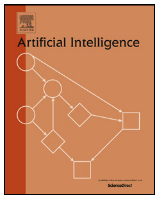
Artificial Intelligence 224 (2015) 72–102

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint



Ordered completion for logic programs with aggregates

Vernon Asuncion [a], Yin Chen [b], Yan Zhang [a], Yi Zhou [a,*]

[a] Artificial Intelligence Research Group (AIRG), School of Computing, Engineering and Mathematics, University of Western Sydney, Australia
[b] Department of Computer Science, South China Normal University, Guangzhou, China

ARTICLE INFO

Article history:
Received 7 August 2013
Received in revised form 16 March 2015
Accepted 21 March 2015
Available online 25 March 2015

Keywords:
Knowledge representation and reasoning
Answer Set Programming
Aggregates
First-order logic
Logic programming

ABSTRACT

We consider the problem of translating first-order answer set programs with aggregates into first-order sentences with the same type of aggregates. In particular, we show that, on finite structures, normal logic programs with convex aggregates, which cover both monotone and antimonotone aggregates as well as the aggregates appearing in most benchmark programs, can always be captured in first-order logic with the same type of aggregates by introducing auxiliary predicates. More precisely, we prove that every finite stable model of a normal program with convex aggregates is corresponding to a classical model of its enhanced ordered completion. This translation then suggests an alternative way for computing the stable models of such kind of programs. We report some experimental results, which demonstrate that our solver GROCv2 is comparable to the state-of-the-art answer set solvers. We further show that convex aggregates form a maximal class for this purpose. That is, we can always construct a normal logic program under any given non-convex aggregate context and prove that it can never be translated into first-order sentences with the same type of aggregates unless $NP = coNP$.

1. Introduction

In this paper, we consider to translate first-order Answer Set Programming (ASP), a predominant declarative programming paradigm in the area of knowledge representation and logic programming [3,20,24,25], into first-order logic. Work in this direction is not only of theoretical interests but also of practical relevances as it suggests an alternative way to implement ASP.

Recently, Asuncion et al. [2] proposed a notion of ordered completion (a first-order sentence with some extra predicates) for first-order normal logic programs, and showed that the stable models of a normal program are exactly corresponding to the classical models of its ordered completion on finite structures. Interestingly, there is no such translation on arbitrary structures nor prohibiting extra predicates. Based on this translation, they developed a new ASP solver, which first translates a program to its ordered completion, then grounds this first-order sentence, and finally calls an SMT solver. This is significantly different from previous ASP solvers, which ground the first-order programs directly. A first implementation shows that this new solver is promising as it performs relatively well for the Hamiltonian Circuit program, particularly on big instances [2].

However, their work cannot handle aggregates, a very important building block for modern Answer Set Programming. The reason why aggregates are crucial in answer set solving is twofold. Firstly, they enhance the expressive power of ASP, and often they can simplify the representation task. For many applications, one can write a simpler and more elegant logic

* Corresponding author.
E-mail address: y.zhou@uws.edu.au (Y. Zhou).

http://dx.doi.org/10.1016/j.artint.2015.03.007
0004-3702/© 2015 Elsevier B.V. All rights reserved.

program by using aggregates, for instance, the job scheduling program [28]. Secondly and more importantly, aggregates can improve the efficiency of ASP solving [19]. Normally, the program using aggregates can be solved much faster [12].

In this paper, we consider the problem of extending ordered completion for programs with aggregates. This is a challenging task as some programs with aggregates are expressive enough to capture disjunctive logic programming (see in [16]), thus can never be captured in first-order logic with the same type of aggregates providing some general assumptions in the computational complexity theory (see Proposition 6 in [2]).

Hence, an important task is to draw a boundary between the normal programs with aggregates that can be captured in first-order logic with the same type of aggregates and those programs that cannot. For this purpose, we extend the notion of convex constraints proposed by Liu and Truszczyński [23] into first-order convex aggregates. We show that the class of convex aggregates is exactly the boundary we need in the sense that

- First-order normal logic programs with convex aggregates can always be captured in first-order logic with the same type of aggregates on finite structures. More precisely, we extend the notion of ordered completion for first-order normal logic programs with convex aggregates, and show that every stable model of such a program is corresponding to a classical model of its enhanced ordered completion.
- Given any non-convex aggregate context, there exists a normal program under this context such that it can never be translated into first-order sentences with the same type of aggregates unless *NP* = *coNP*.

In fact, the class of convex aggregates is expressive enough to capture both monotone and antimonotone aggregates [23] as well as the aggregates appearing in most benchmark programs [5]. Therefore, based on our theoretical results, we are able to develop an alternative ASP solver for first-order normal programs with convex aggregates. Following this idea, we implement a new ASP solver GROCv2. Our experimental results demonstrate that GROCv2 is comparable to the state-of-the-art ASP solvers.

The paper is organized as follows. Section 2 reviews basic concepts and notations that we will need through out the paper. Section 3 presents the ordered completion for logic programs with aggregates, and proves the main theorems. Section 4 introduces the implementation of the ASP solver GROCv2, and reports some experimental results. Finally, Sections 5 and 6 discuss some related work and draw our conclusions respectively. We leave the very long proofs of some theorems to Appendix A for a more fluent reading.

2. Preliminaries

We consider a second-order language without functions but with equality $=$. A *signature* contains a finite set of constants and a finite set of predicates. A *term* is either a variable or a constant. A *standard atom* is an expression $P(\mathbf{t})$, where P is a predicate and $\mathbf{t}$ is a tuple of terms which matches the arity of P. An *equality atom* is an expression $t_1 = t_2$, where t_1 and t_2 are terms.

A *multiset* (also called a *bag*) is a pair $M = (M_s, M_f)$, where M_s is a set and M_f is a function, called the *multiplicity function*, from M_s to $\mathbb{N}$, i.e., the set of positive integers $\{1, 2, 3, \ldots\}$. A multiset (M_s, M_f) is finite if M_s is finite. Let M and M' be two multisets. We denote by $M \subseteq M'$ if $M_s \subseteq M'_s$ and for all elements $a \in M_s$, $M_f(a) \leq M'_f(a)$. We write $M = M'$ if $M \subseteq M'$ and $M' \subseteq M$. For convenience, a multiset M, where $M_s = \{a_1, \ldots, a_n\}$ and $M_f(a_i) = c_i$ $(1 \leq i \leq n)$, is also denoted as $\{\{\underbrace{a_1, \ldots, a_1}_{c_1}, \ldots, \underbrace{a_i, \ldots, a_i}_{c_i}, \ldots, \underbrace{a_n, \ldots, a_n}_{c_n}\}\}$. The order of the elements is irrelevant. For example, $\{\{a, a, b, c\}\}$ is the multiset M, where $M_s = \{a, b, c\}$ and $M_f(a) = 2$, $M_f(b) = M_f(c) = 1$.

2.1. *The syntax of aggregates*

Aggregate is a crucial auxiliary building block for answer set programming [12,13,16,19,22,23,28]. We first define the syntax of aggregates in the first-order case. We assume a set of aggregate symbols $\mathcal{AG}$ and a (fixed) set of comparison operators on numbers $\mathcal{CO} = \{<, \leq, =, \neq, \geq, >\}$.

Definition 1. An *aggregate atom* δ is an expression of the form

$$\textsc{op}\langle \mathbf{v} : \exists \mathbf{w} Q_1(\mathbf{y}_1) \wedge \cdots \wedge Q_s(\mathbf{y}_s) \wedge \neg R_1(\mathbf{z}_1) \wedge \cdots \wedge \neg R_t(\mathbf{z}_t)\rangle \preceq t,^{1} \quad (1)$$

where

- $\textsc{op} \in \mathcal{AG}$ is an aggregate symbol,
- $Q_i(\mathbf{y}_i)$ $(1 \leq i \leq s)$ and $R_j(\mathbf{z}_j)$ $(1 \leq j \leq t)$ are standard atoms or equality atoms. In addition,

$$Q_1(\mathbf{y}_1) \wedge \cdots \wedge Q_s(\mathbf{y}_s) \wedge \neg R_1(\mathbf{z}_1) \wedge \cdots \wedge \neg R_t(\mathbf{z}_t) \quad (2)$$

  is called the *body* of δ, denoted by $Bd(\delta)$,

[1] Here, $\mathbf{w}$ could be empty. In this case, (1) is simply written as $\textsc{op}\langle \mathbf{v} : Bd(\delta)\rangle \preceq t$.