Merging Logic Programs under Answer Set Semantics

James Delgrande\textsuperscript{1}, Torsten Schaub\textsuperscript{2,*}, Hans Tompits\textsuperscript{3}, and Stefan Woltran\textsuperscript{3}

\textsuperscript{1} Simon Fraser University, Burnaby, B.C., Canada V5A 1S6
jim@cs.sfu.ca
\textsuperscript{2} Universität Potsdam, August-Bebel-Str. 89, D–14482 Potsdam, Germany
torsten@cs.uni-potsdam.de
\textsuperscript{3} Technische Universität Wien, Favoritenstraße 9–11, A–1040 Vienna, Austria
tompits@kr.tuwien.ac.at, woltran@dbai.tuwien.ac.at

Abstract. This paper considers a semantic approach for merging logic programs under answer set semantics. Given logic programs $P_1, \ldots, P_n$, the goal is to provide characterisations of the merging of these programs. Our formal techniques are based on notions of relative distance between the underlying SE models of the logic programs. Two approaches are examined. The first informally selects those models of the programs that vary the least from the models of the other programs. The second approach informally selects those models of a program $P_0$ that are closest to the models of programs $P_1, \ldots, P_n$. $P_0$ can be thought of as analogous to a set of database integrity constraints. We examine formal properties of these operators and give encodings for computing the mergings of a multiset of logic programs within the same logic programming framework. As a by-product, we provide a complexity analysis revealing that our operators do not increase the complexity of the base formalism.

Keywords: answer set programming, belief merging, strong equivalence.

1 Introduction

Answer set programming \cite{GelfondLevesque1991} is an appealing approach for representing problems in knowledge representation and reasoning: It has a conceptually simple theoretical foundation, while at the same time it has found application in a wide range of practical problems. As well, there are now efficient and well-studied implementations. However, as is the case with any large program or body of knowledge, a logic program is not a static object in general, but rather it will evolve and be subject to change, whether as a result of correcting information in the program, adding to the information already present, or in some other fashion modifying the knowledge represented in the program.

In the past, research on the evolution of logic programs mostly focussed on updating logic programs \cite{GelfondLevesque1991,Fuhrmann1994,Fuhrmann1995,Fuhrmann1996}. In such approaches, the issue was to characterise the answer sets of a sequence $\langle P_1, \ldots, P_n \rangle$ of programs, where for $j > i$, program $P_j$ has higher priority, in some sense, over $P_i$. However, seemingly the nonmonotonic nature of extended logic programs makes the problem of belief change intrinsically harder compared to a monotonic setting, often leading to subtle effects. In previous work \cite{Fuhrmann1996},

\* Affiliated with Simon Fraser University, Canada, and Griffith University, Australia.
we addressed this challenge by defining an approach for revising logic programs under answer set semantics based on the notion of an SE model [9]. The key point of this undertaking is that SE models provide a monotonic semantic foundation of answer set programs. More specifically, SE models derive from models in the logic of here-and-there, which is intermediate between classical logic and intuitionistic logic, representing the logical underpinning of strong equivalence [10]. Indeed, the latter notion can be seen as the logic programming analogue of ordinary equivalence in classical logic, in the sense that both equivalence notions adhere to a substitution principle. With our revision approach for logic programs based on SE models we thus phrased the problem of belief revision in logic programs in terms analogous to those of revision in classical logic. Additionally, the approach possesses appealing features as it satisfies all but one of the established postulates for belief revision [11].

In this paper, we employ these techniques to address the merging of logic programs. The problem of merging multiple, potentially conflicting bodies of information arises in different contexts. For example, an agent may receive reports from differing sources of knowledge, or from sets of sensors that need to be reconciled. As well, an increasingly common phenomenon is that collections of data may need to be combined into a coherent whole. In these cases, the problem is that of combining knowledge sets that may be jointly inconsistent in order to get a consistent set of merged beliefs.

In characterising the merging of logic programs, the central idea is that the SE models of the merged program are those that are in some sense “closest” to the SE models of the programs to be merged. However, as with merging knowledge bases expressed in classical logic, there is no single preferred notion of distance nor closeness, and consequently different approaches have been defined for combining sources of information. We introduce two merging operators for logic programs under answer set semantics. Both operators take an arbitrary (multi-)set of logic programs as argument. The first operator can be regarded an instance of arbitration [12]. Basically (SE) models are selected from among the SE models of the programs to be merged; in a sense this operator is a natural extension of our belief revision operator, presented in previous work [8]. The second merging operator can be regarded as an instance of Konieczny and Pino Pérez’s merging operator [13]. Here, models of a designated program (representing information analogous to database integrity constraints) are selected that are closest to (or perhaps, informally, represent the best compromise among) the models of the programs to be merged.

2 Background

Answer Set Programming. A (generalised) logic program (GLP) over an alphabet \(\mathcal{A}\) is a finite set of rules of the form

\[
a_1; \ldots; a_m; \sim b_{m+1}; \ldots; \sim b_n \leftarrow c_{n+1}, \ldots, c_o, \sim d_{o+1}, \ldots, \sim d_p,
\]

where \(a_i, b_j, c_k, d_l \in \mathcal{A}\) are atoms, for \(1 \leq i \leq m \leq j \leq n \leq k \leq o \leq l \leq p\). Operators ‘;’ and ‘,’ express disjunctive and conjunctive connectives. A default literal is an atom \(a\) or its (default) negation \(\sim a\). A rule \(r\) as in (1) is called a fact if \(p = 1\),

\(^1\) Such programs were first considered by Lifschitz and Woo [14].