Higher Order Support in Logic Specification Languages for Data Mining Applications

Matthias van der Hallen
KU Leuven
(*e-mail:* `firstname.lastname@cs.kuleuven.be`)

submitted 29 April 2015; accepted 5 June 2015

Abstract

In this paper, we introduce our work on our doctorate with title "Higher Order Support in Logic Specification Languages for Data Mining Applications". Current logic specification languages, such as FO(·) provide an intuitive way for defining the knowledge within a problem domain.

Extended support for data representation is lacking however, and we want to introduce structured recursive types and generic types, together with a first class citizen approach to predicates. These additions correspond to higher order concepts.

We provide a background of the current techniques that might be of interest when implementing these higher order abstractions, such as lazy grounding and oracles. We sketch the eventual goal of our research, and give an overview of the current state and research questions that are being considered.

KEYWORDS: Logic Specification, Higher Order, Grounding, Lifted Reasoning

1 Introduction and Problem Description

Current logic specification languages, like FO(·) (Denecker and Ternovska 2008), offer many intuitive ways to specify the knowledge within a problem domain: aggregates, inductive definitions, ... However, it is still lacking abstractions that are commonplace in other paradigms, for example structured recursive types and generic types. Furthermore, some additional abstractions are possible in logic programming, such as the treatment of relations as first class citizens. These abstractions are very powerful and recognized ways of data representation, and they all correspond to higher order concepts.

We believe that the addition of these abstractions will make the life of programmers and other people who write logic specifications easier, and therefore will open up additional interest in logic specification languages. Additionally, it will allow one to write more modular specifications, and reuse them many different times in many different applications.

One example of an often reusable specification is the transitive closure of a binary

relation, as shown below:

$$\forall a, b : Closure(a, b) \leftarrow P(a, b).$$
$$\forall a, b : Closure(a, b) \leftarrow \exists : P(a, c) \wedge P(c, b).$$

Another example within the context of data mining is the concept of a *graph homomorphism*. A graph homomorphism is a constrained relation between two graphs. As such, it is natural to represent it as a predicate. However, data mining applications frequently use homomorphisms, isomorphisms or other 'relationships' as an object of their reasoning. For example, we want something to hold for no, some, or all homomorphisms. This treats these predicates as a first class citizen as it introduces quantification over these predicates.

The idea that modularity benefits from an expressive higher order language can be found in earlier work done conjunction with I. Dasseville et al. (Dasseville et al. 2016). Here, logic specification languages are extended with templates, a popular way of defining reusable concepts in a modular way, using second order definitions.

The main roadblock towards implementing the forementioned abstractions is that many logic specification languages use a ground-and-solve technique. The grounding phase transforms a logic specification theory to an equivalent theory on propositional level, allowing SAT techniques to be used. It achieves this by enumerating over finite domains, instantiating the theory for every possible substitution of variables by domain elements. However, we often do not want to fix our domain beforehand: we do not want to fix the number of connections or elements in pattern mining, or the names of items in item set mining. Furthermore, structured recursive types and generic types introduce infinite domains of their own, for example the set of all possible lists. As a result, it will be necessary to develop grounding techniques that can handle infinite domains.

2 Background and Overview of the Existing Literature

This research is focussed on providing higher order support to ground-and-solve systems such as IDP (Imperative Declarative Programming) (de Cat et al. 2014). The IDP system allows specifications specified in the FO($\cdot$) language which it subsequently grounds, i.e. translates to an equivalent theory on a *propositional* level. These translations can then be solved using a SAT-solver.

Other systems, such as Flora-2 (Kifer 2005), take a different approach. Flora-2 can translate specifications directly to Prolog for which it can use the XSB system (Warren 1998) with tabling as an inference engine. The Flora-2 system is based on F-Logic (Kifer et al. 1995) and HiLog (Chen et al. 1993). F-logic extends classical logic with the concepts of objects, classes, and types, and allows an object-oriented syntax. One of HiLogs defining characteristics is that it combines a higher order syntax with first order semantics so as to remain decidable. As any language with inductive definitions under the well-founded or the stable semantics is undecidable, this is less of an issue for a system with a language such as FO($\cdot$). Also, in recent times various other, simpler inference techniques such as model checking, model expansion or querying have gained importance with respect to deduction.

2.1 Relevant Techniques

Currently, systems that depend on a grounding phase do not combine with specifications including infinite domains as grounding introduces an exponential blowup when done in a naive way.

However, there are several interesting problems where it is not wise to restrict ourselves to a finite domain. Moreover, as argued in Section 1, our proposed additions and abstractions to allow more user-friendly data representations introduce these infinite domains.

Because of the possible blowup when working with large, (possibly infinite) domains, the design and implementation of novel grounding and inference techniques is necessary. These techniques must be capable of reasoning on the infinite domain in only finite time. While this is in general undecidable, various fields of computational logic and declarative problem solving have developed techniques that provide effective solutions for a broad and practically important class of such problems. We envision to combine existing methods, to generalize them and to add novel ones.

Our main context for this work will be the field of *lifted reasoning*. Possible (and interrelated) techniques in this field that aid with reasoning over infinite domains are:

Lazy grounding (De Cat et al. 2015; De Cat et al. 2012; Palù et al. 2009): The current state-of-the-art ground-and-solve systems keep a strict separation between the grounding and the solving phase. This means that a lot of computing time and storage space is spent on grounding the theory before the solver starts its search in solution space. For modellings where large or infinite domains are used, this grounding phase can be impossible, or at the very least, terribly inefficient. Lazy grounding is a technique that remedies this by (operationally) weaving these two phases together. This means that the solver can solve everything that is already grounded: i.e. make decisions and propagate them. These propagations and decisions then cause grounding to be generated for other sentences on demand, postponing as much of the grounding as possible and beneficial to the systems performance. This grounding behavior is called lazy, as it only does the work (and incurs the costs) related to grounding when it proves to be necessary for the solver.

Quantifier elimination and rewriting: Various techniques for the removal of quantifiers, or replacement of existential quantifiers by universal quantifiers (e.g. Skolemization) exist. Skolemization introduces functions; few state of the art solvers support functions, however the IDP system does and already makes use of Skolemization to reduce grounding size (De Cat et al. 2013).
Other elimination and rewriting techniques are possible, for example quantifier eliminations techniques inspired by the quantifier elimination used in Presburger arithmetic (Cooper 1972).

Bounds propagation for sets: Using several set-axioms and number-theoretic properties, it might be possible to deduce bounds for sets which are not constrained to a finite domain explicitly by the modelling. This greatly reduces

the costs of grounding as it can cause large domains to be limited to much smaller, better tractable domains.

Propagation of homogeneity in subdomains for cofinite sets: Even if a set contains an infinite number of elements, its elements might deviate from a simple default rule in only a finite number of situations. This means that there is a certain homogeneity or simple defining relation for the remainder of the sets domain, and, given this prescription for the set, only a small set of additional values carries additional (meaningful) information. Because of this, for several inference tasks it can prove unnecessary to enumerate the entire set, instead reasoning only on the combination of the prescription and the small complementary set to solve the task at hand.

Specialized algorithms: specialized and efficient algorithms for set operations (Dovier et al. 2006) and enumerations can be conceived, for example the exhaustive enumeration of graphs that satisfy certain logical conditions.

Oracles: The use of subsolvers as oracles is a promising avenue for supporting higher order quantification over predicates. The necessary search is performed by a subsolver which is given a theory that describes all constraints over the quantified predicate. Using negation, universal quantification over a predicate is turned into existential quantification. As a result, it is possible to rewrite every quantification into an existential Second Order theory (or search problem) that a subsolver can tackle.

The subsolver answers whether the new theory is satisfiable, and if so, the correct conclusions about the super-theory are inferred, and if possible, propagated. The performance of these subsolvers, as well as how much information can be shared between them and the optimal level of granularity that decides when a subsolver must run, is the subject of ongoing research.

As an example, consider a shipping dock with multiple separate docking segments and ships. We want to package a large payload in (as few as possible) smaller load that we can later distribute over the ships. However, we do not know which ship will be placed in which dock, and every dock and ship has a maximal load capacity that it can bear. We can express this as follows:

$$\begin{aligned}\exists \text{Weight}[\text{Package} \mapsto \text{Int}] : \forall \text{Place}[\text{Ship} \mapsto \text{Dock}] : \exists \text{Distri}[\text{Package} \mapsto \text{Ship}] : \\ \forall \text{ship} : \text{sum}\{\text{pack}[\text{Package}] : \text{Distri}(\text{pack}) = \text{ship} : \text{Weight}(\text{pack})\} \\ < \min(\text{dockCap}(\text{Place}(\text{ship})), \text{shipCap}(\text{ship})).\end{aligned}$$

Informally, this must be read as follows: there exists a function Weight that gives the weight for each package, such that for every placement of ships in docking segments there exists a distribution Distri of the packages over the ships such that the maximal capacity of neither ship nor dock is exceeded.

Here, the universal quantification over the placement predicate Place can be transformed to existential quantification by negating the remainder of the logical sentence, which will be solved by a subsolver. We will require that this subsolver proves unsatisfiability, as the transformation from universal to existential quantification introduces a negation before and after the quantification. Note that this introduces a negation before the existential quantifi-

cation of the Distri predicate, which in the naive schemes results in another subsolver.

These techniques are called "lifted reasoning" because the reasoning task is partly done at the predicate level, as opposed to the current state-of-the-art that defers all reasoning until after the grounding phase, when it can be done on propositional level.

The development and adaptation of these techniques will pose theoretical and implementation challenges.

Furthermore, significant software architectural questions must be answered to insert these techniques in the classical two phase system with the appropriate level of modularity and encapsulation.

3 Goal

The goal of this research is to add higher order support to logic specification languages running on ground-and-solve systems, specifically with data representation and data mining applications in mind. These higher order support will come in the form of structured recursive types, generic types, and relations as first class citizens.

We expect that these additional features make data handling and reasoning more natural, and that they allow us to write shorter, more specific and modular specifications. These modular specifications should then be combined into a larger specification for solving larger problems, leading the way for a *software design methodology* for logical specifications.

We expect to provide an implementation of the ideas and results stemming from this research for the FO$(\cdot)$ language and the IDP system that supports and provides inferences for this language.

4 Current State and Future Work

Current research topics consist of analyzing how the work on *lazy grounding* (De Cat et al. 2015), *justifications* (Denecker et al.) and *relevance* can be leveraged to work with infinite domains. The idea behind justifications is that given a certain interpretation, the truth value of every ground literal can be justified on the basis of the program. For example, given a true literal its justification could be the body of one of its rules for which the body is true. This is called a *direct justification*. Combining all these justifications results in the *justification graph*.

From all possible justification graphs, the *relevance* of certain literals can be deduced. Some literals are irrelevant: the truth values of these literals does not matter for the truth value of the program. Combining relevance derived from the justification graph together with lazy grounding will likely lead to ways of efficiently handling predicates with an infinite domain by not providing an interpretation for subdomains where the predicate can be chosen freely.

This hypothesis of course needs an experimental evaluation on a well-chosen set of real world problems. We expect to publish at least one summary of the devised

methods and their evaluation to be written for publication, as well as an IDP release with full support for lazy grounding.

Furthermore, we investigate our hypothesis that the justification graph of predicates will allow us to learn characterizations of the 'behavior' of a predicate on large (infinite) parts of its domain, which can be used for lifted clause learning. The idea is that these clauses can then adequately describe the properties of the predicate, without having to compute it fully. For example, an arbitrary predicate could, by analysis of its justification graph, be detected to be an ordered list of five elements of a generic type, leading to a new representation as:

- five constants of a generic type,
- a constraint that all 5 constants have the same type a,
- a constraint that a must contain at least 5 domain elements, and provide an ordering.

Moreover, we are exploring how the concept of *oracles* can be implemented using the idea of subcalls to the same solver. To this effect, we are looking at ways to share information, data structures and other necessary information between different solver calls, with the aim of reducing overhead and setup costs. Later, we will round up our findings regarding oracles for publication.

We are also collecting a benchmark set and use this to test how eager the system must be to perform a subsolver call, as well as how detailed the learned clauses should be when a subsolver does not produce the wanted result, i.e. represents a *conflict*. Using this benchmark set, we will evaluate the different answers to the questions above and publish a paper detailing these experimental results. It will also provide us with an important indication on whether to publish an IDP version incorporating these techniques.

Lastly, we're looking at set theory as supported by the B method (Cansell and Méry 2003): The B method provides many set comprehensions and set operations. Currently, our view on predicates is that they double as sets: $P(a)$. is the same as $a \in P$. Studying this use of sets and their operations and relating them to the view where a set is exclusively defined by its 'elementOf' relation P will lead to interesting new insights in porting the capabilities of reasoning about higher order as available in B method systems (e.g. ProB) to ground-and-solve systems (such as IDP). We believe that this comparison and the derived insights will provide material for another publication.

References

Cansell, D. and Méry, D. 2003. Foundations of the B method. *Computers and Artificial Intelligence 22,* 3-4, 221–256.

Chen, W., Kifer, M., and Warren, D. S. 1993. HILOG: A foundation for higher-order logic programming. *J. Log. Program. 15,* 3, 187–230.

Cooper, D. C. 1972. Theorem proving in arithmetic without multiplication. *Machine Intelligence 7*, 91–99.

Dasseville, I., van der Hallen, M., Janssens, G., and Denecker, M. 2016. Semantics of templates in a compositional framework for building logics. *TPLP 16.* Accepted for publication.

de Cat, B., Bogaerts, B., Bruynooghe, M., and Denecker, M. 2014. Predicate logic as a modelling language: The IDP system. *CoRR abs/1401.6312.*

De Cat, B., Bogaerts, B., Denecker, M., and Devriendt, J. 2013. Model expansion in the presence of function symbols using constraint programming. In *IEEE 25th International Conference on Tools with Artificial Intelligence, ICTAI 2013, Washinton, USA, November 4-6, 2013, International Conference on Tools For Aritificial Intelligence, Washington D.C., 4-6 Nov 2013.* 1068–1075.

De Cat, B., Denecker, M., and Stuckey, P. 2012. Lazy model expansion by incremental grounding. In *Technical Communications of the 28th International Conference on Logic Programming, ICLP 2012, September 4-8, 2012, Budapest, Hungary, Internationcal Conference on Logic Programming, Budapest, 4-8 Sept 2012*, A. Dovier and V. Santos Costa, Eds. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 201–211.

De Cat, B., Denecker, M., Stuckey, P., and Bruynooghe, M. 2015. Lazy model expansion: Interleaving grounding with search. *The Journal of Artificial Intelligence Research 52*, 235–286.

Denecker, M., Brewka, G., and Strass, H. A formal theory of justifications. Accepted.

Denecker, M. and Ternovska, E. 2008. A logic of nonmonotone inductive definitions. *ACM Trans. Comput. Log. 9,* 2.

Dovier, A., Pontelli, E., and Rossi, G. 2006. Set unification. *TPLP 6,* 6, 645–701.

Kifer, M. 2005. Nonmonotonic reasoning in FLORA-2. In *Logic Programming and Nonmonotonic Reasoning, 8th International Conference, LPNMR 2005, Diamante, Italy, September 5-8, 2005, Proceedings*, C. Baral, G. Greco, N. Leone, and G. Terracina, Eds. Lecture Notes in Computer Science, vol. 3662. Springer, 1–12.

Kifer, M., Lausen, G., and Wu, J. 1995. Logical foundations of object-oriented and frame-based languages. *J. ACM 42,* 4, 741–843.

Palù, A. D., Dovier, A., Pontelli, E., and Rossi, G. 2009. GASP: answer set programming with lazy grounding. *Fundam. Inform. 96,* 3, 297–322.

Warren, D. S. 1998. Programming with tabling in XSB. In *Programming Concepts and Methods, IFIP TC2/WG2.2,2.3 International Conference on Programming Concepts and Methods (PROCOMET '98) 8-12 June 1998, Shelter Island, New York, USA*, D. Gries and W. P. de Roever, Eds. IFIP Conference Proceedings, vol. 125. Chapman & Hall, 5–6.