Sorted Monotonic Logic Programs and their Embeddings

C.V. Damásio       L.M. Pereira

Centro Inteligência Artificial
Universidade Nova de Lisboa
Portugal
Overview

- We present a logic programming-based language allowing for the combination of several lattices of truth-values under arbitrary monotonic operators.

- A model and fixpoint theory are presented, extending the classical results by van Emden and Kowalski.

- The main contributions of the paper are the embedding results of a series of existing logic programming semantics dealing with uncertainty, vagueness, or probabilistic reasoning.

- We provide a comparative overview of the several proposals, all of which are translated into a single unified general framework.
The literature contains dozens of proposals for logic programming languages, where the same results are shown over and over again.

Here, all the details are abstracted out and we identify a minimal set of properties which are general enough to capture most of the existing proposals, while keeping the wide accepted and understood model-theory and fixpoint semantics.

Our proposal uses a sorted language, where each sort identifies an underlying complete lattice of truth-values (weights) with an appropriate implication connective.

We describe embeddings for quantitative deduction, fuzzy, possibilistic logic, annotated and probabilistic-based logic programming languages.
Definition 1. A signature is a pair $\Sigma = \langle S, F \rangle$ where $S$ is a set of elements, the sorts, and $F$ is a collection of pairs $\langle f, s_1 \times \cdots \times s_k \rightarrow s \rangle$ denoting functions, such that $s, s_1, \ldots s_k$ are sorts and no symbol $f$ occurs in two different pairs.

Definition 2. Given $\Sigma = \langle S, F \rangle$, a $\Sigma$-algebra $\mathfrak{A}$ is a pair $\langle \{ A^s \}_{s \in S}, I \rangle$ where:

1. Each $A^s$ is a nonempty set called the carrier of sort $s$,

2. $I$ is a function which assigns a map $I(f) : A^{s_1} \times \cdots \times A^{s_k} \rightarrow A^s$ to each $f : s_1 \times \cdots \times s_k \rightarrow s \in F$, where $k > 0$, and an element $I(c) \in A^s$ to each constant symbol $c : s$ in $F$. 
Arrow $\Sigma$-Algebras

For every sort, we require only the existence of an “implication” operator naturally related to the corresponding partial ordering in the lattice of truth-values, as well as a constant denoting the top element:

**Definition 3.** A $\Sigma$-Algebra $\mathfrak{L}$ is an Arrow $\Sigma$-Algebra whenever:

1. The carrier $L^s$ of each sort is a complete lattice under a partial order $\preceq^s$.

2. Each sort $s$ contains a constant symbol $\top^s : s$ mapped to the top element of $L^s$ and an operator $\leftarrow^s : s \times s \to s$ such that for every pair $x, y$ of $L^s$ we have

   \[(x \ I(\leftarrow^s) \ y) = I(\top^s) \text{ iff } x \succeq^s y\]
Sorted Monotonic Logic Programs

- Sorted monotonic logic programs are constructed from the abstract syntax induced by an arrow $\Sigma$-algebra $\mathcal{L}$ on a set of sorted propositional symbols.

- The arrow $\Sigma$-algebra $\mathcal{L}$ may contain arbitrary many monotone operators.

- The syntax of sorted monotonic logic programs resorts to the term $\Sigma$-algebra of formulas $\mathfrak{F} = \text{Terms}(\Sigma, \Pi)$, generated from an infinite set of sorted propositional symbols $\Pi$ and the function symbols in the signature of $\mathcal{L}$.

  To denote that a symbol $A \in \Pi$ has sort $s$ we will often write $A \in \Pi^s$. 

A sorted monotonic logic program is a set $\mathcal{P}$ of rules of the form $A \leftarrow^s B$ where:

1. The rule $A \leftarrow^s B$ is a formula (an algebraic term) of $\mathfrak{F}$;

2. The head $A$ of the rule is a propositional symbol of $\Pi$ of sort $s$.

3. The body $B$ is a formula of $\mathfrak{F}$ with sort $s$, built from sorted propositional symbols $B_1, \ldots, B_n$ ($n \geq 0$) by the use of function symbols in $\Sigma$.

4. Facts are rules with body $\top^s$. 


Semantics of Sorted Monotonic Logic Programs

▶ An interpretation $I$ assigns to every propositional symbol (of sort $s$) a value (in the carrier of $s$).

▶ A rule $A \leftarrow^s B$ is satisfied by $I$ iff $I(A) \succeq^s \hat{I}(B)$

▶ An interpretation $I$ is a model of a program iff all the rules in the program are satisfied by $I$. 
Declarative Semantics of Sorted Monotonic Logic Programs

- The ordering in the lattice of truth-values can be extended to the set of interpretations in the obvious way.

- The semantics of a sorted monotonic logic program $\mathbb{P}$ is given by the least model $M_\mathbb{P}$ of $\mathbb{P}$, which always exists.

- The least model can be obtained by “iterating” the immediate consequences monotonic operator
The immediate consequences operator, given by van Emden and Kowalski, can be easily generalised to the framework of sorted monotonic logic programs.

**Definition 4.** Let $\mathbb{P}$ be a sorted monotonic logic program. The immediate consequences operator $T_\mathbb{P}$ maps interpretations to interpretations:

$$T_\mathbb{P}(I)(A) = \bigsqcup_s \{ \hat{I}(B) \mid A \leftarrow^s B \in \mathbb{P} \}$$

where $A$ is an arbitrary propositional symbol of sort $s$, and $\bigsqcup_s$ is the least upper bound in the lattice $L^s$.

As in the classical setting, an interpretation $I$ is a model of $\mathbb{P}$ iff $T_\mathbb{P}(I) \sqsubseteq I$, and thus the least fixpoint of $T_\mathbb{P}$ coincides with the least model of $\mathbb{P}$.

The $T_\mathbb{P}$ operator is monotonic, but might not be continuous.
Remark on the conditions of an arrow $\Sigma$-algebra

The following necessary and sufficient condition below is essential to guarantee the existence of a least model for every program:

\[(x \ I(\leftarrow^s) \ y) = I(\top^s) \iff x \succeq^s y\]

- If we enforce only

\[(x \ I(\leftarrow^s) \ y) = I(\top^s) \text{ then } x \succeq^s y\]

then the least fixpoint might not be a model of the program.

- If we enforce only

\[x \succeq^s y \text{ then } (x \ I(\leftarrow^s) \ y) = I(\top^s)\]

then the least fixpoint might not be the minimal model of the program.
The seminal paper by van Emden [JLP86] defined the notion of quantitative rules, extending the work by Shapiro [IJCAI83].

Quantitative rules are an extension of Horn clauses with an “attenuation factor”, in our syntax having the form

\[ A \leftarrow f \times \min(B_1, \min(\ldots, \min(B_{n-1}, B_n)\ldots)) \]

There is only one sort with carrier [0.0,1.0].
Marek and Truszczynski [UK00] defined logic programs with costs under two interpretations:

- The parallel-time approach:
  
  \[ A \leftarrow c + \max(B_1, \ldots, \max(B_{n-1}, B_n)) \ldots \]

- The no-reusability approach:
  
  \[ A \leftarrow c + B_1 + \ldots + B_n \]

There is only one sort with carrier \([+\infty, 0]\), meaning that \( \text{lub} = \inf \).

A quantitative rule with attenuation factor \( f \) can be translated into logic programming with costs rule under the parallel-time approach with cost \(-\log f\).
Fuzzy Logic Programming

Vojtáš and Paulík [ELP96] propose a language and corresponding model theory for generalizing definite logic programming, based on Pavelka’s and Hájek’s works.

Rules have the form:

\[ A \leftarrow_1 q \otimes_1 (B_1 \otimes_2 \ldots \otimes_2 B_n) \]

There is only one sort with carrier \([0, 1]\), and \(\otimes_1\) and \(\otimes_2\) are t-norms.

Quantitative Deduction framework of van Emden is an instance of the previous one, by letting \(\leftarrow_1\) be Goguen implication, \(\otimes_1\) its adjunction (product), and \(\otimes_2\) be Gödel t-norm (minimum).
Fuzzy Logic Programming

Later on, Vojtáš [FSS01], extended his previous work by building the bodies of rules from arbitrary conjunctors, disjunctors, and aggregators (monotone operators).

\[ A \leftarrow_i \&_i (Body[B_1, \ldots, B_n], q) \]

There is only one sort with carrier \([0, 1]\), where \(\&_i\) is a conjunctor (extends classical conjunction) that evaluates many-valued Modus Ponens wrt implication \(\leftarrow_i\).

The FPROLOG interpreter of Martin, Baldwin and Pilsworth [FSS87] describes an implementation of a language which can be seen as an instance of Vojtáš scheme.
Residuated Logic Programming

The restriction to the unit interval carrier has been lifted by ourselves in [ECSQARU01]. We assume a single complete residuated lattice $P$ with implication $\leftarrow$ and conjunctor $\&$ such that:

- $(P, \&, \top)$ is a commutative monoid.

- Operation $\&$ is isotonic: if $x_1, x_2, y \in P$ such that $x_1 \leq_P x_2$ then $(x_1 \& y) \leq_P (x_2 \& y)$ and $(y \& x_1) \leq_P (y \& x_2)$;

- Operation $\leftarrow$ is isotonic in the first argument (the consequent) and antitonic in the second argument (the antecedent), i.e. if $x_1, x_2, y \in P$ such that $x_1 \leq_P x_2$ then $(x_1 \leftarrow y) \leq_P (x_2 \leftarrow y)$ and $(y \leftarrow x_2) \leq_P (y \leftarrow x_1)$;

- For any $x, y, z \in P$, then $x \leq_P (y \leftarrow z)$ holds iff $(x \& z) \leq_P y$ holds.

Fitting’s pseudo-Boolean valued Prolog [Studia Logica 87] is an instance of the above, where the conjunctor is the only operator allowed in the body of rules.
Residuated Logic Programs have been extended by Medina, Aciego and Vojtáš in [LPNMR01], by

- Allowing several implications in the language
- Dropping commutative and associativity of the conjunctors, in the spirit of Fuzzy Logic Programming

A sorted version of Multi-adjoint Logic Programming has been proposed by Damásio, Medina, and Aciego [IPMU04], with termination results.

A multi-adjoint logic programming rule $\langle A \leftarrow^s_i B, \vartheta \rangle$ can be immediately translated into the monotonic one $A \leftarrow^s \vartheta \&_i^s B$

Sorted Multi-Adjoint Logic programming allows to associate weights to rules, and there is a least model $I$ where all rules are satisfied, i.e. $\hat{I}(A \leftarrow^s_i B) \succeq^s \vartheta$. 
Possibilistic Logic Programming

A (ground) possibilistic logic program of Dubois, Lang and Prade [ICLP91] is a set of rules of the form

\[ A \leftarrow \min(\alpha, \min(B_1, \min(\ldots, \min(B_{n-1}, B_n)\ldots)) \]

corresponding to the Fuzzy Logic Programming rule, where \(\leftarrow\) is interpreted as Gödel’s implication. The parameter \(\alpha\) is the necessity degree of the rule.

Possibilistic Logic Programming is also an instance of Quantitative Deduction, and Kullman and Sandri [FSS] have proposed a translation into annotated logic programming.
Weighted logic programming with Fuzzy Constants

Weighted logic programming with Fuzzy Constants of Alsinet and Godo [IJIS01] has a possibilistic semantics, and a sound automated deduction method is presented which resembles very much the ideas underlying possibilistic logic programming.

We conjecture that grounded WLFC programs correspond to ground possibilistic logic programs with extra rules of the form:

\[ p(t_1, \ldots, t_n) \leftarrow \min(\beta, p(s_1, \ldots s_m)) \]

with \( \beta = \min \{\beta_1, \ldots, \beta_m\} \) where each \( \beta_i \) is the unification degree of ground term \( s_i \) with ground term \( t_i \).
Kifer and Subrahmanian’s Generalized Annotated Logic Programs [JLP92] consist of annotated clauses:

\[ A : \rho \leftarrow B_1 : \mu_1 \& \ldots \& B_n : \mu_n \]

The reading of the above clause is “if \( B_1 \geq \mu_1, \ldots, B_n \geq \mu_n \) then \( A \geq \rho \)”.

Annotated clauses are built from first-order atoms, and \( \mu_1, \ldots, \mu_n \) might be constant and variable annotations, while \( \rho \) is a complex annotation.

The translation into sorted monotonic logic programs is complex, but the major feature is that variable annotations are all discarded resulting in rules of the form

\[ A \leftarrow (\rho' \& (B_{l_1} \leftarrow \mu_{l_1}) \& \ldots \& (B_{l_o} \leftarrow \mu_{l_o})) \]
Signed Formula Logic Programming

Lu’s signed formula logic programs [JLC96] are constructed from a set of truth-values \( \Delta \) and expressions from a many-valued logic. The least model can be guaranteed to exist when signed clauses have the special form:

\[
s_0 : A \leftarrow s_1 : A_1 \& \ldots \& s_n : A_n
\]

Interpretations map ground atoms to subsets of \( \Delta \). The intuitive meaning of a signed clause is that “if \( I(A_1) \subseteq s_1, \ldots, I(A_n) \subseteq s_n \) then \( I(A) \subseteq s_0 \).”

A signed clause can be translated into a single sorted monotonic rule

\[
A \in (s_0 \cup (A_1 \in s_1) \cup \ldots \cup (A_n \in s_n))
\]

The carrier is the powerset of \( \Delta \), least upper bound is set intersection and

\[
X \in Y = \{\} \text{ if } X \subseteq Y, \text{ otherwise } \Delta
\]
Cao’s Annotated Fuzzy Logic Programs (AFLP) are a formal basis for fuzzy logic programming systems involving soft data [FSS00], inspired by GAPs.

Here annotations denote fuzzy sets in several universes of discourses. We have a sort for each such universe, where least upper bound is intersection of fuzzy sets.

An ideal and restricted semantics in the style of GAPS is given, implementing the Fuzzy Modus Ponens model of Magrez and Smetz [IJIS89].
Ordinary Probabilistic Logic Programs

Lukasiewicz's probabilistic logic programs [ACM TCL01] are sets of conditional constraints

\[(H \mid B)[c_1, c_2]\]

where \(H\) is a conjunction of atoms and \(B\) is either a conjunction of atoms or \(\top\), and \(c_1 \leq c_2\) are rational numbers in the interval \([0, 1]\).

These conditional constraints express that the conditional probability of \(H\) given \(B\) is between \(c_1\) and \(c_2\) or that the probability of the antecedent is 0.

Ordinary probabilistic logic programs are probabilistic logic programs where the conditional constraints have the restricted form

\[(A \mid B_1 \land \ldots \land B_n)[c, 1] \text{ or } (A \mid \top)[c, 1]\]

Under positively correlated probabilistic interpretations, these rules correspond to quantitative ones with attenuation factor \(c\).
Hybrid Probabilistic Logic Programs

Hybrid Probabilistic Logic Programs of Dekhtyar and Subrahmanian [JLP00] have been proposed for constructing rule systems which combine probabilistic information under different probabilistic strategies.

Conjunctive (disjunctive) probabilistic strategies are pairwise combinations of t-norms (t-conorms) over pairs of real numbers in the unit interval $[0,1]$.

A hybrid probabilistic program is a finite set of hp-clauses of the form

$$B_0 : \mu_0 \leftarrow B_1 : \mu_1 \land \ldots \land B_k : \mu_k$$

A hp-clause means that “if the probability of $B_1$ falls in the interval $\mu_1$ and . . . and the probability of $B_k$ falls within the interval $\mu_k$, then the probability of $B_0$ lies in the interval $\mu_0$”.

We have defined in [Studia 02] an (exponential) embedding of HPLPs into residuated ones.
Probabilistic Deductive Databases

A theory of probabilistic deductive databases is proposed by in Lakshmanan and Sadri’s in [TPLP01], where belief and doubt can both be expressed.

Probabilistic programs (p-programs) are finite sets of triples of the form:

\[
\left( A \leftarrow B_1, \ldots, B_n; \mu_r, \mu_p \right)
\]

Truth-values are confidence levels, i.e. pairs of intervals \(\langle [\alpha, \beta], [\gamma, \delta] \rangle\).

The translation into a sorted monotonic logic program is more complex because disjunction modes do not have the properties of a least upper bound operator in the truth-ordering, with the exception of positive correlation:

\[
A \leftarrow \bigvee_{\mu_p} \left( c_i \land_{\mu_{r_i}} \bigwedge_{\mu_{r_i}} B_{i}^{j} \right)
\]
Conclusions

- We have presented a sorted monotonic logic programming language capable of capturing and combining several reasoning paradigms dealing with uncertainty and vagueness.

- Many of the (recent!) semantics are actually instances of work in quantitative deduction from the middle eighties. The work by van Emden was generalized and given a logical interpretation by Fuzzy Logic Programming.

- Multi-adjoint logic programming and residuated logic programming are each a further generalization of Fuzzy Logic Programming, no longer restricted to the real unit interval lattice $[0, 1]$.

- We argue that Monotonic Logic Programming is preferable to Generalized Annotated Logic Programming because our framework requires smaller programs to represent the same knowledge.
Conclusions

▶ The probabilistic-based frameworks are more difficult to handle by sorted monotonic logic programs due to the complexity of inference in these systems.

▶ Both Hybrid Probabilistic Logic Programs and Probabilistic Deductive Databases are captured, but do require complex translations.

▶ The general probabilistic logic programs of Lukasiewicz are not covered as well as the very sophisticated and prominent Fril.

▶ The connection of Bayesian Networks with Logic Programming has been proposed but the exact relationship to our framework requires further enquiry.