Artificial Intelligence 236 (2016) 65–89

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

On abstract modular inference systems and solvers ☆

Yuliya Lierler [a,*], Miroslaw Truszczynski [b,*]

[a] Department of Computer Science, University of Nebraska at Omaha, Omaha, NE 68182, USA
[b] Department of Computer Science, University of Kentucky, Lexington, KY 40506-0633, USA

ARTICLE INFO

Article history:
Received 5 August 2014
Received in revised form 10 March 2016
Accepted 17 March 2016
Available online 29 March 2016

Keywords:
Knowledge representation
Model-generation
Automated reasoning and inference
SAT solving
Answer set programming

ABSTRACT

Integrating diverse formalisms into modular knowledge representation systems offers increased expressivity, modeling convenience, and computational benefits. We introduce the concepts of *abstract inference modules* and *abstract modular inference systems* to study general principles behind the design and analysis of model generating programs, or *solvers*, for integrated multi-logic systems. We show how modules and modular systems give rise to *transition graphs*, which are a natural and convenient representation of solvers, an idea pioneered by the SAT community. These graphs lend themselves well to extensions that capture such important solver design features as learning. In the paper, we consider two flavors of learning for modular formalisms, local and global. We illustrate our approach by showing how it applies to answer set programming, propositional logic, multi-logic systems based on these two formalisms and, more generally, to satisfiability modulo theories.

1. Introduction

Knowledge representation and reasoning (KR&R) is concerned with developing formal languages and logics to model knowledge, and with designing and implementing corresponding automated reasoning tools. The choice of specific logics and tools depends on the type of knowledge to be represented and reasoned about. Different logics are suitable for common-sense reasoning, reasoning under incomplete information and uncertainty, for temporal and spatial reasoning, and for modeling and solving Boolean constraints, or constraints over larger, even continuous domains. In applications in areas such as distributed databases, semantic web, hybrid constraint modeling and solving, to name just a few, several of these aspects come into play. Accordingly, often diverse logics have to be accommodated together.

Modeling convenience is not the only reason why diverse logics are combined into modular hybrid KR&R systems. Another motivation is to exploit in reasoning the transparent structure that comes from modularity, computational strengths of individual logics, and synergies that arise when they are put together. An early example of a successful integration of different types of reasoning is constraint logic programming (CLP) [28,29], which exploited computational properties of different theories of constraints in a formalism centered around logic programming. About two decades later a similar idea appeared in the area of propositional satisfiability. The resulting approach, known as satisfiability modulo theories (SMT) [49,4], consists of integrating diverse constraint theories around the "core" provided by propositional satisfiability. SMT solvers are currently among the most efficient automated reasoning tools and are widely used for computer-aided software verification [10]. Another, more recent example is constraint answer set programming (CASP) [45,20,2,31,35] that integrates answer set

☆ This paper is a substantially extended version of the paper presented at PADL 2014 [38].

* Corresponding authors.
E-mail addresses: ylierler@unomaha.edu (Y. Lierler), mirek@cs.uky.edu (M. Truszczynski).

http://dx.doi.org/10.1016/j.artint.2016.03.004

programming (ASP) [42,47] with constraint modeling and solving [51]. These approaches do not impose any strong *a priori* restrictions on the constraint theories they allow. However, some types of theories are particularly heavily studied (for instance, equality with uninterpreted functions, forms of arithmetic, arrays). Finally, more focused hybrid systems that combine modules expressed in classical logic with modules given as answer set programs have also received substantial attention lately. Examples include the "multi-logics" PC(ID) [43], SM(ASP) [36] and ASP-FO [11].[1] These multi-logic modular integrations facilitate modeling but also often lead to enormous performance gains. A good example is the problem of existence of Hamiltonian cycles in graphs. Known propositional logic encodings require that counter variables be used to represent reachability. That leads to representations of large sizes. Using propositional logic to represent non-recursive constraints and logic programs to represent reachability (which is much more direct than a counter-based propositional encoding) leads to concise encodings. East and Truszczynski [13] demonstrated that the performance of SAT solvers on propositional encodings of the problem lags dramatically behind that of the solver *aspps*, designed for handling together propositional and logic program modules on hybrid representations of the problem.[2] The "computational" motivation behind modular KR&R underlies our paper.

The key computational task arising in KR&R is that of *model generation*. Model Generating programs, or *solvers*, developed in satisfiability (SAT) and ASP proved to be effective in a broad range of KR&R applications. Accordingly, model generation is of critical importance in modular multi-logic systems. Research on formalisms listed above resulted in fast solvers that demonstrate substantial gains that one can obtain from their heterogeneous nature. However, the diversity of logics considered and low-level technical details of their syntax and semantics obscure general principles that are important in the design and analysis of solvers for multi-logic systems.

In this paper, we address this problem by proposing a language for representing modular multi-logic systems that aims to provide a general abstract view on solvers, to bring up key principles behind solver design, and to facilitate studies of their properties. As we are not concerned with the modeling aspect of a KR&R system but with solving, we design our language so that it (i) abstracts away the syntactic details, (ii) can capture diverse concepts of inference, and (iii) is based only on the weakest assumptions concerning the semantics of underlying logics, in particular, this language can capture any formalism whose semantics is determined by a set of models. The basic elements of this language are *abstract inference modules* (or just *modules*) that are defined to consist of *inferences*. Collections of abstract inference modules constitute *abstract modular inference systems* (or just *modular systems*). We define the semantics of abstract inference modules and show that they provide a uniform language to capture inference mechanisms from different logics, and their modular combinations. Importantly, abstract inference modules and abstract modular inference systems give rise to *transition graphs* of the type introduced by Nieuwenhuis, Oliveras, and Tinelli [49] in their study of SAT and SMT solvers. As in that earlier work, our transition graphs provide a natural and convenient representation of solvers for modules and modular systems. They lend themselves well to extensions that capture such important solver design techniques as learning (which here comes in two flavors: *local* that is limited to single modules, and *global* that is applied across modules). In this way, abstract modular inference systems and the corresponding framework of transition graphs are useful conceptualizations clarifying computational principles behind solvers for multi-logic knowledge representation systems and facilitating systematic development of new ones. The design of transition systems based on *syntax-free* modules is what separates this work from earlier uses of graphs for describing model generation algorithms behind SAT, SMT, PC(ID), or ASP solvers [49,43,34,36]. These earlier transition graphs are language specific and based on the syntactic constructs typical of the respective formalisms. Adding a new level of abstraction allows one a bird's eye view on the landscape of solving techniques and their usage in hybrid settings.

To demonstrate the power of our approach, we show that it applies to answer set programming, propositional logic, multi-logic systems based on these two formalisms, and generally to satisfiability modulo theories. As SMT is a general framework for integrating diverse logics, the same expressivity claims hold true for our approach. However, in at least one aspect, our approach goes beyond the basic tenants of SMT. Namely, our modular systems have no central core, the role played by SAT in the case of SMT. Rather, all modules are viewed in exactly the same way and can pass on results of inferences directly to each other. In addition, all modules are presented in a uniform way as sets of inferences. In this way, we can ignore syntactical aspects of logics. Of course, that makes our formalism poorly tailored for modeling, as it is the syntax of logics that is typically used to provide concise representations of knowledge. Yet, our syntax-free modules make explicit the reasoning the logics of the modules support, and that is of central importance to our objective to support the design and analysis of solvers.

The paper is organized as follows. We start by introducing abstract inference modules. We then adapt the transition graphs of Nieuwenhuis et al. [49] to the formalism of abstract inference modules and use them to describe algorithms for finding and enumerating models of modules. In Section 4, we introduce abstract modular inference systems, extend the concept of a transition graph to modular systems, and show that transition graphs can be used to formalize search

[1] Logic PC(ID) is a propositional fragment of classical first-order logic with inductive definitions; SM(ASP) is a propositional language that merges classical logic expressions and logic programs under stable model semantics; ASP(FO) is a first-order language, which encapsulates modules stemming from classical logic and modules stemming from logic programming.

[2] We stress that in this discussion we simply aim at showing that combining modules coming from different logics may be beneficial. Here, limitations of propositional logic in modeling recursive constraints are overcome with modules designed specifically for this task. However, it is possible (in fact, straightforward) to design a single ASP program that efficiently handles the Hamiltonian cycle application.