The Journal of Logic and Algebraic Programming 83 (2014) 20–52

Contents lists available at ScienceDirect

The Journal of Logic and Algebraic Programming

www.elsevier.com/locate/jlap

An extended constraint deductive database: Theory and implementation

Gabriel Aranda-López *, Susana Nieva, Fernando Sáenz-Pérez, Jaime Sánchez-Hernández

Facultad de Informática, Complutense University of Madrid, Spain

ARTICLE INFO

Article history:
Received 28 July 2011
Received in revised form 4 April 2013
Accepted 3 July 2013
Available online 9 July 2013

Keywords:
Deductive databases
Constraints
Hereditary Harrop formulas
Fixpoint semantics

ABSTRACT

The scheme of Hereditary Harrop formulas with constraints, $HH(\mathcal{C})$, has been proposed as a basis for constraint logic programming languages. In the same way that Datalog emerges from logic programming as a deductive database language, such formulas can support a very expressive framework for constraint deductive databases, allowing hypothetical queries and universal quantifications. As negation is needed in the database field, $HH(\mathcal{C})$ is extended with negation to get $HH_{\neg}(\mathcal{C})$. This work presents the theoretical foundations of $HH_{\neg}(\mathcal{C})$ and an implementation that shows the viability and expressive power of the proposal. Moreover, the language is designed in a flexible way in order to support different constraint domains. The implementation includes several domain instances, and it also supports aggregates as usual in database languages. The formal semantics of the language is defined by a proof-theoretic calculus, and for the operational mechanism we use a stratified fixpoint semantics, which is proved to be sound and complete w.r.t. the former. Hypothetical queries and aggregates require a more involved stratification than the common one used in Datalog. The resulting fixpoint semantics constitutes a suitable foundation for the system implementation.

1. Introduction

The extension of *LP* (Logic Programming) with constraints gave rise to the *CLP* (Constraint Logic Programming) scheme [26,25]. In a similar way, the $HH(\mathcal{C})$ scheme (Hereditary Harrop formulas with Constraints) [31,19] extends *HH* by adding constraints. In both cases, a parametric domain of constraints is assumed for which it is possible to consider different instances (such as arithmetical constraints over real numbers or finite domain constraints). The extension is completely integrated into the language: constraints are allowed to occur in goals, bodies of clauses, and answers.

As a programming language, $HH(\mathcal{C})$ can still be viewed as an extension of *CLP* in two main aspects. On the one hand, the logic *HH* introduces new connectives which are not available in Horn Clause logic, such as disjunction, implication and universal quantifiers [36]. On the other hand, and following Saraswat [45], in the scheme $HH(\mathcal{C})$, the notion of constraint system is established in such a way that any $\mathcal{C}$ satisfying certain minimal conditions can be considered as a possible instance for the scheme. In [45], as minimal conditions the language of constraints incorporates $\wedge$ and $\exists$. However, particular constraint systems may include more logical symbols as $\forall$ and $\Rightarrow$, together with the corresponding assumptions related to their behavior. Therefore, the language of constraints itself extends the common ones used in *CLP*, consequently facilitating the representation of more complex constraints.

* Corresponding author.
E-mail address: garanda@fdi.ucm.es (G. Aranda-López).

1567-8326/$ – see front matter
http://dx.doi.org/10.1016/j.jlap.2013.07.002

This paper extends other works [38,2] in which we investigated the use of $HH(\mathcal{C})$ not as a (general purpose) programming language, but as the basis for constraint deductive database (*CDDB*) systems [30,42]. The motivation is that, in the same way that Datalog [51,56] and Datalog with constraints [27] arise for modeling database systems inspired by *LP* and *CLP* respectively, the language $HH(\mathcal{C})$ can offer a suitable starting point for the same purpose. We show that the expressive power of $HH(\mathcal{C})$ improves existing languages by enriching the mechanisms for database definition and querying, with new elements that are useful and natural in practice. In particular, implications can be used to write hypothetical queries, and universal quantification allows encapsulation. The existence of constraints is exploited to represent answers and to finitely model infinite databases and answers. This is also the case of constraint databases, but the syntax of our constraints is also more expressive than the one commonly used in them, as it is the case of Datalog with constraints.

However, $HH(\mathcal{C})$, as it was originally introduced, lacks negation which, as we will see, is needed for our proposal to be complete with respect to relational algebra (*RA*). We have extended $HH(\mathcal{C})$ with negation, to obtain $HH_{\neg}(\mathcal{C})$ to be used as a database language. We have defined a proof-theoretic semantics to provide the meaning of goals (queries) and programs (databases). This meaning is represented by answer constraints, which can be obtained using the goal-oriented rules of a sequent calculus which combines intuitionistic inference rules with deductibility in a generic constraint system. Also, a stratified fixpoint semantics has been defined and proved to be sound and complete with respect to the previous one. The motivation for introducing this new semantics take into account several aspects:

- The fixpoint semantics provides a model for the whole database, while the proof-theoretic one (as well as top-down semantics in general) focuses only on the meaning of a query in the context of a database. In fact, the fixpoint of a database will correspond to the instance of the database. Thereby, the fixpoint semantics supplies a framework in which properties such as equivalence of databases can be easily analyzed, which gives formal support to the study of query optimization.
- In order to deal with recursion and negation, we have followed the stratified negation approach used in [51] which gives semantics to Datalog. The use of a fixpoint semantics as an operational mechanism has been adopted as a good choice in several deductive database systems as it is able to avoid the non-monotonicity inherent to negation. Then it guarantees termination, as long as the constraint answer sets are finite. Further, it facilitates to work with different constraint systems, relegating the problem of termination to the problem of finding intensional representations of data by the constraint system. This issue is dealt in works as [40], where safety conditions are imposed to the constraint system, and some particular systems are identified as satisfying such conditions. Hence, termination for any query is ensured for these systems. But, identifying such particular systems is out of the scope of this work.
- Introducing negation in goals makes a given database to may have several meanings [51]. Stratified negation is one of the approaches that, by imposing syntactical restrictions, guarantees a unique model for the database: the minimal fixpoint interpretation. Moreover, stratification has been previously used as a useful resource when dealing with hypothetical queries in Datalog [9], in order to get a unique minimal model for the database, as it is the case of our proposal.
- Stratification is a common technique to deal with aggregates [42], because it ensures monotonicity. Our stratified design of the fixpoint semantics has become a good framework to implement aggregates.

In order to define a stratified fixpoint semantics for $HH_{\neg}(\mathcal{C})$, we have adapted the usual notion of dependency graphs to include the dependencies derived from implications inside goals, as well as those derived from aggregate functions. The fixpoint of a database is computed as a set of pairs (A, C), where A is an atom and C a constraint. The atom A can be understood as an n-ary relation instance, where its arguments are constrained by C. According to the dependency graph, predicates are classified by strata and these pairs are computed by strata. Each stratum should become saturated before trying to saturate any other higher stratum. However, as an implication may occur in a goal, the computation must take into account that the database is augmented with the hypothesis posed in the implication antecedent. From the theoretical point of view, this issue does not make any obstacle, but it can be a drawback for a concrete implementation. In our approach, the fixpoint of the augmented database must be locally computed to solve the implication. But our proposal takes advantage of the stratification to avoid cycles during the computation. In addition, the dependency graph can be useful to reduce this computation to the part of the database that is involved in the implication.

The fixpoint semantics provides support for a concrete database system. We have implemented a Prolog prototype very close to the underlying theory as a proof-of-concept. Essentially, it incorporates all the features introduced in the paper, and moreover it supports aggregate functions. Two main components can be distinguished in the implementation of this database language. One corresponds to the implementation of the fixpoint semantics which is independent of the concrete constraint system. The other component corresponds to the implementation of the constraint system. We have considered and implemented solvers for the following constraint systems: Boolean, Reals, and Finite Domains, as instances of $\mathcal{C}$ in the scheme $HH_{\neg}(\mathcal{C})$. The implementation is designed in such a way that more than one constraint system can be used within the same database. We have designed a type system for identifying the constraint system each constraint in a database belongs to.

1.1. Related work

It is well known that negation in logic programming is a difficult issue and there has been a great amount of work about it [1], and also in constraint logic programming [46] and deductive databases [8] since long time ago. A first bag of issues