Termination and Confluence of an Extended CHASE Algorithm

Andreas Görres
Supervised by Prof. Andreas Heuer

Computer Science Department, University of Rostock
18051 Rostock, Germany
andreas.goerres@uni-rostock.de

Abstract. Many requirements of systems managing and analyzing large volumes of data are interconnected and should therefore be realized together. In this research project, we use a fundamental algorithm of database theory – the CHASE – to address those requirements in a unified manner. While highly expressive, the language of the CHASE is still inadequate to formulate many problems of practical importance. Extending the CHASE with additional features would increase its range of applications, but might jeopardize its key features confluence, safe termination and efficiency.
In this work, we demonstrate that calculating basic linear algebra operations with the CHASE is feasible after extending the algorithm with negation and a restricted set of scalar functions. We discuss how confluence and termination of the CHASE are influenced by these extensions.

Keywords: CHASE · Data Science Pipeline · Termination · Confluence · Efficieny · Relational Algebra · Linear Algebra · Negation.

1 Introduction and Motivation

While processing large amounts of data is a traditional part of database management, big data analytics becomes more and more ubiquitous. Especially applications like the Internet of Things generate large amounts of information, opening up new areas of application for classic database algorithms, e.g., analyzing big data by extended database languages, such as linear algebra extensions to relational algebra.

Our research is mainly based on two application areas: Research data management and smart assistive systems. Use cases for research data management result from our long-time cooperation with the Institute for Baltic Sea Research (IOW). Resulting from this research, we gain insights into the (e.g. statistical) data analysis operations used in real world applications. Furthermore, the historical development of the used database is base for our research on schema evolution and on database transformation operations. Basically, published research data referring to a historical database schema should be reproducible even in the future. To allow the reproduction of results, the involved tuples need

to be saved, not the entire database of raw measurement data. Identifying those tuples motivates our research on provenance. However, simply publishing the entire used data set is not always possible. Especially medical data might contain personal information, resulting in the problem to guarantee privacy while still allowing the reproduction of results. This problem is even more pronounced in our second application area, smart assistive systems. The behavior of a user, e.g. a patient, is observed by a large amount of sensor data, so there current activities and intentions can be predicted. If parts of the analysis can be done on an early level of computation, in a completely distributed manner near the sensors or even by the sensors themselves, collecting personal information on a central server is no longer needed. This not only results in more privacy (privacy by design), but even in a more efficient, parallel evaluation of data. As before, our need for privacy counteracts our interest in reproducibility. However, tracing back results to the original data allows us to focus on a small amount of sensors having an actually significant influence on the calculation. This way, we could not only increase efficiency by disregarding unneeded data, but even realize privacy. Ideally, our three main objectives provenance, privacy and efficiency exhibit synergistic effects if we include them into a unified framework.
In our research, we try to realize those features using a fundamental algorithm of database theory: the CHASE.

2 Formalizing Problems in Terms of the CHASE

More than forty years ago, the CHASE algorithm was introduced in two seminal works [3, 10]. Right from the start, seemingly unrelated use cases – query optimization and schema construction – were examined in a unified way . Later on, the number of application areas (for slightly adapted variants of the CHASE) became even broader and the concept of "universal solution" was introduced to describe the connection between the different problem cases [8]. In this tradition, we understand the CHASE as a universal algorithm that is able to process a variety of parameters (e.g. integrity constraints, privacy constraints, and view definitions) into a variety of database objects (e.g. database instances or database queries).

Formally, let P be a parameter and O be an object, then the $\text{CHASE}_P(O)$ applies the parameter P to the database object O. If P is a query and O a relational database, the result of the CHASE is the result relation of the query P. If P are integrity constraints and O is a database query, the result of the CHASE is a query implicitly satisfying all the constraints, which is needed, e.g., to apply semantic query optimization techniques. We refer to the result of the CHASE as target $T = \text{CHASE}_P(O)$.

For our research in research data management and smart assistive systems, we need to combine the following steps of a data science pipeline: (1) Designing and evolving database schemas; (2) Analyzing data by means of a relational database language, extended by linear algebra operations; (3) Supporting reproducibility of the data analysis by data provenance techniques such as

why-provenance; and (4) Guaranteeing privacy (as well as integrity) constraints within the analysis and reproducibility steps. To be able to unify and combine these four tasks, we use the CHASE algorithm in the following ways.

- To describe the evolution of database schemes, we interpret P as a schema mapping and O as the source database schema, resulting in the target database schema T after the evolution.
- To perform an extended query as a representation of a data analysis, we interpret P as a query and O as the relational database, resulting in the target relation, i.e. the query result T.
- To calculate the why-provenance of the query used for the data analysis, we interpret the query result T as the new database object O and *invert* the query operations to represent the provenance query, i.e. the new parameter P. The new result T of this provenance query is a sub-database of the original database, the set of witnesses for the data analysis.
- To satisfy all the constraints, such as integrity or privacy constraints, while performing the data analysis by an extended relational query, we use the extended query as the database object O and the constraints as the CHASE parameter P. The result of the CHASE is a query respecting all the constraints in P.

The CHASE can be interpreted as a rule system, applying different rules in P to the logical representation of O. Since P is a set of rules, there is no fixed order in which the rules are applied to the object O. So we have the basic problems of **termination** (does the application of rules stop?) and **confluence** (if we apply the rules in a different order, is the algorithm always calculating the same result?). As a third problem, we have to check the **efficiency** of the CHASE, since in general the CHASE is often of higher complexity than specialized solutions for the above problems considered in isolation.

Even though, the classical CHASE is too restricted for most practical applications and therefore needs to be extended. In the classical CHASE, P and O are simple formulas of first-order predicate logic. For example, if P or O are used as database queries, the queries are restricted to conjunctions of positive select-project-join queries, with equality tests being the only permitted select operation.

3 Main Tasks of the PhD Thesis

For our PhD project, we define two main tasks: Unification and extension of the CHASE. As we described before, parameter and object of the CHASE depend on the application area. Even though the same parameters (e.g. integrity constraints) could be used for different CHASE objects, the actual behavior of the algorithm differs. For example, the same constraint might lead to the creation of an existentially quantified variables in a query and to the introduction of a new null value in a database instance. Most implementations of the CHASE algorithm are tailored towards a certain use case, for instance query optimization

or database repair. Even in this paper, we focus on the CHASE on database instances. Ultimately, the different variants of the CHASE need to be unified, resulting in a truly universal algorithm.

To address real life use cases in an effective manner, the CHASE needs to be extended, for example with negation, functions and arithmetic comparisons. However, extending the CHASE might influence its termination behavior, confluence and efficiency. Concerns regarding those properties are not restricted to the extended CHASE. Originally, CHASE parameters were restricted to functional dependencies and full inclusion dependencies, which ensured a finite and unique CHASE result. Once embedded dependencies were introduced to the CHASE, termination and confluence were no longer guaranteed. Still, those problems of the original algorithm can be addressed, for example by testing for termination beforehand [5]. There were attempts to extend the CHASE in the past (section 4). However, those extensions are usually tailored towards a specific use case and not the universal CHASE. The semantics of most CHASE extensions depends on the CHASE object: Negation in a CHASE parameter refers to tuples absent from a CHASE instance, but to explicitly negated atoms in a query. Furthermore, it is unclear if the results of a specialized extended CHASE (e.g. annotations encoding conditions for certain tuples) can be interpreted by the next step of data processing.

As described in section 2, the second step of the data science pipeline considers linear algebra operations as an extension of relational database languages. At the end of this paper, we present a simple CHASE program calculating matrix addition. Using this simple example, we explain the procedure of the CHASE and demonstrate the necessity for two CHASE extensions, negation and scalar functions. Furthermore, we examine why the presented program is confluent and terminates. In future works, we plan to evaluate practical effectiveness of the described extended universal CHASE based on a prototypical implementation.

4 Related Work

Previous efforts to extend the CHASE focused on arithmetic comparisons and negation. Along the way, complexity and termination behavior of these CHASE variants were examined. In contrast, arithmetic functions seem to be of lesser interest for current CHASE research. In [2], a CHASE variant extended with arithmetic comparisons is used to describe data exchange. This AC-CHASE tree considers all possible orders of null values, therefore generating an exponential number of possible results. In a similar manner, [5] uses the C-CHASE tree to answer queries with negation on ontologies. Recently, [7] simulated the disjunctive CHASE with the non-disjunctive variant of the algorithm. This way, disjunctive properties of arithmetic comparisons and negation can be described using the standard CHASE. However, this solution is not guaranteed to terminate in polynomial time and is therefore not necessarily more efficient than the tree-like CHASE variants. In fact, if we restrict the AC-CHASE to certain combinations of arithmetic comparisons, we preserve the so called *homomorphism property*,

thereby achieving polynomial data complexity of the terminating CHASE. This result extends to other application areas of the CHASE, like query optimization [1].

5 The CHASE Algorithm

The CHASE is a fixpoint algorithm incorporating CHASE parameters into objects, so that the resulting objects implicitly contain the parameters. CHASE parameters are expressed as logical implications. For equality generating dependencies (EGDs) and tuple generating dependencies (TGDs), a general algorithm can be found in [6]. In this work, we focus on TGDs:

$$TGD : \phi(\boldsymbol{x}, \boldsymbol{y}) \rightarrow \exists \boldsymbol{Z} : \psi(\boldsymbol{x}, \boldsymbol{Z})$$
$$EGD : \phi(\boldsymbol{x}) \rightarrow x_1 = x_2; x_1, x_2 \in \boldsymbol{x}.$$

In the next subsections, we extend the general schema of a TGD with negation and simple scalar functions. Similar to [5] and unlike [8], we do not restrict negation to single atoms. CHASE objects, which include database instances and queries, are encoded as sets of relational atoms.

The CHASE consists of a sequence of CHASE steps. In each step, a homomorphism (the trigger) between all atoms of a (nondeterministically chosen) CHASE parameter's body and some atoms of the CHASE object is defined. If the image of the TGD head atoms (possibly extended for existentially quantified variables) is not present in the CHASE object, an image of the head atoms is materialized in the CHASE object. For each existentially quantified variable, a fresh marked null value (or existentially quantified variable) is generated. The CHASE continues until a fixpoint is reached. If the TGDs are cyclic and existentially quantified variables are present in any head atom, the CHASE might not terminate. The problem of CHASE termination is, in general, undecidable, but well researched [5].

5.1 Introduction to CHASE Extensions Using Matrix Addition as an Example

Basic linear algebra operations are fundamental for machine learning algorithms used in big data analytics. In the following, we show how matrix addition can be defined using extended TGDs, a necessary requirement for reasoning about the algorithm with the CHASE. Let us consider matrices A and B, encoded in relations $A(I, J, V)$ and $B(I, J, V)$. Coordinates of a matrix field are encoded in attributes I and J and the field value in attribute V. Matrix AB encodes the sum of the matrices A and B.

$$\begin{array}{ccc} \text{A} & \text{B} & \text{AB} \\ \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} & \begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix} & \begin{pmatrix} 2 & 0 \\ 1 & 1 \end{pmatrix} \end{array}$$

To illustrate the basic working of an extended CHASE algorithm, we define a simple program calculating matrix addition:

$$r_1 : A(i_1, j_1, v_1), B(i_1, j_1, v_2) \rightarrow AB(i_1, j_1, sum(v_1, v_2))$$
$$r_2 : A(i_1, j_1, v_1), \neg(\exists V_2 : B(i_1, j_1, V_2)) \rightarrow AB(i_1, j_1, v_1)$$
$$r_3 : \neg(\exists V_1 : A(i_1, j_1, V_1)), B(i_1, j_1, v_2) \rightarrow AB(i_1, j_1, v_2).$$

The only new values created by the standard CHASE are marked null values (and variables). In addition to this, we can generate new constants if we allow scalar function terms, like $sum()$, in the TGD head.

We define a homomorphism from the body of r_1 to the matrix fields with the coordinates $(1, 1)$ in both matrices. Consequently, the CHASE generates a tuple $AB(1, 1, 2)$, the image of r_1's head. Notice that we immediately calculated the result of the scalar function instead of generating a nested term in AB. Of course, by saving the nested function term in relation AB, we might be able to optimize the term later on using arithmetic transformations, but the CHASE is ill-equipped for this kind of optimization.

While r_1 is sufficient to calculate the result of matrix addition if all matrix fields are represented by tuples in the database, this is not the case if we use the compressed database representation of matrices defined in [11], which allows a more efficient treatment of sparsely populated matrices. Here, fields with value zero are represented by missing relational tuples. Therefore, the tuple $B(2, 2, 0)$ is absent from this representation and we are unable to find a homomorphism for the coordinates $(2, 2)$. Consequently, no tuple with those coordinates is generated in AB, even though we expect this field to have a value of one. In [11], this problem is addressed using an outer join. We adapt this approach with additional TGDs r_2 and r_3 containing negation.

After defining the mapping $\{i_1 \mapsto 2, j_1 \mapsto 2, V_1 \mapsto 1\}$ for the positive atoms of r_2, we rewrite the negative atom into the the following boolean subquery $\exists V_2 : B(2, 2, V_2) \rightarrow ()$ by substituting the variables i_1 and j_1 with their respective mappings. A boolean query is basically a TGD whose head consists of an empty tuple. Since we are unable to define a consistent homomorphism for the boolean query, we are unable to generate this tuple, which is interpreted as "false". Consequently, we proceed by materializing the image of the head atom, $AB(2, 2, 1)$. Notice that the body of a boolean query might have multiple atoms and can contain existentially quantified variables (in this example V_2). Variables in negated atoms not present in any positive atoms are known as *unsafe*. In this example, we could avoid unsafe negation by defining a view that projects over attributes containing the safe variables. However, by interpreting negation as a negated subquery (with explicit quantification of unsafe variables) we show that multiatomic unsafe negation does not pose a challenge to the CHASE on database instances.

5.2 Confluence and Termination of the Example

While negation is often (even in the previous example) restricted to single atoms (negative subgoals) and safe variables, exceeding those limitations is surprisingly

natural for the extended CHASE. In fact, the CHASE algorithm already utilizes this kind of generalized negation when testing trigger activity. After finding a valid homomorphism for the body atoms of a TGD (the trigger), we test if the image of the head atoms is already present in the CHASE object. This active trigger test can be interpreted as an implicit multiatomic negation, treating the TGD head as a negated conjunction of body atoms. For this, existentially quantified variables from the TGD head are implicitly renamed and act as unsafe variables. While the CHASE on full TGDs is confluent, the general CHASE algorithm is not. One main reason for this behavior is the previously described test for trigger activity. Since this test can be expressed using negation, it is unsurprising that general negation leads to additional cases of non-confluence. If a TGD is blocked by tuples a second TGD generates, the order in which both TGDs are applied directly influences the result. This remains valid even if we restrict ourselves to stratified negation (that is, there are no circular dependencies between TGDs containing negation). Stratified negation simply guarantees there is an order of rule application in which blocking TGDs are applied before the respective blocked TGDs. The presented CHASE program, however, is confluent since it is "semi-positive": Only tuples are negated whose relations never appear in any TGD head.

The main purpose of scalar functions in the given example is the generation of new constants. In this regard, they have similar effects as existentially quantified variables in head atoms, which also contribute to the creation of new values (null values or variables). Similar to existentially quantified variables, scalar functions might lead to a non-terminating CHASE sequence. However, adjustment of classical termination tests, like *Weak Acyclicity* [9], would still guarantee CHASE termination. Being non-recursive, the given CHASE program is weakly-acyclic and guaranteed to terminate. However, there is a major difference between null values and the constants generated by a scalar function: Only by applying an explicitly defined EGD, we can unify two different null values. This way, the application of an EGD can terminate a CHASE sequence that might otherwise be infinite. It might also unify the results of two alternative CHASE sequences, thereby guaranteeing confluence. For scalar functions, this unification is not defined explicitly by the CHASE parameter, but by arithmetic rules instead (e.g. the commutativity of addition). Furthermore, two attribute values might not even be identical, but converge to the same constant in the progress of an infinite CHASE sequence.

6 Conclusions and Future Work

In this work, we extended the CHASE on database instances with negation and scalar functions, exemplified by a simple CHASE program. In a similar manner, we have defined more complex, but still confluent and terminating programs. Since these programs are unions of (extended) TGDs, CHASE techniques used for optimizing unions of conjunctive queries could be used to optimize them. However, while unsafe negation was smoothly integrated into the CHASE on

instances, it poses a serious challenge to the CHASE on queries. In future works, we will describe our solution to these problems. Furthermore, we intend to evaluate this solution by implementing it into our prototypical CHASE software ChaTEAU [4].

The CHASE algorithm of classic database theory can be applied to a multitude of problem cases, solving them in a unified manner. In this regard, interactions between the requirements provenance, privacy and efficiency are of particular interest to us. However, for practice-oriented use cases, extensions of the algorithm are needed. These extensions might affect efficiency, confluence and termination of the algorithm in a negative way.
In this work, we illustrated the prospects of an extended CHASE algorithm with a simple framework calculating linear algebra operations. As a next step of our research, we will examine how CHASE programs can be modified using the CHASE algorithm, thereby contributing to the solution of the initially formulated requirements.

Acknowledgements This work was supported by a scholarship of the Landesgraduiertenförderung Mecklenburg-Vorpommern.

References

1. Afrati, F.N.: The homomorphism property in query containment and data integration. In: IDEAS. pp. 2:1–2:12. ACM (2019)
2. Afrati, F.N., Li, C., Pavlaki, V.: Data exchange in the presence of arithmetic comparisons. In: EDBT. ACM International Conference Proceeding Series, vol. 261, pp. 487–498. ACM (2008)
3. Aho, A.V., Sagiv, Y., Ullman, J.D.: Efficient optimization of a class of relational expressions. ACM Trans. Database Syst. **4**(4), 435–454 (1979)
4. Auge, T., Heuer, A.: Prosa - using the CHASE for provenance management. In: ADBIS. Lecture Notes in Computer Science, vol. 11695, pp. 357–372. Springer (2019)
5. Baget, J., Garreau, F., Mugnier, M., Rocher, S.: Revisiting chase termination for existential rules and their extension to nonmonotonic negation. CoRR **abs/1405.1071** (2014)
6. Benedikt, M., Konstantinidis, G., Mecca, G., Motik, B., Papotti, P., Santoro, D., Tsamoura, E.: Benchmarking the chase. In: PODS. pp. 37–52. ACM (2017)
7. Bourgaux, C., Carral, D., Krötzsch, M., Rudolph, S., Thomazo, M.: Capturing homomorphism-closed decidable queries with existential rules. In: KR. pp. 141–150 (2021)
8. Deutsch, A., Nash, A., Remmel, J.B.: The chase revisited. In: PODS. pp. 149–158. ACM (2008)
9. Fagin, R., Kolaitis, P.G., Miller, R.J., Popa, L.: Data exchange: semantics and query answering. Theor. Comput. Sci. **336**(1), 89–124 (2005)
10. Maier, D., Mendelzon, A.O., Sagiv, Y.: Testing implications of data dependencies. ACM Trans. Database Syst. **4**(4), 455–469 (1979)
11. Marten, D., Meyer, H., Dietrich, D., Heuer, A.: Sparse and dense linear algebra for machine learning on parallel-rdbms using SQL. OJBD **5**(1), 1–34 (2019)