unsere Erkenntnisse jedoch auf andere CHASE-Anwendungsgebiete (z.B. Anfragetransformation) verallgemeinert werden.

Der Schwerpunkt der Vorarbeiten lag auf dem Vergleich der bisher bekannten Terminierungskriterien des Standard-CHASE und der Implementierung mehrerer dieser Kriterien. Obwohl das Problem der CHASE-Terminierung nämlich im Allgemeinen unentscheidbar ist, gibt es Klassen von Integritätsbedingungen, für die der CHASE stets terminiert (Abbildung 1).

Indem wir vor Durchführung des eigentlichen CHASE den verwendeten CHASE-Parameter auf seine Zugehörigkeit zu einer dieser Klassen überprüfen, können wir die Terminierung des CHASE unabhängig vom verwendeten CHASE-Objekt garantieren. Die meisten dieser Tests beziehen sich zwar nicht auf den Standard-CHASE, sondern auf den Skolem-Oblivious-CHASE, können jedoch auch für ersteren verwendet werden. Des Weiteren berücksichtigen die meisten Terminierungstests nur eine Form von CHASE-Parameter, und zwar TGDs, während EGDs entweder ignoriert werden können (z.B. im Fall der Schwachen Azyklizität) oder sogar zu fehlerhaften Testergebnissen führen können (z.B. im Fall der Azyklizität). Aus den verglichenen Terminierungskriterien wurden die Kriterien Schwache Azyklizität, Reiche Azyklizität, Safety und Azyklizität ausgewählt und in Form eines Terminierungstesters des CHASE-Werkzeugs ChaTEAU umgesetzt.

Schwache Azyklizität. Wenn eine TGD zu einer Endlosschleife des Standard-CHASE beiträgt, muss sie dazu in der Lage sein, beliebig oft neue Tupel zu generieren. Da

[2] In stratifizierten Mengen von TGDs sind Negation und Rekursion logisch getrennt.

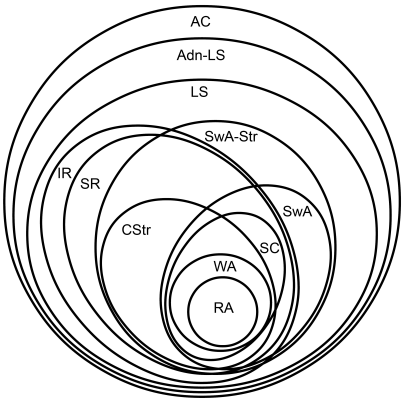

Abbildung 1: Hierarchische Beziehung einer Auswahl verschiedener Terminierungsklassen des CHASE (verändert nach [15]). RA: Reiche Azyklizität, WA: Schwache Azyklizität, SC: Safety, CStr: C-Stratifizierung, SwA: Superschwache Azyklizität, SwA-Str: SwA-Stratifizierung, SR: Safe Restriction, IR: Induktive Restriction, LS: Lokale Stratifizierung, Adn-LS: Constraint-Rewriting mit Lokaler Stratifizierung, AC: Azyklizität.

die ursprüngliche Datenbankinstanz als endlich angenommen wird, enthalten diese Tupel notwendigerweise Werte, die zuvor noch nicht in der Datenbankinstanz vorkamen – durch den CHASE zuvor erzeugte markierte Nullwerte. Diese Erkenntnis gilt ebenfalls für die eingebetteten TGDs, welche diese Nullwerte zuvor generiert hatten. Das Kriterium der Schwachen Azyklizität verfolgt die Weitergabe dieser Nullwerte anhand eines Graphen mit besonderen (Beitrag zur Erzeugung eines neuen Nullwertes) und gewöhnlichen Kanten (Weitergabe eines Wertes). Existiert in diesem Graphen ein Zyklus, der durch eine besondere Kante geht, ist die Terminierung des CHASE nicht garantiert.

Reiche Azyklizität. Im Gegensatz zum Standard-Chase überprüft der naive Oblivious-Chase die Aktivität des Triggers nicht. Gerät diese CHASE-Variante in eine Endlosschleife, so enthalten wenigstens die Trigger der beteiligten TGDs Werte, die zuvor nicht in der Datenbankinstanz vorkamen. Wir fordern also nicht, dass diese Nullwerte des Triggers in die neu erzeugten Tupel gelangen. Es werden folglich in zusätzlichen Fällen neue Nullwerte erzeugt. Wir ergänzen den Graphen der Schwachen Azyklizität daher um weitere spezielle Kanten. Die Terminierung des naiven Oblivious-CHASE kann nur dann garantiert werden, wenn der Graph keine Zyklen durch besondere oder spezielle Kanten enthält. In diesem Fall würde auch der Standard-CHASE terminieren, da die Terminierung des naiven Oblivious-CHASE hinreichendes Kriterium für die Terminierung des Standard-CHASE ist.

Safety. Für die Kriterien der Schwachen und Reichen Azyklizität wird zwischen der Erzeugung von Nullwerten und der Weitergabe von Werten unterschieden. Unter Umständen können wir allerdings ausschließen, dass bestimmte Variablen einer TGD mit einem Nullwert der Instanz belegt sind (indem wir Gleichheit mit einem Attribut fordern, welches keinen Nullwert enthalten kann). Für das Safety-Kriterium schränken wir die Weitergabe allgemeiner Werte auf die Weitergabe potentieller Nullwerte ein, wodurch wir die Mächtigkeit des Testkriteriums der Schwachen Azyklizität erhöhen.

Azyklizität. Bei diesem Terminierungskriterium wird die Azyklizität der gegenseitigen TGD-Aufrufung nicht untersucht (was einem wesentlich einfacheren Terminierungskriterium entspräche). Stattdessen werden die relationalen Atome der einzelnen TGDs um einen String sogenannter „Adornments“ ergänzt. Die Adornments werden auf eine Weise zwischen Atomen des TGD-Rumpfes und TGD-Kopfes einer TGD bzw. zwischen TGD-Kopf und TGD-Rumpf verschiedener TGDs weitergegeben, die dem CHASE-Algorithmus ähnelt. Diese Übertragung und Neuentstehung von adornten Atomen wird durch einen Graphen modelliert, dessen Zyklenfreiheit die Terminierung des CHASE garantiert.

Durch die Nutzung des ebenfalls implementierten Constraint-Rewriting-Algorithmus der Substitutionslosen Simulation verarbeitet der Terminierungstester neben TGDs auch EGDs. Gerade in Verbindung mit Substitutionsloser Simulation zeichnet sich der mächtigste implementierte Test – der Test auf Azyklizität – jedoch durch eine nicht adäquate Laufzeit auf. Die exponentielle Komplexität des zugrunde liegenden Algorithmus ist zwar bereits aus der Literatur bekannt [19], dies betrifft jedoch zum einen den Worst-Case, zum anderen Parameter, die in praktisch relevanten Szenarien relativ beschränkt sind. Durch Laufzeitanalysen der Implementierung konnten wir zeigen, dass die Komplexität des Tests auf Azyklizität auch in relativ kleinen Anwendungsfällen nicht handhabbar ist, wenn EGDs berücksichtigt werden. Im konkreten Anwendungsfall sollte daher zunächst ein effizienterer Terminierungstest durchgeführt werden.

Diese Vorarbeiten hinsichtlich der CHASE-Terminierung sollen in Folgearbeiten aktualisiert und durch vergleichbare Untersuchungen zur Konfluenz ergänzt werden.

5. ERSTE ERWEITERUNGEN: NEGATION UND KONFLUENZ

Um die Inkonfluenz des CHASE anschaulicher diskutieren zu können, ergänzen wir die in Unterabschnitt 3.1 definierte Beispieldatenbank um eine Tabelle für Noten des Studiengangs Informatik und einer Tabelle für alle eingeschriebenen Studierenden. Wir nehmen an, dass die Tabelle der Noten aller Studierenden aus Effizienzgründen horizontal fragmentiert wurde, sodass jeder Studiengang über eine separate Notentabelle verfügt. Die Relation Noten_Informatik besteht aus den Attributen Matrikelnummer (Ma), Modulnummer (Mo) und Note (No), während die Attribute der Studierenden-Relation Matrikelnummer (Ma), Name (Na) und Studiengang (St) sind.

Im Folgenden orientieren sich die Namen der Variablen an den Abkürzungen der Attribute, in denen sie stehen, wobei existenzquantifizierte Variablen durch zwei Großbuchstaben, allquantifizierte Variablen hingegen durch zwei Kleinbuchstaben gekennzeichnet sind. Aufgrund dieser Konvention kann auf Quantoren verzichtet werden.

Über die Matrikelnummer der Studierenden sei eine Fremdschlüsselbeziehung zwischen beiden Relationen definiert. Die folgende TGD r_1 beschreibt die hiermit verbundene Inklusionsabhängigkeit:

$$r_1 : \text{Noten\_Informatik}(ma, mo, no) \rightarrow \text{Studierende}(ma, NA, ST).$$

Wie in Unterabschnitt 3.2 erwähnt, kann der Test der Triggeraktivität als Negation verstanden werden:

$$r_1' : \text{Noten\_Informatik}(ma, mo, no), \neg\text{Studierende}(ma, NA_2, ST_2) \rightarrow \text{Studierende}(ma, NA, ST).$$

Wird der CHASE ausschließlich auf volle TGDs angewandt, ist er monoton, womit sich seine Konfluenz begründen lässt. Wenn der CHASE durch Erweiterungen ein nicht-monotones Verhalten zeigt, führt dies also unter Umständen zur Inkonfluenz des CHASE. Beispiele für derartige nicht-monotonen Erweiterungen sind Negation (einschließlich des Tests der Triggeraktiviertheit) und Deletion. Letzteres bezieht sich etwa auf das Verschmelzen zweier Tupel nach Anwendung einer EGD. Allerdings führen EGDs unter Umständen divergente Datenbankzustände auch zusammen und etablieren so Konfluenz.

Betrachten wir zunächst einen einfachen Fall von Inkonfluenz auf Grundlage der zuvor definierten Inklusionsabhängigkeit r_1. Wir ergänzen eine weitere Inklusionsabhängigkeit für die Noten-Tabelle der Mathematikstudenten. Hier wollen wir jedoch zusätzlich festlegen, dass Studierende, welche Noten in Mathematik-Modulen erhalten, auch im Studiengang Mathematik eingeschrieben sind:

$$r_2 : \text{Noten\_Mathematik}(ma, mo, no) \rightarrow \text{Studierende}(ma, NA, \text{„Mathematik“}).$$

Objekt des CHASE sei folgende Datenbankinstanz I:

$$\begin{aligned}\mathrm{I} = \{&\text{Noten\_Informatik}(2, \text{„Datenbanken III“}, 4.0),\\ &\text{Noten\_Mathematik}(2, \text{„Stochastik I“}, 1.0),\\ &\text{Studierende}(1, \text{„Mustermann“}, \text{„Informatik“})\}.\end{aligned}$$

Wenden wir zuerst r_1, und dann r_2 an, werden zwei neue Tupel erzeugt. Das gleiche Ergebnis erhalten wir, wenn die Aktivität des Triggers nicht getestet wird oder beide Regeln parallel ausgewertet werden:

$$\begin{aligned}\mathrm{I}' = \mathrm{I} \cup \{&\text{Studierende}(2, \eta_1, \eta_2),\\ &\text{Studierende}(2, \eta_3, \text{„Mathematik“})\}.\end{aligned}$$

Wenden wir r_2 zuerst an, wird der Trigger von r_1 deaktiviert, und kein zusätzliches Tupel kann erzeugt werden:

$$\mathrm{I}'' = \mathrm{I} \cup \{\text{Studierende}(2, \eta_1, \text{„Mathematik“})\}.$$

Das Ergebnis der Regelanwendung ist hier abhängig von ihrer Reihenfolge. Beide Ergebnisse erfüllen alle gegebenen Abhängigkeiten und sind universelle Lösungen. Sind wir nur an der Existenz eines bestimmten Eintrags interessiert, liefen beide Reihenfolgen dasselbe Ergebnis. Wollen wir hingegen die Anzahl vorhandener Tupel bestimmen – z.B. durch Verwendung der Aggregatfunktion `COUNT()` in der Datenbankanfrage – lässt sich eine sichere Antwort lediglich in Form eines Intervalles ermitteln.

Nehmen wir an, dass nicht alle Studierenden, die Noten im Studiengang Informatik erworben haben, auch Informatik studieren, sondern Informatikmodule etwa nur als Nebenfach besuchen. Mathematik-Studierende, die Informatik als Nebenfach belegen, haben durch ihre Hauptfach bereits einen Eintrag in der Relation Studierende, sollten dort aber zusätzlich als Informatikstudenten geführt werden, wenn sie entsprechende Module absolviert haben:

$$\begin{aligned}r_2' : &\text{Noten\_Informatik}(ma, mo, no),\\ &\text{Studierende}(ma, na, \text{„Mathematik“})\\ &\rightarrow \text{Studierende}(ma, NA_2, \text{„Informatik (Nebenfach)“}).\end{aligned}$$

Erweitern wir die zuvor verwendete Instanz um eine Mathematikstudentin:

$$\begin{aligned}\mathrm{I}_2 = \{&\text{Noten\_Informatik}(2, \text{„Datenbanken III“}, 4.0),\\ &\text{Noten\_Mathematik}(2, \text{„Stochastik I“}, 1.0),\\ &\text{Studierende}(1, \text{„Mustermann“}, \text{„Informatik“})\\ &\text{Studierende}(2, \text{„Musterfrau“}, \text{„Mathematik“})\}.\end{aligned}$$

Obwohl r_2 und r_2' nahezu identische Köpfe aufweisen, liegt keine Inkonfluenz zwischen r_1 und r_2' vor. Bei der Untersuchung von Inkonfluenz nehmen wir an, dass mehrere Regeln gleichzeitig angewandt werden könnten. Wir setzen für r_2' jedoch voraus, dass Mathematik-Studierende bereits einen Eintrag in der Relation Studierende besitzen. Unter dieser Annahme (die in I_2 zutrifft) kann r_1 aber nicht angewandt werden:

$$\mathrm{I}_3 = \mathrm{I}_2 \cup \{\text{Studierende}(2, \eta_1, \text{„Informatik (Nebenfach)“})\}.$$

Varianten dieser Überlegung können genutzt werden, um zyklische Beziehungen zu untersuchen. Wir definieren hierfür, dass Studierende, die Mathematik studieren, auch mindestens einen Eintrag in der Relation Noten_Mathematik besitzen (vor Absolvieren ihres ersten Moduls erhalten Studierende etwa einen Platzhalter-Eintrag in dieser Tabelle):

$$r_3 : \text{Studierende}(ma, na, \text{„Mathematik“}) \rightarrow \text{Noten\_Mathematik}(ma, MO, NO).$$

Offensichtlich sind r_2 und r_3 zyklisch. Tupel, die von der einen TGD erzeugt werden, können von der jeweils anderen Regel als Trigger verwendet werden. Dennoch besteht (bei Verwendung des Standard-CHASE) keine Gefahr, in einer Endlosschleife unendlich viele Tupel zu generieren. Wenn wir r_2 anwenden, erzeugen wir für einen Studierenden der Relation Noten_Mathematik, dessen Matrikelnummer bekannt ist, einen Eintrag in der Relation Studierende. Es existieren nun also Einträge für den Studierenden mit besagter Matrikelnummer in beiden Tabellen. Wenn wir für diesen Studierenden r_3 anwenden wollen, darf in der Noten_Mathematik-Tabelle noch kein Eintrag mit dieser Matrikelnummer existieren (alle anderen Attribute sind nicht festgelegt), was für diesen Studierenden aber nicht zutrifft. Tatsächlich ist dieser Gedankengang Grundlage eines einfachen Terminierungskriteriums des CHASE, der Schwachen Azyklizität.

6. FAZIT

Der CHASE-Algorithmus ist ein universeller Algorithmus der Datenbanktheorie. Um ihn tatsächlich universell in praktischen Anwendungsszenarien einsetzen zu können, werden Erweiterungen – wie Negation oder Funktionssymbole – benötigt.

Es ist bereits abzusehen, dass diese Erweiterungen Konfluenz und Terminierung beeinflussen. In der vorliegenden Arbeit wurde dargelegt, dass bereits der etablierte CHASE eine Form von Negation implementiert und daher inkonfluent ist.

Da die Nicht-Terminierung des CHASE auf der Generierung neuer Werte beruht, ist offensichtlich, dass skalare Funktionsterme eine ähnlichen Auswirkung wie existenzquantifizierte Variablen auf die CHASE-Terminierung haben. Da Konfluenz, Terminierung und Effizienz also wohl nicht im Allgemeinen für den erweiterten CHASE garantiert werden können, müssen Spezialfälle identifiziert werden, in denen der CHASE effektiv auf die untersuchten Anwendungsfälle anwendbar ist.

7. DANKSAGUNG

Diese Arbeit wurde durch ein Stipendium der Landesgraduiertenförderung finanziell unterstützt.

8. LITERATUR

[1] F. N. Afrati and P. G. Kolaitis. Repair checking in inconsistent databases: algorithms and complexity. In *ICDT*, volume 361 of *ACM International Conference Proceeding Series*, pages 31–41. ACM, 2009.

[2] A. V. Aho, Y. Sagiv, and J. D. Ullman. Efficient optimization of a class of relational expressions. *ACM Trans. Database Syst.*, 4(4):435–454, 1979.

[3] T. Auge. Extended provenance management for data science applications. In *PhD@VLDB*, volume 2652 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2020.

[4] T. Auge and A. Heuer. Prosa - using the CHASE for provenance management. In *ADBIS*, volume 11695 of *Lecture Notes in Computer Science*, pages 357–372. Springer, 2019.

[5] J. Baget, M. Leclère, M. Mugnier, S. Rocher, and C. Sipieter. Graal: A toolkit for query answering with existential rules. In *RuleML*, volume 9202 of *Lecture Notes in Computer Science*, pages 328–344. Springer, 2015.

[6] M. Benedikt, J. Leblay, and E. Tsamoura. PDQ: proof-driven query answering over web-based data. *Proc. VLDB Endow.*, 7(13):1553–1556, 2014.

[7] D. Carral, I. Dragoste, L. González, C. J. H. Jacobs, M. Krötzsch, and J. Urbani. Vlog: A rule engine for knowledge graphs. In *ISWC (2)*, volume 11779 of *Lecture Notes in Computer Science*, pages 19–35. Springer, 2019.

[8] D. Carral, I. Dragoste, and M. Krötzsch. Restricted chase (non)termination for existential rules with disjunctions. In *IJCAI*, pages 922–928. ijcai.org, 2017.

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

[10] A. Deutsch, L. Popa, and V. Tannen. Physical data independence, constraints, and optimization with universal plans. In *VLDB*, pages 459–470. Morgan Kaufmann, 1999.

[11] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: semantics and query answering. *Theor. Comput. Sci.*, 336(1):89–124, 2005.

[12] W. Fan, Z. Fan, C. Tian, and X. L. Dong. Keys for graphs. *Proc. VLDB Endow.*, 8(12):1590–1601, 2015.

[13] F. Geerts, G. Mecca, P. Papotti, and D. Santoro. An overview of the llunatic system. In *SEBD*, pages 159–166, 2014.

[14] A. Görres. *Erweiterung des CHASE-Werkzeugs ChaTEAU um ein Terminierungskriterium.* Masterthesis, Universität Rostock, 2020.

[15] S. Greco, F. Spezzano, and I. Trubitsyna. Stratification criteria and rewriting techniques for checking chase termination. *Proc. VLDB Endow.*, 4(11):1158–1168, 2011.

[16] H. Grunert and A. Heuer. Query rewriting by contract under privacy constraints. *Open J. Internet Things*, 4(1):54–69, 2018.

[17] D. Maier, A. O. Mendelzon, and Y. Sagiv. Testing implications of data dependencies. *ACM Trans. Database Syst.*, 4(4):455–469, 1979.

[18] R. D. Ndindi. *Repairing data with conflict-free conditional dependencies.* Masterthesis, University of Edinburgh, 2011.

[19] F. Spezzano. *On the Problem of Checking Chase Termination.* PhD thesis, Università della Calabria, 2011.