Modularity of P-Log Programs

Carlos Viegas Damásio and João Moura
CENTRIA, Departamento de Informática
Universidade Nova de Lisboa, 2829-516 Caparica, Portugal

Abstract. We propose an approach for modularizing P-log programs and corresponding compositional semantics based on conditional probability measures. We do so by resorting to Oikarinen and Janhunen’s definition of a logic program module and extending it to P-log by introducing the notions of input random attributes and output literals. For answering to P-log queries our method does not imply calculating all the stable models (possible worlds) of a given program, and previous calculations can be reused. Our proposal also handles probabilistic evidence by conditioning (observations).

Keywords: P-log, Answer Set Programming, Modularization, Probabilistic Reasoning.

1 Introduction and Motivation

The P-log language \cite{3} has emerged as one of the most flexible frameworks for combining probabilistic reasoning with logical reasoning, in particular, by distinguishing acting (doing) from observations and allowing non-trivial conditioning forms \cite{3,4}. The P-log languages is a non-monotonic probabilistic logic language supported by two major formalisms, namely Answer Set Programming \cite{7,11,12} for declarative knowledge representation and Causal Bayesian Networks \cite{15} as its probabilistic foundation. In particular, ordinary Bayesian Networks can be encoded in P-log. The relationships of P-log to other alternative uncertainty knowledge representation languages like \cite{10,16,17} have been studied in \cite{3}. Unfortunately, the existing current implementations of P-log \cite{18} have exponential best case complexity, since they enumerate all possible models, even though it is known that for singly connected Bayesian Networks (polytrees) reasoning can be performed in polynomial time \cite{14}.

The contribution of this paper is the definition of modules for the P-log language, and corresponding compositional semantics as well as its probabilistic interpretation. The semantics relies on a translation to logic program modules of Oikarinen and Janhunen \cite{13}. With this appropriate notion of P-log modules is possible to obtain possible worlds incrementally, and this can be optimized for answering to probabilistic queries in polynomial time for specific cases, using techniques inspired in the variable elimination algorithm \cite{20}.

The rest of the paper is organized as follows. Section \ref{sec:related} briefly summarizes P-log syntax and semantics, as well as the essential modularity results for answer
set programming. Next, Section 3 is the core of the paper defining modules for P-log language and its translation into ASP modules. The subsequent section presents the module theorem and a discussion of the application of the result to Bayesian Networks. We conclude with final remarks and foreseen work.

2 Preliminaries

In this section, we review the syntax and semantics of P-log language [3], and illustrate it with an example encoding a Bayesian Network. Subsequently, the major results regarding composition of answer set semantics are presented [13]. The reader is assumed to have familiarity with (Causal) Bayesian Networks [15] and good knowledge of answer set programming. A good introduction to Bayesian Networks can be found in [19], and to answer set programming in [2,12].

2.1 P-Log Programs

P-log is a declarative language [3], based on a logic formalism for probabilistic reasoning and action, that uses answer set programming (ASP) as its logical foundation and Causal Bayesian Networks (CBNs) as its probabilistic foundation. P-log is a complex language to present in a short amount of space, and the reader is referred to [3] for full details. We will try to make the presentation self-contained for this paper, abbreviating or even neglecting the irrelevant parts, and follows closely [3].

P-log syntax. A probabilistic logic program (P-log program) \( \Pi \) consists of (i) a sorted signature, (ii) a declaration part, (iii) a regular part, (iv) a set of random selection rules, (v) a probabilistic information part, and (vi) a set of observations and actions. Notice that the first four parts correspond to the actual stable models’ generation, and the last two define the probabilistic information.

The declaration part defines a sort \( c \) by explicitly listing its members with a statement \( c = \{ x_1, \ldots, x_n \} \), or by defining a unary predicate \( c \) in a program with a single answer set. An attribute \( a \) with \( n \) parameters is declared by a statement \( a : c_1 \times \ldots \times c_n \rightarrow c_0 \) where each \( c_i \) is a sort \( (0 \leq i \leq m) \); in the case of an attributes with no parameter the syntax \( a : c_0 \) may be used. By \( \text{range}(a) \) we denote the set of elements of sort \( c_0 \). The sorts can be understood as domain declarations for predicates and attributes used in the program, for appropriate typing of argument variables.

The regular part of a P-log program is just a set of Answer Set Programming rules (without disjunction) constructed from the usual literals in answer set programming plus attribute literals of the form \( a(\overline{t}) = t_0 \) (including strongly negated literals), where \( \overline{t} \) is a vector of \( n \) terms and \( t_0 \) is a term, respecting the corresponding sorts in the attribute declaration. Given a sorted signature \( \Sigma \) we denote by \( \text{Lit}(\Sigma) \) the set of literals in \( \Sigma \) (i.e. \( \Sigma \)-literals) excluding all unary atoms \( c_i/1 \) used for specifying sorts.

Random selection rules define random attributes and possible values for them through statements of the form \( \{ r \} \ random \ a(\overline{t}) : \{ X : p(X) \} \leftarrow B \), expressing