Late-Breaking Developments in the Field of Artificial Intelligence
Papers Presented at the Twenty-Seventh AAAI Conference on Artificial Intelligence

Modular Answer Set Solving

Yuliya Lierler
University of Nebraska-Omaha
ylierler@unomaha.edu

Mirosław Truszczyński
University of Kentucky
mirek@cs.uky.edu

Abstract

Modularity is essential for modeling large-scale practical applications. We propose modular logic programs as a modular version of answer set programming and study the relationship of our formalism to an earlier concept of lp-modules.

Introduction

Answer set programming (ASP) is a declarative programming formalism used with success in numerous practical applications (Brewka, Niemelä, and Truszczyński 2011). Modularity is one of the key techniques in principled software development. Yet, the research on modular ASP is at an early stage. Here we propose *modular logic programs* as a modular version of ASP and study the relationship of our formalism to lp-modules considered earlier in the literature (Oikarinen and Janhunen 2006).

Oikarinen and Janhunen (2006) were primarily concerned with the decomposition of lp-modules into sets of simpler ones. They proved that under some assumptions such decompositions are possible. Our perspective is different. We consider programs as modules and define modular programs as sets of modules. In other words, the modular structure of our modular programs is explicit. We relate here our approach to that of Oikarinen and Janhunen. The relationship shows that (i) our modules can be seen as a generalization of lp-modules, and that (ii) we can view the module theorem in (Oikarinen and Janhunen 2006) as offering conditions under which a modular program has the same semantics as the "regular" program obtained by taking the union of modules. We also mention an extension of modular logic programs, *modular lp-pl theories*, in which both logic programs and propositional theories can appear as modules. That formalism subsumes two other recent modular formalisms, the logic PC(ID) (Mariën et al. 2008) and SM(ASP) (Lierler and Truszczynski 2011).

Review: Logic Programs and Lp-modules

We focus on propositional languages — generalizations to the first order case are straightforward. We fix a *vocabulary* σ of *atoms* and define *literals* as expressions a and $\neg a$, where $a \in \sigma$. As usual, an *interpretation* of σ is a complete and consistent set of literals over σ. When talking about an interpretation we often refer to it by the set of its atoms (all negated atoms being then implicitly determined).

A *logic program* is a finite set of *rules*, that is, expressions

$$a_0 \leftarrow a_1, \ldots, a_\ell, not\ a_{\ell+1}, \ldots, not\ a_m, \quad not\ not\ a_{m+1}, \ldots, not\ not\ a_n, \tag{1}$$

where a_0 is an atom or $\bot$ and each a_i, $1 \leq i \leq n$, is an atom. If a_0 is an atom and $n = m$ then a rule (1) is *normal*. Programs consisting only of normal rules are called *normal*. For a program Π, by $\sigma(\Pi)$ we denote the vocabulary of Π, that is, the set of atoms that occur in Π. The expression a_0 is the *head* of the rule. The expression on the right hand side of the arrow is the *body*. We write $hd(\Pi)$ for the set of nonempty heads of rules in Π. We assume the reader is familiar with the definition of an *answer set* (we refer the reader to the paper by Lifschitz et al. (1999) for details).

Oikarinen and Janhunen (2006) introduced the concept of a *logic program module* (*lp-module*), a triple $\mathbb{P} = (\Pi, I, O)$, where (i) Π is a logic program[1]; (ii) I and O are sets of atoms such that $I \cap O = \emptyset$; and (iii) $hd(\Pi) \cap I = \emptyset$. Intuitively the set I defines the *input* of a module and the set O is the *output*. A set X of atoms is an *answer set* of an lp-module $\mathbb{P} = (\Pi, I, O)$ if X is an answer set of $\Pi \cup (X \cap I)$.

The focus of Oikarinen and Janhunen was to establish conditions under which an lp-module could be represented in terms of two simpler ones by means of some composition operation. We will discuss that now.

The *dependency graph* of a program Π is the directed graph G such that (i) the vertices of G are the atoms occurring in Π; (ii) for every rule (1) in Π that is not a constraint, G has an edge from a_0 to each atom in $\{a_1, \ldots a_\ell\}$ (the positive part of the body).

Let $\mathbb{P}_1 = (\Pi_1, I_1, O_1)$ and $\mathbb{P}_2 = (\Pi_2, I_2, O_2)$ be lp-modules and $C_1, \ldots, C_n$ the strongly connected components of the positive dependency graph of $\Pi_1 \cup \Pi_2$. There is a *positive recursion* between Π_1 and Π_2, if $C_i \cap O_1 \neq \emptyset$ and $C_i \cap O_2 \neq \emptyset$ for some component C_i.

[1] Oikarinen and Janhunen (2006) considered only normal programs. Our definition is therefore a direct generalization to programs allowing doubly negated atoms. Janhunen et al. (2007), allowed disjunctive programs in lp-modules. Our approach can be extended to cover that case.

The *join* of modules $\mathbb{P}_1 = (\Pi_1, I_1, O_1)$ and $\mathbb{P}_2 = (\Pi_2, I_2, O_2)$, denoted by $\mathbb{P}_1 \sqcup \mathbb{P}_2$, is defined if $O_1 \cap O_2 = \emptyset$; there is no positive recursion between Π_1 and Π_2; and

$$(\sigma(\Pi_1) \setminus (I_1 \cup O_1)) \cap \sigma(\Pi_2) = (\sigma(\Pi_2) \setminus (I_2 \cup O_2)) \cap \sigma(\Pi_1) = \emptyset.$$

If the join is defined, it is given by

$$\mathbb{P}_1 \sqcup \mathbb{P}_2 = (\Pi_1 \cup \Pi_2, (I1 \setminus O2) \cup (I_2 \setminus O_1), O1 \cup O2).$$

Theorem 1 (Module theorem) *For normal modules $\mathbb{P}_1$, $\mathbb{P}_2$ and a set X of atoms over $\sigma(\mathbb{P}_1 \sqcup \mathbb{P}_2)$, if $\mathbb{P}_1 \sqcup \mathbb{P}_2$ is defined, then X is an answer set of $\mathbb{P}_1 \sqcup \mathbb{P}_2$ iff $X_1 = X \cap \sigma(\mathbb{P}_1)$ and $X_2 = X \cap \sigma(\mathbb{P}_2)$ are answer sets of $\mathbb{P}_1$ and $\mathbb{P}_2$ respectively.*

Note that Oikarinen and Janhunen do not consider modular systems as such, i.e., collections of lp-modules and their overall semantics. In the following section we address this issue by introducing modular logic programs.

Modular Logic Programs

We start by defining *input answer sets*, the key semantic object for facilitating the communication between modules.

Definition 1 (Lierler and Truszczynski 2011) *A set X of atoms is an* input answer set *of a logic program Π if X is an answer set of the program $\Pi \cup (X \setminus hd(\Pi))$.*

As before, we regard answer sets and input answer sets, which are sets of atoms, as interpretations.

Definition 2 *A set of logic programs is a* modular logic program *or, simply, a* modular program.

Definition 3 *Given a modular program $\mathcal{P}$, a set X of atoms is an* answer set *of $\mathcal{P}$ if X is an input answer set of every program Π in $\mathcal{P}$.*

For example, let Π_1 be a program $\{a \leftarrow b, \quad c \leftarrow not\ b\}$ and Π_2 be $\{b \leftarrow not\ not\ b. \ \leftarrow c\}$. Set $\{a, b\}$ is the only answer set over the signature $\{a, b, c\}$ of a modular program $\{\Pi_1, \Pi_2\}$.

Modular logic programs are expressions to model situations where several independently developed "knowledge modules" need to be combined. Syntactically, the matter is trivial. The key question lies in the semantics. How to interpret the module "composition" operation? We treat it in the most natural fashion as the conjunction. That is, an interpretation is a model of the combined system if and only if it is a model of each individual module. However, the semantics of answer sets, if taken literally, is inappropriate. It would lead to trivial results with the composition of modules typically having no answer sets. In particular, if modules are based on disjoint signatures, the only possibility for the combined system to have an answer set would be when an empty set were the answer set of every module.

Input answer sets address this problem. They can be viewed as a more general version of answer sets. Under the restriction to the head atoms, they coincide with answer sets. If the signature contains atoms other than those appearing in the heads of rules, those atoms are treated as "input" atoms. They can be interpreted arbitrarily, their values "inform" the program. If that "informed" program has an answer set, that answer set extended by the "input" interpretation of input atoms is an input answer set of the program. Thus, atoms that do not appear as heads are no longer subject to CWA but are "open." The way we use answer sets here to define the semantics of input modular logic programs is essentially the same as that used in (Denecker et al. 2012) and (Denecker and Ternovska 2008) to handle D-modules and definitions respectively.

To relate the semantics of input answer sets and answer sets of lp-modules, for a program Π we define an *lp-module induced by* Π to be the lp-module

$$\mathbb{P}(\Pi) = (\Pi, \sigma(\Pi) \setminus hd(\Pi), hd(\Pi)).$$

Theorem 2 *A set X of atoms is an input answer set of a program Π if and only if $X \cap \sigma(\Pi)$ is an answer set of $\mathbb{P}(\Pi)$.*

Thus, input answer sets of programs can be specified in terms of answer sets of lp-modules. There is a similar connection in the opposite direction.

Theorem 3 *A set X of atoms is an answer set of an lp-module (Π, I, O) if and only if X is an input answer set of Π and $X \subseteq hd(\Pi) \cup I$.*

Theorems 2 and 3 show that programs with the semantics of input answer sets are closely related to lp-modules (Oikarinen and Janhunen 2006).

The following result establishes the relation between pairs of "joinable" lp-modules and modular logic programs.

Theorem 4 *For programs Π_1 and Π_2 such that $\mathbb{P}(\Pi_1) \sqcup \mathbb{P}(\Pi_2)$ is defined, and a set X of atoms over $\sigma(\Pi_1 \cup \Pi_2)$, X is an answer set of $\mathbb{P}(\Pi_1) \sqcup \mathbb{P}(\Pi_2)$ if and only if X is an answer set of the modular logic program $\{\Pi_1, \Pi_2\}$.*

The semantics of input answer sets makes it possible to combine logic programs and propositional theories in a *mixed* modular theory.

Definition 4 *A* modular lp-pl theory *is a set of logic programs and propositional theories. Given an lp-pl theory $\mathcal{T}$, a set X of atoms is an* answer set *of $\mathcal{T}$ if X is an input answer set of every program $\Pi \in \mathcal{T}$ and a (classical) model of every propositional theory $T \in \mathcal{T}$.*

In modular lp-pl theories one can replace all modules that are propositional theories by a single module, their union. Such simplifications are, in general not possible with modules that are programs. However, in some cases a counterpart to the module theorem for lp-modules can be proved so that simplification is possible. Lp-pl theories are closely related to the logic PC(ID) (Mariën et al. 2008). In the case when they consist of a single program and a single propositional theory they were considered by Lierler and Truszczynski (2011).

Discussions

Järvisalo et al. (2009), and Tasharrofi and Ternovska (2011) also studied the generalizations of lp-modules. In that work the main focus was to abstract lp-modules formalism away from any particular syntax or semantics. In contrast, we are interested in building simple modular systems on top of specific formalisms — ASP, or ASP and propositional logic. Extensions to cover other constraint formalisms are left for the future work.

References

Brewka, G.; Niemelä, I.; and Truszczyński, M. 2011. Answer set programming at a glance. *Communications of the ACM* 54(12):92–103.

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

Denecker, M.; Lierler, Y.; Truszczynski, M.; and Vennekens, J. 2012. A tarskian informal semantics for answer set programming. In *Technical Communications of the 28th International Conference on Logic Programming (ICLP)*, 277–289.

Janhunen, T.; Oikarinen, E.; Tompits, H.; and Woltran, S. 2007. Modularity aspects of disjunctive stable models. In *Procedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, 175–187.

Järvisalo, M.; Oikarinen, E.; Janhunen, T.; and Niemelä, I. 2009. A module-based framework for multi-language constraint modeling. In *Proceedings of the 10th International Conference on Logic Programming and Nonmonotonic Reasoning*, LPNMR '09, 155–168. Berlin, Heidelberg: Springer-Verlag.

Lierler, Y., and Truszczynski, M. 2011. Transition systems for model generators — a unifying approach. *Theory and Practice of Logic Programming, 27th Int'l. Conference on Logic Programming (ICLP'11) Special Issue* 11, issue 4-5.

Lifschitz, V.; Tang, L. R.; and Turner, H. 1999. Nested expressions in logic programs. *Annals of Mathematics and Artificial Intelligence* 25:369–389.

Mariën, M.; Wittocx, J.; Denecker, M.; and Bruynooghe, M. 2008. SAT(ID): Satisfiability of propositional logic extended with inductive definitions. In *SAT*, 211–224.

Oikarinen, E., and Janhunen, T. 2006. Modular equivalence for normal logic programs. In *17th European Conference on Artificial Intelligence(ECAI)*, 412–416.

Tasharrofi, S., and Ternovska, E. 2011. A semantic account for modularity in multi-language modelling of search problems. In *Frontiers of Combining Systems, 8th International Symposium (FroCoS)*, 259–274.