# Stratified Probabilistic Description Logic Programs

Thomas Lukasiewicz*

Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza"
Via Salaria 113, I-00198 Rome, Italy
lukasiewicz@dis.uniroma1.it

**Abstract.** In previous work, we have introduced probabilistic description logic programs (or pdl-programs), which are a combination of description logic programs (or dl-programs) under the answer set and well-founded semantics with Poole's independent choice logic. Such programs are directed towards sophisticated representation and reasoning techniques that allow for probabilistic uncertainty in the Rules, Logic, and Proof layers of the Semantic Web. In this paper, we continue this line of research. We concentrate on the special case of stratified probabilistic description logic programs (or spdl-programs). In particular, we present an algorithm for query processing in such pdl-programs, which is based on a reduction to computing the canonical model of stratified dl-programs.

## 1 Introduction

The *Semantic Web* initiative [2,9] aims at an extension of the current World Wide Web by standards and technologies that help machines to understand the information on the Web so that they can support richer discovery, data integration, navigation, and automation of tasks. The main ideas behind it are to add a machine-readable meaning to Web pages, to use ontologies for a precise definition of shared terms in Web resources, to make use of KR technology for automated reasoning from Web resources, and to apply cooperative agent technology for processing the information of the Web.

The Semantic Web consists of several hierarchical layers, where the *Ontology layer*, in form of the *OWL Web Ontology Language* [30,18] (recommended by the W3C), is currently the highest layer of sufficient maturity. OWL consists of three increasingly expressive sublanguages, namely *OWL Lite*, *OWL DL*, and *OWL Full*. OWL Lite and OWL DL are essentially very expressive description logics with an RDF syntax [18]. As shown in [16], ontology entailment in OWL Lite (resp., OWL DL) reduces to knowledge base (un)satisfiability in the description logic $\mathcal{SHIF}(\mathbf{D})$ (resp., $\mathcal{SHOIN}(\mathbf{D})$). On top of the Ontology layer, the *Rules*, *Logic*, and *Proof layers* of the Semantic Web will be developed next, which should offer sophisticated representation and reasoning capabilities. As a first effort in this direction, *RuleML* (Rule Markup Language) [3] is an XML-based markup language for rules and rule-based systems, whereas the OWL Rules Language [17] is a first proposal for extending OWL by Horn clause rules.

A key requirement of the layered architecture of the Semantic Web is to integrate the Rules and the Ontology layer. In particular, it is crucial to allow for building rules on top

* Alternate address: Institut für Informationssysteme, Technische Universität Wien, Favoritenstraße 9-11, A-1040 Vienna, Austria; e-mail: lukasiewicz@kr.tuwien.ac.at.

of ontologies, that is, for rule-based systems that use vocabulary from ontology knowledge bases. Another type of combination is to build ontologies on top of rules, which means that ontological definitions are supplemented by rules or imported from rules. Towards this goal, the works [7,8] have proposed *description logic programs* (or *dl-programs*), which are of the form $KB = (L, P)$, where $L$ is a knowledge base in a description logic and $P$ is a finite set of description logic rules (or *dl-rules*). Such dl-rules are similar to usual rules in logic programs with negation as failure, but may also contain *queries to* $L$ in their bodies, which are given by special atoms (on which possibly default negation may apply). Another important feature of dl-rules is that queries to $L$ also allow for specifying an input from $P$, and thus for a *flow of information from* $P$ *to* $L$, besides the flow of information from $L$ to $P$, given by any query to $L$. Hence, description logic programs allow for building rules on top of ontologies, but also (to some extent) building ontologies on top of rules. In this way, additional knowledge (gained in the program) can be supplied to $L$ before querying. The semantics of dl-programs was defined in [7] and [8] as an extension of the answer set semantics by Gelfond and Lifschitz [12] and the well-founded semantics by Van Gelder, Ross, and Schlipf [29], respectively, which are the two most widely used semantics for nonmonotonic logic programs. The description logic knowledge bases in dl-programs are specified in the well-known description logics $\mathcal{SHIF}(\mathbf{D})$ and $\mathcal{SHOIN}(\mathbf{D})$.

In [22], towards sophisticated representation and reasoning techniques that also allow for modeling probabilistic uncertainty in the Rules, Logic, and Proof layers of the Semantic Web, we have presented *probabilistic description logic programs* (or *pdl-programs*), which generalize dl-programs under the answer set and well-founded semantics by probabilistic uncertainty. They have been developed as a combination of dl-programs with Poole's independent choice logic (ICL) [25].

It is important to point out that Poole's ICL is a powerful representation and reasoning formalism for single- and also multi-agent systems, which combines logic and probability, and which can represent a number of important uncertainty formalisms, in particular, influence diagrams, Bayesian networks, Markov decision processes, and normal form games [25]. Furthermore, Poole's ICL also allows for natural notions of causes and explanations as in Pearl's structural causal models [10].

In this paper, we continue this line of research. We concentrate on the special case of stratified pdl-programs (or *spdl-programs*). In particular, as a main new contribution, we present an algorithm for query processing in spdl-programs. It is based on a reduction to computing the canonical model of stratified dl-programs, which can be done by a finite sequence of finite fixpoint iterations. This shows especially that query processing in spdl-programs is conceptually easier than query processing in general pdl-programs, which is reducible to computing the set of all answer sets of general dl-programs and solving linear optimization problems. To my knowledge, this paper and [22] are the first works that combine description logic programs with probabilistic uncertainty.

The rest of this paper is organized as follows. In Section 2, we recall the description logics $\mathcal{SHIF}(\mathbf{D})$ and $\mathcal{SHOIN}(\mathbf{D})$ as well as stratified description logic programs. Section 3 defines stratified probabilistic description logic programs, and Section 4 deals with query processing in such programs. In Section 5, we discuss related work. Section 6 summarizes the main results and gives an outlook on future research.

## 2 Preliminaries

In this section, we first recall the description logics $\mathcal{SHIF}(\mathbf{D})$ and $\mathcal{SHOIN}(\mathbf{D})$. We then recall positive and stratified *description logic programs* (or *dl-programs*) under their canonical semantics [7], which combine description logics and normal programs. They consist of a knowledge base $L$ in a description logic and a finite set of description logic rules $P$. Such rules are similar to usual rules in logic programs with negation as failure, but may also contain *queries to* $L$, possibly default negated.

### 2.1 $\mathcal{SHIF}(\mathbf{D})$ and $\mathcal{SHOIN}(\mathbf{D})$

We first describe $\mathcal{SHOIN}(\mathbf{D})$. We assume a set $\mathbf{D}$ of *elementary datatypes*. Each $d \in \mathbf{D}$ has a set of *data values*, called the *domain* of $d$, denoted $\mathrm{dom}(d)$. Let $\mathrm{dom}(\mathbf{D}) = \bigcup_{d \in \mathbf{D}} \mathrm{dom}(d)$. A *datatype* is either an element of $\mathbf{D}$ or a subset of $\mathrm{dom}(\mathbf{D})$ (called *datatype oneOf*). Let $\mathbf{A}$, $\mathbf{R}_A$, $\mathbf{R}_D$, and $\mathbf{I}$ be nonempty finite pairwise disjoint sets of *atomic concepts*, *abstract roles*, *datatype roles*, and *individuals*, respectively. Let $\mathbf{R}_A^-$ denote the set of all inverses $R^-$ of abstract roles $R \in \mathbf{R}_A$.

A *role* is an element of $\mathbf{R}_A \cup \mathbf{R}_A^- \cup \mathbf{R}_D$. *Concepts* are inductively defined as follows. Every $C \in \mathbf{A}$ is a concept, and if $o_1, o_2, \ldots \in \mathbf{I}$, then $\{o_1, o_2, \ldots\}$ is a concept (called *oneOf*). If $C$ and $D$ are concepts and if $R \in \mathbf{R}_A \cup \mathbf{R}_A^-$, then $(C \sqcap D)$, $(C \sqcup D)$, and $\neg C$ are concepts (called *conjunction*, *disjunction*, and *negation*, respectively), as well as $\exists R.C$, $\forall R.C$, $\geq nR$, and $\leq nR$ (called *exists*, *value*, *atleast*, and *atmost restriction*, respectively) for an integer $n \geq 0$. If $d \in \mathbf{D}$ and $U \in \mathbf{R}_D$, then $\exists U.d$, $\forall U.d$, $\geq nU$, and $\leq nU$ are concepts (called *datatype exists*, *value*, *atleast*, and *atmost restriction*, respectively) for an integer $n \geq 0$. We write $\top$ and $\bot$ to abbreviate $C \sqcup \neg C$ and $C \sqcap \neg C$, respectively, and we eliminate parentheses as usual.

An *axiom* is of one of the following forms: (1) $C \sqsubseteq D$, where $C$ and $D$ are concepts (*concept inclusion*); (2) $R \sqsubseteq S$, where either $R, S \in \mathbf{R}_A$ or $R, S \in \mathbf{R}_D$ (*role inclusion*); (3) $\mathrm{Trans}(R)$, where $R \in \mathbf{R}_A$ (*transitivity*); (4) $C(a)$, where $C$ is a concept and $a \in \mathbf{I}$ (*concept membership*); (5) $R(a, b)$ (resp., $U(a, v)$), where $R \in \mathbf{R}_A$ (resp., $U \in \mathbf{R}_D$) and $a, b \in \mathbf{I}$ (resp., $a \in \mathbf{I}$ and $v \in \mathrm{dom}(\mathbf{D})$) (*role membership*); and (6) $a = b$ (resp., $a \neq b$), where $a, b \in \mathbf{I}$ (*equality* (resp., *inequality*)). A *knowledge base* $L$ is a finite set of axioms. For decidability, number restrictions in $L$ are restricted to simple $R \in \mathbf{R}_A$ [19].

The syntax of $\mathcal{SHIF}(\mathbf{D})$ is as the above syntax of $\mathcal{SHOIN}(\mathbf{D})$, but without the oneOf constructor and with the *atleast* and *atmost* constructors limited to 0 and 1.

For the semantics of $\mathcal{SHIF}(\mathbf{D})$ and $\mathcal{SHOIN}(\mathbf{D})$, we refer the reader to [16].

*Example 2.1.* An online store (such as *amazon.com*) may use a description logic knowledge base to classify and characterize its products. For example, suppose that (1) textbooks are books, (2) personal computers and cameras are electronic products, (3) books and electronic products are products, (4) every product has at least one related product, (5) only products are related to each other, (6) $tb\_ai$ and $tb\_lp$ are textbooks, which are related to each other, (7) $pc\_ibm$ and $pc\_hp$ are personal computers, which are related to each other, and (8) $ibm$ and $hp$ are providers for $pc\_ibm$ and $pc\_hp$, respectively. This knowledge is expressed by the following description logic knowledge base $L_1$:

$$Textbook \sqsubseteq Book;\ \ PC \sqcup Camera \sqsubseteq Electronics;\ \ Book \sqcup Electronics \sqsubseteq Product;$$

$Product \sqsubseteq \geq 1\, related$; $\geq 1\, related \sqcup \geq 1\, related^- \sqsubseteq Product$;
$Textbook(tb\_ai)$; $Textbook(tb\_lp)$; $PC(pc\_ibm)$; $PC(pc\_hp)$;
$related(tb\_ai, tb\_lp)$; $related(pc\_ibm, pc\_hp)$;
$provides(ibm, pc\_ibm)$; $provides(hp, pc\_hp)$.

### 2.2 Syntax of Description Logic Programs

We assume a function-free first-order vocabulary $\Phi$ with nonempty finite sets of constant and predicate symbols, and a set $\mathcal{X}$ of variables. A *term* is a constant symbol from $\Phi$ or a variable from $\mathcal{X}$. If $p$ is a predicate symbol of arity $k \geq 0$ from $\Phi$ and $t_1, \ldots, t_k$ are terms, then $p(t_1, \ldots, t_k)$ is an *atom*. A *negation-as-failure* (*NAF*) *literal* is an atom $a$ or a default-negated atom $not\, a$. A *normal rule* $r$ is of the form

$$a \leftarrow b_1, \ldots, b_k, not\, b_{k+1}, \ldots, not\, b_m\ ,\ m \geq k \geq 0\,, \tag{1}$$

where $a, b_1, \ldots, b_m$ are atoms. We refer to $a$ as the *head* of $r$, denoted $H(r)$, while the conjunction $b_1, \ldots, b_k, not\, b_{k+1}, \ldots, not\, b_m$ is the *body* of $r$; its *positive* (resp., *negative*) part is $b_1, \ldots, b_k$ (resp., $not\, b_{k+1}, \ldots, not\, b_m$). We define $B(r) = B^+(r) \cup B^-(r)$, where $B^+(r) = \{b_1, \ldots, b_k\}$ and $B^-(r) = \{b_{k+1}, \ldots, b_m\}$. A *normal program* $P$ is a finite set of normal rules. Informally, a dl-program consists of a description logic knowledge base $L$ and a generalized normal program $P$, which may contain queries to $L$. In such a query, it is asked whether a certain description logic axiom or its negation logically follows from $L$ or not. Formally, a *dl-query* $Q(\mathbf{t})$ is either

(a) a concept inclusion axiom $F$ or its negation $\neg F$; or
(b) of the forms $C(t)$ or $\neg C(t)$, where $C$ is a concept and $t$ is a term; or
(c) of the forms $R(t_1, t_2)$ or $\neg R(t_1, t_2)$, where $R$ is a role and $t_1$, $t_2$ are terms.

A *dl-atom* has the form $DL[S_1 op_1 p_1, \ldots, S_m op_m\ p_m; Q](\mathbf{t})$, where each $S_i$ is a concept or role, $op_i \in \{\uplus, \cup\}$, $p_i$ is a unary resp. binary predicate symbol, $Q(\mathbf{t})$ is a dl-query, and $m \geq 0$. We call $p_1, \ldots, p_m$ its *input predicate symbols*. Intuitively, $op_i = \uplus$ (resp., $op_i = \cup$) increases $S_i$ (resp., $\neg S_i$) by the extension of $p_i$. A *dl-rule* $r$ is of form (1), where any $b \in B(r)$ may be a dl-atom. A *dl-program* $KB = (L, P)$ consists of a description logic knowledge base $L$ and a finite set of dl-rules $P$. *Ground terms*, *atoms*, *literals*, etc., are defined as usual. The *Herbrand base* of $P$, denoted $HB_P$, is the set of all ground atoms with standard predicate symbols in $P$ and constant symbols in $\Phi$. Let $ground(P)$ be the set of all ground instances of dl-rules in $P$ w.r.t. $HB_P$.

*Example 2.2.* Consider the dl-program $KB_1 = (L_1, P_1)$, where $L_1$ is the description logic knowledge base from Example 2.1, and $P_1$ is the following set of dl-rules:

(1) $pc(pc\_1)$; $pc(pc\_2)$; $pc(pc\_3)$;
(2) $brand\_new(pc\_1)$; $brand\_new(pc\_2)$;
(3) $vendor(dell, pc\_1)$; $vendor(dell, pc\_2)$; $vendor(dell, pc\_3)$;
(4) $avoid(X) \leftarrow DL[Camera](X), not\ offer(X)$;
(5) $offer(X) \leftarrow DL[PC \uplus pc; Electronics](X), not\ brand\_new(X)$;

(6) $provider(P) \leftarrow vendor(P,X), DL[PC \uplus pc; Product](X)$;
(7) $provider(P) \leftarrow DL[provides](P,X), DL[PC \uplus pc; Product](X)$;
(8) $similar(X,Y) \leftarrow DL[related](X,Y)$;
(9) $similar(X,Z) \leftarrow similar(X,Y), similar(Y,Z)$.

The above dl-rules express that (1) $pc\_1$, $pc\_2$, and $pc\_3$ are additional personal computers, (2) $pc\_1$ and $pc\_2$ are brand new, (3) $dell$ is the vendor of $pc\_1$, $pc\_2$, and $pc\_3$, (4) a customer avoids all cameras that are not on offer, (5) all electronic products that are not brand new are on offer, (6) every vendor of a product is a provider, (7) every entity providing a product is a provider, (8) all related products are similar, and (9) the binary similarity relation on products is transitively closed.

### 2.3 Semantics of Positive Description Logic Programs

In the sequel, let $KB{=}(L,P)$ be a dl-program. An *interpretation* $I$ relative to $P$ is any $I \subseteq HB_P$. We say that $I$ is a *model* of $a \in HB_P$ under $L$, denoted $I \models_L a$, iff $a \in I$. We say that $I$ is a *model* of a ground dl-atom $a = DL[S_1 op_1\ p_1, \ldots, S_m op_m p_m; Q](\mathbf{c})$ under $L$, denoted $I \models_L a$, iff $L \cup \bigcup_{i=1}^{m} A_i(I) \models Q(\mathbf{c})$, where $A_i(I) = \{S_i(\mathbf{e}) \mid p_i(\mathbf{e}){\in}I\}$, for $op_i = \uplus$; and $A_i(I) = \{\neg S_i(\mathbf{e}) \mid p_i(\mathbf{e}){\in}I\}$, for $op_i = \cup$. A ground dl-atom $a$ is *monotonic* relative to $KB = (L,P)$ iff $I \subseteq I' \subseteq HB_P$ implies that if $I \models_L a$ then $I' \models_L a$. In this paper, we consider only monotonic ground dl-atoms, but observe that one can also define dl-atoms that are not monotonic; see [7]. We say that $I$ is a *model* of a ground dl-rule $r$ iff $I \models_L H(r)$ whenever $I \models_L B(r)$, that is, $I \models_L a$ for all $a \in B^+(r)$ and $I \not\models_L a$ for all $a \in B^-(r)$. We say that $I$ is a *model* of a dl-program $KB = (L,P)$, denoted $I \models KB$, iff $I \models_L r$ for every $r \in ground(P)$. We say that $KB$ is *satisfiable* (resp., *unsatisfiable*) iff it has some (resp., no) model.

We say that $KB{=}(L,P)$ is *positive* iff no dl-rule in $P$ contains default-negated atoms. Like ordinary positive programs, every positive dl-program $KB$ is satisfiable and has a unique least model, denoted $M_{KB}$, that canonically characterizes its semantics.

### 2.4 Semantics of Stratified Description Logic Programs

We next define stratified dl-programs and their canonical semantics. They are intuitively composed of hierarchic layers of positive dl-programs linked via default negation. Like ordinary stratified normal programs, they are always satisfiable and can be assigned a canonical minimal model via a number of iterative least models.

For any dl-program $KB = (L,P)$, we denote by $DL_P$ the set of all ground dl-atoms that occur in $ground(P)$. An *input atom* of $a \in DL_P$ is a ground atom with an input predicate of $a$ and constant symbols in $\Phi$. A *(local) stratification of* $KB = (L,P)$ is a mapping $\lambda\colon HB_P \cup DL_P \rightarrow \{0,1,\ldots,k\}$ such that (i) $\lambda(H(r)) \geq \lambda(b')$ (resp., $\lambda(H(r)) > \lambda(b')$) for each $r \in ground(P)$ and $b' \in B^+(r)$ (resp., $b' \in B^-(r)$), and (ii) $\lambda(a) \geq \lambda(b)$ for each input atom $b$ of each $a \in DL_P$, where $k \geq 0$ is the *length* of $\lambda$. For $i \in \{0,\ldots,k\}$, let $KB_i = (L,P_i) = (L, \{r \in ground(P) \mid \lambda(H(r)) = i\})$, and let $HB_{P_i}$ (resp., $HB^{\star}_{P_i}$) be the set of all $b \in HB_P$ such that $\lambda(b) = i$ (resp., $\lambda(b) \leq i$). A dl-program $KB = (L,P)$ is *(locally) stratified* iff it has a stratification $\lambda$ of some length $k \geq 0$. We define its iterative least models $M_i \subseteq HB_P$ with $i \in \{0,\ldots,k\}$ as follows:

(i) $M_0$ is the least model of $KB_0$;
(ii) if $i > 0$, then $M_i$ is the least model of $KB_i$ such that $M_i|HB^{\star}_{P_{i-1}} = M_{i-1}|HB^{\star}_{P_{i-1}}$.

The canonical model of the stratified dl-program $KB$, denoted $M_{KB}$, is then defined as $M_k$. Observe that $M_{KB}$ is well-defined, since it does not depend on a particular $\lambda$. Furthermore, $M_{KB}$ is in fact a minimal model of $KB$.

## 3 Stratified Probabilistic Description Logic Programs

In this section, we define stratified probabilistic dl-programs as a combination of dl-programs with Poole's independent choice logic (ICL) [25]. Poole's ICL is based on ordinary acyclic logic programs under different "atomic choices", where each atomic choice along with an acyclic logic program produces a first-order model, and one then obtains a probability distribution over the set of first-order models by placing a distribution over the different atomic choices. In stratified probabilistic dl-programs, we here use stratified dl-programs rather than ordinary acyclic logic programs.

### 3.1 Syntax

We assume a function-free first-order vocabulary $\Phi$ with nonempty finite sets of constant and predicate symbols, and a set of variables $\mathcal{X}$, as in Section 2. We use $HB_\Phi$ (resp., $HU_\Phi$) to denote the Herbrand base (resp., universe) over $\Phi$. In the sequel, we assume that $HB_\Phi$ is nonempty. We define *classical formulas* by induction as follows. The propositional constants *false* and *true*, denoted $\bot$ and $\top$, respectively, and all atoms are classical formulas. If $\phi$ and $\psi$ are classical formulas, then also $\neg\phi$ and $(\phi \wedge \psi)$. A *conditional constraint* is of the form $(\psi|\phi)[l,u]$ with reals $l, u \in [0,1]$ and classical formulas $\phi$ and $\psi$. We define *probabilistic formulas* inductively as follows. Every conditional constraint is a probabilistic formula. If $F$ and $G$ are probabilistic formulas, then also $\neg F$ and $(F \wedge G)$. We use $(F \vee G)$, $(F \Leftarrow G)$, and $(F \Leftrightarrow G)$ to abbreviate $\neg(\neg F \wedge \neg G)$, $\neg(\neg F \wedge G)$, and $(\neg(\neg F \wedge G) \wedge \neg(F \wedge \neg G))$, respectively, and adopt the usual conventions to eliminate parentheses. *Ground terms*, *ground formulas*, *substitutions*, and *ground instances* of probabilistic formulas are defined as usual.

A *choice space* $C$ is a set of pairwise disjoint and nonempty sets $A \subseteq HB_\Phi$. Any member $A \in C$ is called an *alternative* of $C$ and any element $a \in A$ an *atomic choice* of $C$. A *total choice* of $C$ is a set $B \subseteq HB_\Phi$ such that $|B \cap A| = 1$ for all $A \in C$. A *probability* $\mu$ on a choice space $C$ is a probability function on the set of all total choices of $C$. Since $C$ and all its alternatives are finite, $\mu$ can be defined by (i) a mapping $\mu\colon \bigcup C \rightarrow [0,1]$ such that $\sum_{a \in A} \mu(a) = 1$ for all $A \in C$, and (ii) $\mu(B) = \Pi_{b \in B} \mu(b)$ for all total choices $B$ of $C$. Intuitively, (i) associates a probability with each atomic choice of $C$, and (ii) assumes independence between the alternatives of $C$.

A *probabilistic dl-program* (or *pdl-program*) $KB = (L, P, C, \mu)$ consists of a dl-program $(L, P)$, a choice space $C$ such that (i) $\bigcup C \subseteq HB_P$ and (ii) no atomic choice in $C$ coincides with the head of any dl-rule in $ground(P)$, and a probability $\mu$ on $C$. A *stratified probabilistic dl-program* (or *spdl-program*) is a pdl-program $KB = (L, P, C, \mu)$ where $(L, P)$ is stratified. A *probabilistic query* to $KB$ has the form $?F$ or the form

$?(\beta|\alpha)[L,U]$, where $F$ is a probabilistic formula, $\beta, \alpha$ are classical formulas, and $L, U$ are variables. The *correct answer* to $?F$ is the set of all substitutions $\theta$ such that $F\theta$ is a consequence of $KB$. The *tight answer* to $?(\beta|\alpha)[L,U]$ is the set of all substitutions $\theta$ such that $?(\beta|\alpha)[L,U]\theta$ is a tight consequence of $KB$. In the following paragraphs, we define the notions of *consequence* and *tight consequence*.

*Example 3.1.* Consider the spdl-program $KB_1 = (L_1, P_1, C_1, \mu_1)$, where $L_1$ is as in Example 2.1, and $P_1$ is as in Example 2.2 except that the dl-rules (4) and (5) are replaced by the dl-rules (4') and (5'), respectively, and the dl-rules (10) and (11) are added:

(4') $avoid(X) \leftarrow DL[Camera](X), not\ offer(X), avoid\_pos$;
(5') $offer(X) \leftarrow DL[PC \uplus pc; Electronics](X), not\ brand\_new(X), offer\_pos$;
(10) $buy(C,X) \leftarrow needs(C,X), view(X), not avoid(X), v\_buy\_pos$;
(11) $buy(C,X) \leftarrow needs(C,X), buy(C,Y), also\_buy(Y,X), a\_buy\_pos$.

Furthermore, let $C_1$ be given by $\{\{avoid\_pos, avoid\_neg\}, \{offer\_pos, offer\_neg\}, \{v\_buy\_pos,\ v\_buy\_neg\}, \{a\_buy\_pos,\ a\_buy\_neg\}\}$, and let $\mu_1(avoid\_pos) = 0.9$, $\mu_1(avoid\_neg) = 0.1$, $\mu_1(offer\_pos) = 0.9$, $\mu_1(offer\_neg) = 0.1$, $\mu_1(v\_buy\_pos) = 0.7$, $\mu_1(v\_buy\_neg) = 0.3$, $\mu_1(a\_buy\_pos) = 0.7$, and $\mu_1(a\_buy\_neg) = 0.3$.

Here, the new dl-rules (4') and (5') express that the dl-rules (4) and (5) actually only hold with the probability $0.9$. Furthermore, (10) expresses that a customer buys a needed product that is viewed and not avoided with the probability $0.7$, while (11) says that a customer buys a needed product $x$ with probability $0.7$, if she bought another product $y$, and every customer that previously had bought $y$ also bought $x$.

In a probabilistic query, one may ask for the tight probability bounds that a customer $c$ buys a needed product $x$, if (i) $c$ bought another product $y$, (ii) every customer that previously had bought $y$ also bought $x$, (iii) $x$ is not avoided, and (iv) $c$ has been shown product $x$ (the result to this query may, e.g., help to decide whether it is useful to make a customer automatically also view product $x$ when buying $y$):

$$?(buy(c,x) \mid needs(c,x) \wedge buy(c,y) \wedge also\_buy(y,x) \wedge view(x) \wedge not\, avoid(x))[L,U]\,.$$

### 3.2 Semantics

A *world* $I$ is a subset of $HB_\Phi$. We use $\mathcal{I}_\Phi$ to denote the set of all worlds over $\Phi$. A *variable assignment* $\sigma$ maps each variable $X \in \mathcal{X}$ to an element of $HU_\Phi$. It is extended to all terms by $\sigma(c) = c$ for all constant symbols $c$ from $\Phi$. The *truth* of classical formulas $\phi$ in $I$ under $\sigma$, denoted $I \models_\sigma \phi$ (or $I \models \phi$ when $\phi$ is ground), is inductively defined by:

- $I \models_\sigma p(t_1, \ldots, t_k)$ iff $p(\sigma(t_1), \ldots, \sigma(t_k)) \in I$;
- $I \models_\sigma \neg\phi$ iff not $I \models_\sigma \phi$; and $I \models_\sigma (\phi \wedge \psi)$ iff $I \models_\sigma \phi$ and $I \models_\sigma \psi$.

A *probabilistic interpretation* $Pr$ is a probability function on $\mathcal{I}_\Phi$ (that is, since $\mathcal{I}_\Phi$ is finite, a mapping $Pr\colon \mathcal{I}_\Phi \to [0,1]$ such that all $Pr(I)$ with $I \in \mathcal{I}_\Phi$ sum up to 1). The *probability* of a classical formula $\phi$ in $Pr$ under a variable assignment $\sigma$, denoted $Pr_\sigma(\phi)$ (or $Pr(\phi)$ when $\phi$ is ground), is defined as the sum of all $Pr(I)$ such that $I \in \mathcal{I}_\Phi$ and $I \models_\sigma \phi$. For classical formulas $\phi$ and $\psi$ with $Pr_\sigma(\phi) > 0$, we use $Pr_\sigma(\psi|\phi)$ to abbreviate $Pr_\sigma(\psi \wedge \phi) \,/\, Pr_\sigma(\phi)$. The *truth* of probabilistic formulas $F$ in $Pr$ under a variable assignment $\sigma$, denoted $Pr \models_\sigma F$, is inductively defined as follows:

- $Pr \models_\sigma (\psi|\phi)[l,u]$ iff $Pr_\sigma(\phi)=0$ or $Pr_\sigma(\psi|\phi)\in[l,u]$;
- $Pr \models_\sigma \neg F$ iff not $Pr \models_\sigma F$; and $Pr \models_\sigma (F \wedge G)$ iff $Pr \models_\sigma F$ and $Pr \models_\sigma G$.

A probabilistic interpretation $Pr$ is a *model* of a probabilistic formula $F$ iff $Pr \models_\sigma F$ for every variable assignment $\sigma$. We say that $Pr$ is the *canonical model* of an spdl-program $KB=(L,P,C,\mu)$ iff every world $I\in\mathcal{I}_\Phi$ with $Pr(I)>0$ is the canonical model of $(L, P\cup\{p\leftarrow \mid p\in B\})$ for some total choice $B$ of $C$ such that $Pr(I)=\mu(B)$. Notice that every $KB$ has a unique canonical model $Pr$. A probabilistic formula $F$ is a *consequence* of $KB$, denoted $KB \mid\!\sim F$, iff every model of $KB$ is also a model of $F$. A conditional constraint $(\psi|\phi)[l,u]$ is a *tight consequence* of $KB$, denoted $KB \mid\!\sim_{tight} (\psi|\phi)[l,u]$, iff $l$ (resp., $u$) is the infimum (resp., supremum) of $Pr_\sigma(\psi|\phi)$ subject to all models $Pr$ of $KB$ and all variable assignments $\sigma$ with $Pr_\sigma(\phi)>0$. Here, we assume that $l=1$ and $u=0$, when $Pr_\sigma(\phi)=0$ for all models $Pr$ of $KB$ and all $\sigma$.

## 4 Query Processing

The canonical model of an ordinary positive (resp., stratified) normal logic program $P$ has a fixpoint characterization in terms of an immediate consequence operator $T_P$, which generalizes to dl-programs. This can be used for a bottom-up computation of the canonical model of a positive (resp., stratified) dl-program, and thus also for computing the canonical model of an spdl-program and for query processing in spdl-programs.

### 4.1 Canonical Models of Positive Description Logic Programs

For a dl-program $KB=(L,P)$, define the operator $T_{KB}$ on the subsets of $HB_P$ as follows. For every $I\subseteq HB_P$, let

$$T_{KB}(I) \;=\; \{H(r) \mid r\in \mathit{ground}(P),\ I\models_L \ell \text{ for all } \ell\in B(r)\}\,.$$

If $KB$ is positive, then $T_{KB}$ is monotonic. Hence, $T_{KB}$ has a least fixpoint, denoted $\mathit{lfp}(T_{KB})$. Furthermore, $\mathit{lfp}(T_{KB})$ can be computed by finite fixpoint iteration (given finiteness of $P$ and the number of constant symbols in $\Phi$). For every $I\subseteq HB_P$, we define $T^i_{KB}(I)=I$, if $i=0$, and $T^i_{KB}(I)=T_{KB}(T^{i-1}_{KB}(I))$, if $i>0$.

**Theorem 4.1.** *For every positive dl-program $KB=(L,P)$, it holds that $\mathit{lfp}(T_{KB}) = M_{KB}$. Furthermore, $\mathit{lfp}(T_{KB})=\bigcup_{i=0}^{n} T^i_{KB}(\emptyset)=T^n_{KB}(\emptyset)$, for some $n\geq 0$.*

### 4.2 Canonical Models of Stratified Description Logic Programs

We next describe a fixpoint iteration for stratified dl-programs. Using Theorem 4.1, we can characterize the canonical model $M_{KB}$ of a stratified dl-program $KB=(L,P)$ as follows. Let $\widehat{T}^i_{KB}(I)=T^i_{KB}(I)\cup I$, for all $i\geq 0$.

**Theorem 4.2.** *Suppose $KB=(L,P)$ has a stratification $\lambda$ of length $k\geq 0$. Define $M_i\subseteq HB_P$, $i\in\{-1,0,\ldots,k\}$, as follows: $M_{-1}=\emptyset$, and $M_i=\widehat{T}^{n_i}_{KB_i}(M_{i-1})$ for $i\geq 0$, where $n_i\geq 0$ such that $\widehat{T}^{n_i}_{KB_i}(M_{i-1})=\widehat{T}^{n_i+1}_{KB_i}(M_{i-1})$. Then, $M_k=M_{KB}$.*

### 4.3 Query Processing in Stratified Probabilistic Description Logic Programs

Fig. 1 shows Algorithm canonical_model, which computes the canonical model $Pr$ of a given spdl-program $KB = (L, P, C, \mu)$. This algorithm is essentially based on a reduction to computing the canonical model of stratified dl-programs (see step (4)), which can be done using the above finite sequence of finite fixpoint iterations.

**Algorithm canonical_model**

**Input**: spdl-program $KB = (L, P, C, \mu)$.
**Output**: canonical model $Pr$ of $KB$.

1. **for every** interpretation $I \in \mathcal{I}_\Phi$ **do**
2. $Pr(I) := 0$;
3. **for every** total choice $B$ of $C$ **do begin**
4. compute the canonical model $I$ of the stratified dl-program $(L, P \cup \{p \leftarrow \mid p \in B\})$;
5. $Pr(I) := \mu(B)$;
6. **end**;
7. **return** $Pr$.

**Fig. 1.** Algorithm canonical_model

Fig. 2 shows Algorithm tight_answer, which computes tight answers $\theta = \{L/l, U/u\}$ for a given query $?(\beta|\alpha)[L, U]$ to a given spdl-program $KB$. The algorithm first computes the canonical model of $KB$ in step (1) and then the tight answer in steps (2)–(8).

**Algorithm tight_answer**

**Input**: spdl-program $KB = (L, P, C, \mu)$ and probabilistic query $?(\beta|\alpha)[L, U]$.
**Output**: tight answer $\theta = \{L/l, U/u\}$ for $?(\beta|\alpha)[L, U]$ to $KB$.

1. $Pr :=$ canonical_model$(KB)$;
2. $l := 1$;
3. $u := 0$;
4. **for every** ground instance $\beta'|\alpha'$ of $\beta|\alpha$ **do begin**
5. $l := \min(l, Pr(\beta'|\alpha'))$;
6. $u := \max(u, Pr(\beta'|\alpha'))$;
7. **end**;
8. **return** $\theta = \{L/l, U/u\}$.

**Fig. 2.** Algorithm tight_answer

## 5 Related Work

Related approaches can be roughly divided into (a) description logic programs with non-probabilistic uncertainty, (b) probabilistic generalizations of description logics, and (c) probabilistic generalizations of web ontology languages. Note that related work on description logic programs without uncertainty is discussed in [7,8,22].

As for (a), Straccia [28] combines description logic programs with *non-probabilistic uncertainty* using interval annotations. To my knowledge, the present paper and [22] are the first ones on description logic programs with *probabilistic uncertainty*.

As for (b), Giugno and Lukasiewicz [13] present a probabilistic generalization of the expressive description logic $\mathcal{SHOQ}(\mathbf{D})$ behind DAML+OIL, which is based on lexicographic probabilistic reasoning. In earlier work, Heinsohn [15] and Jaeger [20] present probabilistic extensions to the description logic $\mathcal{ALC}$, which are essentially based on probabilistic reasoning in probabilistic logics. Koller et al. [21] present a probabilistic generalization of the CLASSIC description logic, which uses Bayesian networks as underlying probabilistic reasoning formalism. Note that fuzzy description logics, such as the ones by Straccia [26,27], are less closely related to probabilistic description logics, since fuzzy uncertainty deals with vagueness, rather than ambiguity and imprecision.

As for (c), especially the works by Costa [4], Pool and Aikin [24], and Ding and Peng [6] present probabilistic extensions to OWL. In particular, Costa's work [4] is semantically based on multi-entity Bayesian networks, while [6] has a semantics in standard Bayesian networks. In closely related work, Fukushige [11] proposes a basic framework for representing probabilistic relationships in RDF. Finally, Nottelmann and Fuhr [23] present pDAML+OIL, which is a probabilistic generalization of the web ontology language DAML+OIL, and a mapping to stratified probabilistic datalog.

## 6 Summary and Outlook

We have continued the research on probabilistic dl-programs. We have focused on the special case of stratified probabilistic dl-programs. In particular, we have presented an algorithm for query processing in such probabilistic dl-programs, which is based on a reduction to computing the canonical model of stratified dl-programs.

A topic of future research is to further enhance stratified probabilistic dl-programs towards a possible use for Web Services. This may be done by exploiting and generalizing further features of Poole's ICL for dynamic and multi-agent systems [25].

**Acknowledgments.** This work has been supported by a Heisenberg Professorship of the German Research Foundation. I thank the reviewers of this paper for their constructive comments, which helped to improve this work.

## References

1. G. Antoniou. Nonmonotonic rule systems on top of ontology layers. In *Proceedings ISWC-2002*, *LNCS* 2342, pp. 394–398.
2. T. Berners-Lee. *Weaving the Web*. Harper, San Francisco, CA, 1999.
3. H. Boley, S. Tabet, and G. Wagner. Design rationale for RuleML: A markup language for Semantic Web rules. In *Proceedings SWWS-2001*, pp. 381–401.
4. P. C. G. da Costa. Bayesian semantics for the Semantic Web. Doctoral Dissertation, George Mason University, Fairfax, VA, USA, 2005.
5. C. V. Damásio. The W$^4$ Project, 2002. `http://centria.di.fct.unl.pt/~cd/projectos/w4/index.htm`.

6. Z. Ding and Y. Peng. A Probabilistic extension to ontology language OWL. In *Proceedings HICSS-2004*.
7. T. Eiter, T. Lukasiewicz, R. Schindlauer, and H. Tompits. Combining answer set programming with description logics for the Semantic Web. In *Proceedings KR-2004*, pp. 141–151.
8. T. Eiter, T. Lukasiewicz, R. Schindlauer, and H. Tompits. Well-founded semantics for description logic programs in the Semantic Web. In *Proc. RuleML-2004*, *LNCS* 3323, pp. 81–97.
9. D. Fensel, W. Wahlster, H. Lieberman, and J. Hendler, editors. *Spinning the Semantic Web: Bringing the World Wide Web to Its Full Potential*. MIT Press, 2002.
10. A. Finzi and T. Lukasiewicz. Structure-based causes and explanations in the independent choice logic. In *Proceedings UAI-2003*, pp. 225–232.
11. Y. Fukushige. Representing probabilistic knowledge in the Semantic Web. In *Proceedings of the W3C Workshop on Semantic Web for Life Sciences*, Cambridge, MA, USA, 2004.
12. M. Gelfond and V. Lifschitz. Classical negation in logic programs and deductive databases. *New Generation Computing*, 17:365–387, 1991.
13. R. Giugno and T. Lukasiewicz. P-$\mathcal{SHOQ}(\mathbf{D})$: A probabilistic extension of $\mathcal{SHOQ}(\mathbf{D})$ for probabilistic ontologies in the Semantic Web. In *Proc. JELIA-2002*, *LNCS* 2424, pp. 86–97.
14. B. N. Grosof, I. Horrocks, R. Volz, and S. Decker. Description logic programs: Combining logic programs with description logics. In *Proceedings WWW-2003*, pp. 48–57.
15. J. Heinsohn. Probabilistic description logics. In *Proceedings UAI-1994*, pp. 311–318.
16. I. Horrocks and P. F. Patel-Schneider. Reducing OWL entailment to description logic satisfiability. In *Proceedings ISWC-2003*, *LNCS* 2870, pp. 17–29.
17. I. Horrocks and P. F. Patel-Schneider. A proposal for an OWL Rules Language. In *Proceedings WWW-2004*, pp. 723–731.
18. I. Horrocks, P. F. Patel-Schneider, and F. van Harmelen. From $\mathcal{SHIQ}$ and RDF to OWL: The making of a web ontology language. *Journal of Web Semantics*, 1(1):7–26, 2003.
19. I. Horrocks, U. Sattler, and S. Tobies. Practical reasoning for expressive description logics. In *Proceedings LPAR-1999*, *LNCS* 1705, pp. 161–180.
20. M. Jaeger. Probabilistic reasoning in terminological logics. In *Proc. KR-1994*, pp. 305–316.
21. D. Koller, A. Levy, and A. Pfeffer. P-CLASSIC: A tractable probabilistic description logic. In *Proceedings AAAI-1997*, pp. 390–397.
22. T. Lukasiewicz. Probabilistic description logic programs. In *Proceedings ECSQARU-2005*, *LNCS* 3571, pp. 737–749.
23. H. Nottelmann and N. Fuhr. pDAML+OIL: A probabilistic extension to DAML+OIL based on probabilistic Datalog. In *Proceedings IPMU-2004*.
24. M. Pool and J. Aikin. KEEPER and Protégé: An elicitation environment for Bayesian inference tools. In *Proceedings of the Workshop on Protégé and Reasoning* held at the *7th International Protégé Conference*, 2004.
25. D. Poole. The independent choice logic for modelling multiple agents under uncertainty. *Artificial Intelligence*, 94:7–56, 1997.
26. U. Straccia. Reasoning within fuzzy description logics. *Journal of Artificial Intelligence Research*, 14:137–166, 2001.
27. U. Straccia. Towards a fuzzy description logic for the Semantic Web (preliminary report). In *Proceedings ESWC-2005*, *LNCS* 3532, pp. 167–181.
28. U. Straccia. Uncertainty and description logic programs: A proposal for expressing rules and uncertainty on top of ontologies. Technical Report ISTI-2004-TR, CNR Pisa, 2004.
29. A. van Gelder, K. A. Ross, and J. S. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650, 1991.
30. W3C. OWL web ontology language overview, 2004. W3C Recommendation (10 February 2004). Available at `www.w3.org/TR/2004/REC-owl-features-20040210/`.