Labelled Variables in Logic Programming: Foundations

Roberta Calegari[1], Enrico Denti[1], Agostino Dovier[2], and Andrea Omicini[1]

[1] Dipartimento di Informatica, Scienza e Ingegneria (DISI)
ALMA MATER STUDIORUM–Università di Bologna, Italy
{roberta.calegari, enrico.denti, andrea.omicini}@unibo.it
[2] Dipartimento di Scienze Matematiche, Informatiche e Fisiche
Università degli Studi di Udine, Italy
agostino.dovier@uniud.it

Abstract. We define a new notion of truth for logic programs extended with *labelled variables*, interpreted in non-Herbrand domains. There, usual terms maintain their Herbrand interpretations, whereas diverse domain-specific computational models depending on the local situation of the computing device can be expressed via suitably-tailored labelled models. After some introductory examples, we define the theoretical model for *labelled variables in logic programming* (LVLP). Then, we present both the fixpoint and the operational semantics, and discuss their correctness and completeness, as well as their equivalence.

1 Introduction

To face the challenges of today's pervasive systems – inherently complex, distributed, situated, and intelligent [14] – suitable models and technologies are required to effectively support *distributed situated intelligence*. Once it is extended to capture the many different domains where situated intelligence is required, logic programming (LP) has the potential to work at the core of such models and technologies based on its declarative representation and inferential capabilities.

Along this line, in this paper we present a logic programming extension based on *labelled variables* [8, 10], the Labelled Variables model in Logic Programming (LVLP). There, diverse computational models can be tailored to the *local* needs of situated components by means of suitable labelled variables systems, and work together against the common LP background. In fact, whereas logic-based approaches are natural candidates for intelligent systems, a pure LP approach seems not to fit well the needs of situated systems. Instead, a hybrid approach makes it possible in principle to exploit LP for what it is most suited for – symbolic computation –, while possibly delegating other aspects – such as situated computations – to other languages, or to other levels of computation.

Most other works in this area – such as [1, 2, 9, 16, 17] – primarily focus onto specific scenarios and sorts of systems—e.g. modal logic, deductive systems, fuzzy systems, etc. Instead, our model aims at providing a general-purpose framework and mechanism which, while seamlessly rooted in the LP landscape, could fit several relevant context-specific systems.

2 Context, Motivation & Examples

In nowadays complex and pervasive scenario, where software engineering, programming languages, and distributed artificial intelligence meet, logic-based languages have the potential to play a prominent role both as *intelligence providers* and *technology integrators* [15]. Typical LP features – such as programs as logic theories, computation as deduction, and programming with relations and inference – make logic languages a natural choice for building intelligent components. Nevertheless, diverse computational models tailored to the local needs of situated systems can be required in order to capture different paradigms and languages of the distributed environment.

Along this line, we introduce the notion of *label* to tag logic variables, and the associated computational model for their elaboration, which proceeds along with the resolution procedure without being strictly subject to the standard LP rules. Generally speaking, labels provide a customisable computational model, which can be charged of the domain-specific – and possibly not declarative – part of the computation, and bound to the standard LP computation.

Before formally defining (Section 3) the Labelled Variables model in Logic Programming (LVLP), we informally discuss some examples to give the flavour of our approach. For instance, in a WordNet-like scenario [18], representing a network of semantically-related words, labels (unlike standard LP) offer a way to split the representation onto two different levels, each one with its own computational model: logic variables stand for individual objects in the domain, while labels can be exploited to represent the network of related words.

Following our prototype syntax, `X^`*`label`* is a labelled variable denoting a logic variable `X` labelled with *`label`*—where *`label`* is a term in the set of admissible labels defined by the user. In a LVLP clause, the list of the labelled variables precedes the remaining part of the body. Given the following LVLP program:

```
  wordnet_fact([‘dog’,‘domestic dog’,‘canis’,‘pet’,‘mammal’,‘vertebrate’]).
  wordnet_fact([‘cat’, ‘domestic cat’, ‘pet’, ‘mammal’, ‘vertebrate’]).
  wordnet_fact([‘fish’, ‘aquatic vertebrates’, ‘vertebrate’]).
  wordnet_fact([‘frog’, ‘toad’, ‘anauran’, ‘batrachian’]).
  animal(X) :- X^[‘pet’], X = ‘minnie’.
  animal(X) :- X^[‘fish’], X = ‘nemo’.
  animal(X) :- X^[‘cat’], X = ‘vmolly’.
  animal(X) :- X^[‘dog’], X = ‘frida’.
  animal(X) :- X^[‘frog’], X = ‘cra’.
```

the following query, looking for a `pet` animal, generates four solutions:

```
?- X^[‘pet’], animal(X).

yes. X / ‘minnie’
[X^[‘dog’, ‘domestic dog’, ‘canis’, ‘pet’, ‘mammal’, ‘vertebrate’]] ;

yes. X / ‘minnie’
[X^[‘cat’, ‘domestic cat’, ‘pet’, ‘mammal’, ‘vertebrate’]] ;

yes. X / ‘molly’
[X^[‘cat’, ‘domestic cat’, ‘pet’, ‘mammal’, ‘vertebrate’]] ;

yes. X / ‘frida’
[X^[‘dog’, ‘domestic dog’, ‘canis’, ‘pet’, ‘mammal’, ‘vertebrate’]]
```

Looking instead for a less specific `vertebrate` produces five solutions:

```
?- X^[vertebrate], animal(X).

yes. X = ‘minnie’
X^[‘dog’, ‘domestic dog’, ‘canis’, ‘pet’, ‘mammal’, ‘vertebrate’] ;

yes. X = ‘minnie’
X^[‘cat’, ‘domestic cat’, ‘pet’, ‘mammal’, ‘vertebrate’] ;

yes. X = ‘molly’
X^[‘cat’, ‘domestic cat’, ‘pet’, ‘mammal’, ‘vertebrate’] ;

yes. X = ‘frida’
X^[‘dog’, ‘domestic dog’, ‘canis’, ‘pet’, ‘mammal’, ‘vertebrate’] ;

yes. X = ‘nemo’
X^[‘fish’,‘aquatic vertebrates’, ‘vertebrate’]
```

A relevant aspect of LVLP is that labels are not subject to the single-assignment assumption: each time two labelled variables unify, their labels are processed and *combined* according to a user-defined function that embeds the desired computational model, and the resulting label is associated to the unified variable. Thus, the LP model *per se* is left untouched, whereas diverse computational models can be defined next to it, possibly influencing the result of a logic computation by *restricting* the set of admissible solutions according to each specific domain.

For instance, let us consider the case in which the knowledge base consists of a collection of ingredients, each featuring properties such as calories count, amount of proteins, fat, etc. There, labels could be used to describe the weight, calories and category of the food, as in the program below:

```
meat(M) :- M^[100g, 156cal, [alternativeMeats]], M=[‘tofu meatballs’].
meat(M) :- M^[100g, 145cal, [curedMeats]],       M=[‘Parma ham’].
meat(M) :- M^[100g, 210cal, [curedMeats]],       M=[‘ham’].
meat(M) :- M^[200g, 220cal, [whiteMeats]],       M=[‘chicken’].
vegetable(V) :- V^[100g,  35cal, [vegetables]], V=[‘carrots’].
vegetable(V) :- V^[ 50g,  10cal, [vegetables]], V=[‘salad’].
bread(B) :- B^[100g, 265cal, [whiteBread]],    B=[‘bread with poolish’].
bread(B) :- B^[100g, 240cal, [integralBread]], B=[‘whole grain bread’].
recipe(R) :- R^◇, meat(Meat), vegetable(Vegetable), R=[Meat,Vegetable].
recipe(R) :- R^◇, Meat^[any, any, [curedMeats]], bread(Bread), meat(Meat),
             vegetable(Vegetable), R=[Bread, Meat, Vegetable].
```

Labels (with ◇ representing the *any* label) make it possible, for instance, to ask for a meal with an upper bound of calories, with a given tolerance. Assuming, e.g., that the label-combining function accounts for a 10% tolerance, a possible query asking for a recipe with 200(±20) calories could be the following:

```
  ?- R^[any, 200cal, any], recipe(R).

  yes. R / [‘tofu meatballs’, ‘carrots’]
  R^[200g, 191cal, [alternativeMeats, vegetables]] ;

  yes. R / [‘tofu meatballs’, ‘salad’]
  R^[200g, 166cal, [alternativeMeats, vegetables]] ;

  yes. R / [‘Parma ham’, ‘carrots’]
  R^[200g, 180cal, [curedMeats, vegetables]] ;

  yes. R / [‘ham’, ‘salad’]
  R^[150g, 220cal, [curedMeats, vegetables]]
```

Comparing the labels of R in the query and in the answers should make it clear how avoiding single assignment in the labels (not in the logic variable) makes the language more expressive, without harming the logic computation.

Multiple matching criteria could also be applied simultaneously—for instance, combining calories with a specific food category:

```
?- R^[any, 180cal, [alternativeMeats]], recipe(R).

yes. R / ['tofu meatballs','carrots']
R^[200g, 191cal, [alternativeMeats, vegetables]] ;

yes. R / ['tofu meatballs','salad']
R^[150g, 166cal, [alternativeMeats, vegetables]]
```

All “standard” domains for logic languages (like the CLP ones [4]) are also supported. For instance, labels could be used to represent the integer interval over which the logic variable values span: accordingly, the label syntax could take the form `X^[`*min*`,`*max*`]`, and a simple interval program could look as follows:

```
    interval(X):- X^[-1,4].
    interval(X):- X^[6,10].
```

The unification of two variables labelled with an interval would then result in a variable labelled with the intersection of the intervals. Accordingly, the following simple query generates two solutions:

```
    ?- X^[2,7], interval(X).

    yes.
    X^[2,4] ;

    yes.
    X^[6,7]
```

In addition, the expressiveness of LVLP makes it possible to easily move from the domain of integer intervals to more complex domains, like for instance the domain of integers with a neighbourhood, as in the following program:

```
    neighbourhood(X):- X^[-1,4], X=3.
    neighbourhood(X):- X^[6,10], X=8.
```

There, constant values unify with labelled variables if they belong to the interval in the label. Accordingly, the following query would result in just one solution:

```
    ?- X^[2,7], neighbourhood(X).

    yes.  X / 3
    X^[2,4]
```

given that the second clause would set X out of the interval in the query.

3 The Labelled Variables Model in Logic Programming

3.1 The model

Let $\mathscr{C}$ be the set of *constants*, with $c_1, c_2 \in \mathscr{C}$ being two generic constants. Let $\mathscr{V}$ be the set of *variables*, with $v_1, v_2 \in \mathscr{V}$ being two generic variables. Let $\mathscr{F} \in \mathscr{H}$ be the set of *function symbols*, with $f_1, f_2 \in \mathscr{F}$ being two generic function symbols. $f \in \mathscr{F}$ is assigned to an arity $ar(f) > 0$ that states the number of arguments. Let $\mathscr{T} \subseteq \mathscr{H}$ be the set of *terms*, with $t_1, t_2 \in \mathscr{T}$ being two generic terms. Terms can be either simple (constant, variable) or compound (when containing a $f \in F, ar(f) > 0$). Let $\mathscr{H}$ be the *Herbrand universe* and $\mathscr{H}_B$ be the *Herbrand base*.

A *Labelled Variables Model in Logic Programming* is defined as follows.

- A set $\mathscr{B} = \{\beta_1, \ldots, \beta_n\}$ of *basic labels* that are the basic entities of the *domain of labels*.[3]
- A set $\mathscr{L} \subseteq \wp(\mathscr{B})$ of *labels*, where each $\ell \in \mathscr{L}$ is a subset of $\mathscr{B}$, namely a set of basic labels (e.g., $\ell = \{\beta_1, \beta_3, \beta_6\}$). A "singleton" label $\{\beta\}$ where $\beta \in \mathscr{B}$ will be identified simply by β in the following for the sake of simplicity.
- A set of *variables* $\mathscr{V}$, where $v \in \mathscr{V}$ is the generic variable.
- A *labelling* which is a set of pairs $\langle v_i, \ell_i \rangle$, associating the label ℓ_i to the variable v_i. A "singleton" labelling $\{\langle v, \ell \rangle\}$ – a *labelled variable* – will be identified simply by $\langle v, \ell \rangle$ in the following for the sake of simplicity.
- A *(label-)combining function* f_L, synthesising a new label from two given ones
$$f_L : \mathscr{L} \times \mathscr{L} \longrightarrow \mathscr{L}$$
We require that f_L be associative, commutative, and idempotent (briefly, ACI), namely that $f_L(\ell_1, f_L(\ell_2, \ell_3)) = f_L(f_L(\ell_1, \ell_2), \ell_3), f_L(\ell_1, \ell_2) = f_L(\ell_2, \ell_1), f_L(\ell, \ell) = \ell$. As discussed later, in some cases (e.g., incompatibility), the output of f_L can be the empty set $\emptyset$ which is a signal for "wrong".

3.2 Programs, clauses, unification

In a labelled variables logic program a rule is represented as

$$Head \leftarrow Labellings, Body.$$

where *Head* is an atomic formula, *Labellings* is the list of Variable/Label associations in the clause and *Body* is a list of atomic formulas. This can be read declaratively as *Head if Body* given *Labellings*.

The unification of two labelled variables is represented by the extended tuple (true/false, θ, ℓ), where true/false represents the existence of an answer, θ represents the most general substitution, and ℓ represents the new label associated to the unified variables defined by the (label-)combining function f_L.

More generally, considering all the variables of the goal, the solution is represented by the extended tuple (true/false, Θ, A), where true/false represents the existence of an answer, Θ represents the most general substitutions for all such variables, and A represents the corresponding label-variable associations.

[3] In this paper we require that $\mathscr{B}$ is finite and it is not a real restriction for the applications we discuss. An infinite set of basic labels can be considered anyway, as soon as we can guarantee a sort of solution compactness of the extended unification algorithms involved.

	constant c_2	variable v_2	labelled variable $\langle v_2, \ell_2 \rangle$	compound term s_2
constant c_1	if $c_1 = c_2$ then true else false	true, $\{v_2/c_1\}$	if $f_C(c_1, \ell_2) =$ true then true, $\{v_1/c_2\}$, ℓ_1 else false	false
variable v_1	true, $\{v_1/c_2\}$	true, $\{v_1/v_2\}$	true, $\{v_1/v_2\}, \ell_2$	if v_1 does not occur in s_2 then true, $\{v_1/s_2\}$ else false
labelled variable $\langle v_1, \ell_1 \rangle$	if $f_C(c_2, \ell_1) =$ true then true, $\{v_1/c_2\}$, ℓ_1 else false	true, $\{v_1/v_2\}, \ell_1$	if $f_L(\ell_1, \ell_2) \neq \emptyset$ then true, $\{v_1/v_2\}$, $f_L(\ell_1, \ell_2)$ else false	if v_1 does not occur in s_2 and $f_L(\ell_1, f_L(\ell_2, ..., f_L(\ell_{n-1}, \ell_n)...)) \neq \emptyset$ where $\ell_2, ...\ell_n$ are labels of variables in s_2 then true, $\{v_1/s_2\}$, $f_L(\ell_1, f_L(\ell_2, ..., f_L(\ell_{n-1}, \ell_n)...))$ else false
compound term s_1	false	if v_2 does not occur in s_1 then true, $\{v_2/s_1\}$ else false	if v_2 does not occur in s_1 and $f_L(\ell_1, f_L(\ell_2, ..., f_L(\ell_{n-1}, \ell_n)...)) \neq \emptyset$ where $\ell_2, ...\ell_n$ are labels of variables in s_1 then true, $\{v_2/s_1\}$, $f_L(\ell_1, f_L(\ell_2, ..., f_L(\ell_{n-1}, \ell_n)...))$ else false	if s_1 and s_2 have same functor and arity and arguments (recursively) unify then true else false

Table 1. Unification rules in LVLP

3.3 Compatibility

The tuple ($\mathsf{true}/\mathsf{false}$, Θ, A) represents the solution of the labelled variables program. The implicit assumption is that the LP computation, whose answer is given by Θ, and of the label computation, whose answer is given by A, can be read independently from each other from the domain expert's viewpoint.

This is as to say that, in principle, any computed label-variable association is "acceptable" in the domain—for instance, if LP computes over numbers and labels over colors, each number/color association might be equally valid.

For some application scenarios, however, this might not necessarily be the case: if the LP and label computations somehow capture different views over the same domain (or two interconnected domains), some label-variable association could actually be unacceptable from the domain expert's viewpoint—for instance, in the above example even numbers could not be red, or alike.

In order to formalise such a notion of acceptability, let us introduce the *compatibility function* f_C as:

$$f_C : \mathscr{H} \times \mathscr{L} \longrightarrow \{\mathsf{true}, \mathsf{false}\}$$

In particular, given a a ground term $t \in \mathscr{H}$ and label $\ell \in \mathscr{L}$:

$$f_C(t,\ell) = \begin{cases} \mathsf{true} & \text{there exists at least an element of the domain which} \\ & \text{the interpretations of } t \text{ and } \ell \text{ both refer to the same} \\ & \text{entity} \\ \mathsf{false} & \text{otherwise} \end{cases}$$

Example 1 discusses an application scenario where variables are labelled with their admissible numeric interval, formalising the f_L and f_C functions accordingly.

Example 1. Let us suppose that the LP variables in a certain application scenario span over some integer domains and that they are labelled with their admissible numeric interval. In this case, the combining function f_L, which embeds the scenario-specific label semantics, is supposed to intersects intervals: accordingly, f_L is defined so that, given two labels $\ell_1 = \{\beta_{11}, \ldots, \beta_{1n}\}$ and $\ell_2 = \{\beta_{21}, \ldots, \beta_{2m}\}$, the resulting label ℓ_3 is the intersection of ℓ_1 and ℓ_2, as:

$$\ell_3 = f_L(\ell_1,\ell_2) = f_L(\{\beta_{11},\ldots\beta_{1n}\},\{\beta_{21},\ldots\beta_{2m}\}) =$$
$$\{(\beta_{11}\cap\beta_{21}),\cdots,(\beta_{11}\cap\beta_{2m}),\ldots,(\beta_{1n}\cap\beta_{21}),\cdots,(\beta_{1n}\cap\beta_{2m})\}$$

In such a scenario, the LP computation aims to compute numeric values, while the label computation aims to compute admissible numeric intervals for the LP variables.

In principle, the solution tuple ($\mathsf{true}/\mathsf{false}$, Θ, A) could describe situations such as $(\mathsf{true}, X = 3, \langle 3, [4,5]\rangle)$, since the LP computation and the label computation proceed without interfering with each other. However, if numeric intervals are interpreted as the boundaries for acceptable values of LP variables, such label-variable associations could appear to be inconsistent to the domain expert's eyes. So, a reasonable behaviour for the LVLP system should allow the domain expert to embed his/her knowledge, capturing such incompatibility so that the system can reject this solution.

This is what the compatibility function f_C is for: its purpose is to check whether the ground term $t \in \mathscr{H}$ is *compatible* with the interval represented by label $\ell = [A..B], \ell \in \mathcal{L}$. It could be defined so as to return *false* for solutions where the computed LP values do not belong to the numeric interval computed by the corresponding label computation, thus capturing the connection among the two universes.

Summing up, the result of a labelled variables program can be written as:

$$(\mathsf{true/false} \wedge f_C(\Theta, A), \Theta, A)$$

meaning that the truth value potentially computed by the LP computation can be restricted – i.e., forced to false – by the $f_C(\Theta, A)$. In turn, this is a convenient shortcut for the conjunction of all $f_C(t, \ell)$ $\forall(t, \ell)$ couples, where ℓ and t are, respectively, such that the label-variable association $\langle v, \ell \rangle \in A$ and the substitution $v/t \in \Theta$. Of course, in case of independent domains, $f_C(t, \ell)$ is true for any t and any ℓ.

Table 1 reports the unification rules for two generic terms: to lighten the notation, undefined elements in the tuple are omitted—i.e., labels or substitutions that have no sense in a particular situation.

4 Semantics

Labels domains are expected to support tests and operations on labels, in order to guarantee the basic theoretical results of logic programming, such as the equivalence of denotational and operational semantics.

In the following, in Subsection 4.1 we define the denotational (fixpoint) semantics in the context of labels domain (under reasonable requests for f_L), while in Subsection 4.2 we discuss the operational semantics of the model describing an abstract state machine.

4.1 Fixpoint Semantics

Let us call $\mathcal{X} = (\mathscr{H}, \mathcal{L})$ a *labelled variable logic programming domain* and let us define the notion of $\mathcal{X}$-interpretation I as a set of pairs of the form

$$\langle p(t_1, \ldots, t_n), [\ell_1, \ldots, \ell_n] \rangle$$

where $p(t_1, \ldots, t_n)$ is a ground atom and $\ell_1, \ldots, \ell_n$ are labels s.t. for $i = 1, \ldots, n$ the term t_i is *compatible* with the label ℓ_i, i.e. $f_C(t_i, \ell_i) = \mathsf{true}$. Truthness of f_C is based on the LVLP domain $\mathcal{X}$. With a slight abuse of notation, we write $\mathcal{X} \models [\langle t_1, \ell_1\rangle, \ldots, \langle t_n, \ell_n\rangle]$ iff $\bigwedge_{i=1}^{n} f_C(t_i, \ell_i) = \mathsf{true}$. We also write $\mathcal{X} \models (p(t_1, \ldots, t_n), [\ell_1, \ldots, \ell_n])$ if p is a predicate symbol and $\bigwedge_{i=1}^{n} f_C(t_i, \ell_i) = \mathsf{true}$.

We denote as Λ the part of clause body that stores labelling associations. Without loss of generality we assume that there is exactly one label association for each variable in the head. We define the function $\widetilde{f_L}$ that in a sense extends f_L and takes as argument a rule

$$r = h \leftarrow \Lambda, b_1, \ldots, b_n$$

and $n+1$ lists of labels. The rule r is used here to identify multiple occurrences of the variables. Let us assume the variables in h are $x_1, \ldots, x_m$, and consider one of them, say x_i. If x_i occurs in h (and hence in Λ) and in (some of) $b_1, \ldots, b_n$ then the corresponding labels $\ell, \ell_{1,i}, \ldots, \ell_{n,i}$ for x_i are retrieved from Λ (if x_i does not occur in b_j simply we do not consider such contribution). Then $\ell'_i = f_L(\ell, f_L(\ell_{1,i}, \ldots, f_L(\ell_{n-1,i}, \ell_{n,i}) \ldots))$ is computed and the pair $\langle x_i, \ell'\rangle$ is returned. This is done for all variables $x_1, \ldots, x_m$ occurring in the head h and the list $[\ell'_1, \ldots, \ell'_m]$ is returned.

The denotational semantics we are presenting is based on the one-step consequence functions T_P defined as:

$$T_P(I) = \left\{ \begin{array}{ll} \langle p(\widetilde{t}), \widetilde{\ell}\,\rangle : & \\ r = p(\widetilde{x}) \leftarrow \Lambda_{\widetilde{x}} \mid b_1, \ldots, b_n & (1) \\ \text{where r is a fresh renaming of a rule of } P, & \\ v \text{ is a valuation on } \mathscr{H} \text{ such that } v(\widetilde{x}) = \widetilde{t}, & (2) \\ \bigwedge_{i=1}^{n} \exists\, \widetilde{\ell_i} \text{ s.t. } \langle v(b_i), \widetilde{\ell_i}\rangle \in I, & (3) \\ \mathcal{X} \models \Lambda_{\widetilde{t}} \wedge \bigwedge_{i=1}^{n} f_C\big(v(b_i), \widetilde{\ell_i}\big), & (4) \\ \widetilde{\ell} = \widetilde{f}_L\big(r, \Lambda_{\widetilde{t}}, \widetilde{\ell}_1, \cdots, \widetilde{\ell}_n\big) & (5) \end{array} \right\}$$

where $\Lambda_{\widetilde{t}} = v(\Lambda_{\widetilde{x}}) = [\langle t_1, \ell_1\rangle, \ldots, \langle t_n, \ell_m\rangle]$ if $\Lambda_{\widetilde{x}} = [\langle x_1, \ell_1\rangle, \ldots, \langle x_m, \ell_m\rangle]$. It is worth noting that the condition $\bigwedge_{i=1}^{n} f_C\big(v(b_i), \widetilde{\ell_i}\big)$ (line 4) is always satisfied when T_P is used "bottom-up" starting from $I = \emptyset$.

In the following some examples are discussed. Note that, in the LVLP the equality operator is represented by =/2 and its behaviour is summarised by Table 1 (unification rules). One label association is possibly *null* in that might be the *any* label, defined as the neutral element of f_L, henceforth represented as $\diamond$ in the following. As a consequence, any standard (i.e. non labelled) LP variable is interpreted as labelled with $\diamond$. For such reasons, the =/2 in LVLP can be defined as:

$$\mathtt{X} = \mathtt{Y} : -[< \mathtt{X}, \diamond >, < \mathtt{Y}, \diamond >], \mathtt{X} =_{\mathtt{LP}} \mathtt{Y}$$

where $=_{\mathtt{LP}}$ is the standard =/2 LP operator.

Example 2. Now let us consider the labelled variables program P:

```
r(0). r(1). r(2). r(3). r(4).
r(5). r(6). r(7). r(8). r(9).
q(Y,Z) :- Y^[2,4], Z^[3,8], Y=Z, r(Y), r(Z).
p(X,Y,Z) :- X^[0,3], Y^◇, Z^◇, X=Y, q(Y,Z).
```

Let us consider the interpretation $I_0 = \emptyset$. Then:

$$I_1 = T_P(I_0) = \{ \langle r(0), [\diamond]\rangle, \ldots, \langle r(9)[\diamond]\rangle\}$$

Let us apply T_P to I_1:

$$I_2 = T_P(I_1) = I_1 \cup \{\langle q(3,3), [[3,4],[3,4]]\rangle, \langle q(4,4), [[3,4],[3,4]]\rangle\}$$

and now T_P to I_2:

$$I_3 = T_P(I_2) = I_2 \cup \{\langle p(3,3,3), [[3],[3],[3]]\rangle\}$$

which is also the least fixpoint of T_P.

The above example shows the similarities between labelled logic programming on a domain $\mathcal{X}$ (briefly, $\mathsf{LVLP}(\mathcal{X})$) and $CLP(\mathcal{X})$. It would be sufficient, e.g., to replace $[\langle Y, [2,4]\rangle, \langle Z, [3,8]\rangle]$ with $Y\ \mathsf{in}\ 2..4, Z\ \mathsf{in}\ 3..8$. However, this is not true for other domains of labels, as in Example 3.

Example 3. In the WordNet case presented in Section 2, labels are words describing the variable object. The combination of two different labels (combining function) returns a new label only if the two labels have a lexical relation, fail otherwise. The decision is based on the WordNet network [18].

WordNet is a large lexical database of English. Nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms (synsets). Synsets are interlinked by means of conceptual-semantic and lexical relations. The resulting network of meaningfully related words and concepts can be navigated. WordNet superficially resembles a thesaurus, in that it groups words together based on their meanings.

Here, the knowledge base locally stores some WordNet groups (a dynamic consultation to WordNet could be implemented, instead). Let be program P represented by the following facts—where `wordnet_fact` is a simulated wordnet synset, while `animal(Name)` is a predicate describing an animal's name:

```
wordnet_fact(['dog','domestic dog','canis','pet','mammal']).
wordnet_fact(['cat', 'domestic cat', 'pet', 'mammal']).
wordnet_fact(['fish','aquatic vertebrates', 'vertebrate']).
wordnet_fact(['frog','toad', 'anauran', 'batrachian']).

pet_name('minnie').
fish_name('nemo').
animal(X) :- X^['pet'], pet_name(X).
animal(X) :- X^['fish'], fish_name(X).
```

The combining function f_L is supposed to find and return a wordnet synset compatible with both labels. For instance, if $\ell_1 = pet$ and $\ell_2 =$ '*domestic cat*', the new generated label ℓ_3 will be ['*cat*', '*domestic cat*', '*pet*', '*mammal*']. The compatibility function f_C in this scenario is always true, since any animal name is considered compatible with any animal group.

In order to show the construction of T_P, let us consider the interpretation $I_0 = \emptyset$. Then, the subsequent step I_1 can be computed as:

$$I_1 = T_P(I_0) = \{\ \langle pet\_name(\text{'}minnie\text{'}), [\diamond]\rangle, \langle fish\_name(\text{'}nemo\text{'}), [\diamond]\rangle\}$$

Now, let us apply T_P to I_1 to compute I_2:

$$\begin{aligned} I_2 = T_P(I_1) = I_1 \cup \{ & \\ & \langle animal(\text{'}minnie\text{'}), [[\text{'}dog\text{'}, \text{'}domestic\ dog\text{'}, \text{'}canis\text{'}, \text{'}pet\text{'}, \text{'}mammal\text{'}]]\rangle, \\ & \langle animal(\text{'}minnie\text{'}), [[\text{'}cat\text{'}, \text{'}domestic\ cat\text{'}, \text{'}pet\text{'}, \text{'}mammal\text{'}]]\rangle, \\ & \langle animal(\text{'}nemo\text{'}), [[\text{'}fish\text{'}, \text{'}aquatic\ vertebrates\text{'}, \text{'}vertebrate\text{'}]]\rangle\} \end{aligned}$$

which is also the least fixpoint of T_P.

In order to guarantee soundness and completeness of T_P in the following we demonstrate that it is a monotonic and continuous mapping on partial orders.

Definition 1. *An interpretation I is a model of a program P if for each rule $r = p(\widetilde{x}) \leftarrow \Lambda_{\widetilde{x}} \mid b_1, \ldots, b_n$ of P and each valuation v of the variables in r on $\mathscr{H}$ (let us denote with $\widetilde{t} = v(\widetilde{x})$) it holds that*

- *if for $i = 1, \ldots, n$ there are $\langle v(b_i), \widetilde{\ell}_i\rangle \in I$*
- *such that $\mathcal{X} \models f_C(v(b_i), \widetilde{\ell}_i)$ and*
- *$\mathcal{X} \models \Lambda_{\widetilde{t}}$, then it must be*
- *$\big\langle p(\widetilde{t}), \widetilde{f}_L\big(r, \Lambda_{\widetilde{t}}, \widetilde{\ell}_1, \cdots, \widetilde{\ell}_n\big)\big\rangle \in I$.*

Proposition 1. *Given a* LVLP *program P and a* LVLP *domain $\mathcal{X}$, T_P is (1) monotonic and (2) continuos and (3) $T_P(I) \subseteq I$ iff I is a model of P.*

Proof. (1) Let I and J be two interpretations such that $I \subseteq J$. We need to prove that $T_P(I) \subseteq T_P(J)$. If $a = \langle p(\widetilde{t}), \widetilde{\ell}\ \rangle \in T_P(I)$ it means that there is a clause $r \in P$, a valuation v on $\mathscr{H}$ for the variables in r and n elements $\langle v(b_i), \widetilde{\ell}_i\rangle \in I$ satisfying the remaining conditions. Since $I \subseteq J$ they belong to J as well. Then $a \in T_P(J)$.

(2) Let us consider a chain of interpretations $I_0 \subseteq I_1 \subseteq \ldots$. We need to prove that: $T_P\big(\bigcup_{k=0}^{\infty} I_k\big) = \bigcup_{k=0}^{\infty} T_P(I_k)$.
($\subseteq$) Let $a = \langle p(\widetilde{t}), \widetilde{\ell}\ \rangle \in T_P\big(\bigcup_{i=0}^{\infty} I_k\big)$. This means that there is a clause $r \in P$, a valuation v on $\mathscr{H}$ for the variables in r and n elements $\langle v(b_i), \widetilde{\ell}_i\rangle \in \bigcup_{k=0}^{\infty} I_k$ satisfying the remaining conditions. This means that there are $j_1, \ldots, j_n$ such that for $i = 1, \ldots, n$ $\langle v(b_i), \widetilde{\ell}_i\rangle \in I_{j_i}$. Let $j = \max\{j_1, \ldots, j_n\}$, since $I_0 \subseteq I_1 \subseteq$

$\cdots \subseteq I_j$ we have that all $\langle v(b_i), \widetilde{\ell_i}\rangle \in I_j$. Thus $a \in T_P(I_{j+1})$ and henceforth $a \in \bigcup_{k=0}^{\infty} T_P(I_k)$.
($\supseteq$) Let $a = \langle p(\widetilde{t}), \widetilde{\ell}\ \rangle \in \bigcup_{k=0}^{\infty} T_P(I_k)$. This means that there is j such that $a \in T_P(I_j)$. Then, due to monotonicity, $a \in T_P\big(\bigcup_{k=0}^{\infty} I_k\big)$.

(3) Let $T_P(I) \subseteq I$ and let $r = p(\widetilde{x}) \leftarrow \Lambda_{\widetilde{x}}\ ,\ b_1, \dots, b_n$ be a generic clause of P, and v be a generic valuation on $\mathscr{H}$ of all the variables of r. If we assume that $\langle v(b_i), \widetilde{\ell_i}\rangle \in I$ for $i = 1, \dots, n$, and that $\mathcal{X} \models f_C(v(b_i), \widetilde{\ell_i})$ and $\mathcal{X} \models \Lambda_{\widetilde{t}}$, then the pair $h = v(p(\widetilde{x}), \widetilde{\ell}) \in T_P(I)$ by definition of T_P—and $\widetilde{\ell}$ is the same of the definition of model. Since $T_P(I) \subseteq I$ then $h \in I$, therefore (since r and v are chosen in general) I is a model of P.

On the other hand, if I is a model of P we prove that $T_P(I) \subseteq I$. Let $a = \langle\, p(\widetilde{t}), \widetilde{\ell}\ \rangle \in T_P(I)$. This means that there is a rule $r = p(\widetilde{x}) \leftarrow \Lambda_{\widetilde{x}}\ ,\ b_1, \dots, b_n$ of P and a valuation v of the variables in r on $\mathscr{H}$ such that for $i = 1, \dots, n$ there are $\big(v(b_i), \widetilde{\ell_i}\big) \in I$ and $\mathcal{X} \models \Lambda_{\widetilde{t}}$ and $\widetilde{\ell} = \widetilde{f_L}\big(r, \Lambda_{\widetilde{t}}, \widetilde{\ell_1}, \cdots, \widetilde{\ell_n}\big)\big)$. such that $\mathcal{X} \models f_C(v(b_i), \widetilde{\ell_i})$. Since I is a model, and $\mathcal{X} \models f_C(v(b_i), \widetilde{\ell_i})$ then $a \in I$. □

Let be $T_P \uparrow \omega$ defined as usual: $T_P \uparrow \omega = \bigcup\{T_P \uparrow k : k \geq 0\}$. Then:

Corollary 1. *There is a notion of least model which is $T_P \uparrow \omega$.*

Proof. Immediate from Knaster-Tarksi theorem, Kleene's fixpoint Theorem, and Proposition 1. □

4.2 Operational Semantics

In this section we define the operational interpretation of labels: our approach is inspired by the methodology introduced for Constraint Logic Programming [4, 12, 13], namely we define the LVLP abstract state machine based on that suggested by Colmerauer for Prolog III. It resembles a push-down automaton [11] whose stack is updated whenever a program rule is applied. We define a labelled-machine state σ as the triplet:

$$\sigma = \langle t_0\ t_1 ... t_n, W, \Lambda\rangle$$

where

- $t_0\ t_1 ... t_n$ is the list of terms (goals)
- W is the current list of variables bindings
- Λ is the set of the current labelling associations on W.

An inference step for the machine consists of making a transition from the state σ to a state σ' by applying a program rule. Considering the rule ($m \geq 0$)

$$s_0 \leftarrow \Lambda', s_1, s_2, \dots, s_m$$

where Λ' is the set of the labellings of the rule, if the following conditions hold:

- $\exists\, mgu\, \theta$ such that $\theta(t_0) = \theta(s_0)$ and

– $\widetilde{f}_{\theta L}(\Lambda, \theta(\Lambda')) \neq \emptyset$

the rule is applied (fail otherwise). The function $\widetilde{f}_{\theta L}$ is a generalisation of the combining function f_L in that takes as arguments two vectors of labelling associations. If x_i occurs both in Λ and $\theta(\Lambda')$ with labels ℓ and ℓ', $\ell'' = f_L(\ell, \ell')$ is first calculated, then, if $f_C(\theta(x_i), \ell'') = true$ the labelling association $\langle x_i, \ell'' \rangle$ is returned. The new state becomes:

$$\sigma' = \langle \theta(s_1, \dots, s_m, t_1, \dots, t_n), W' = W \circ \theta, \Lambda'' = \widetilde{f}_{\theta L}(\Lambda, \theta(\Lambda')) \rangle$$

where $\circ$ applies the classical composition of substitutions.

A solution is found when a final state is reached. The final state has the form:

$$\sigma_f = (\epsilon, W_f, \Lambda_f)$$

where ϵ is the empty sequence, W_f is the final list of variables and bindings, Λ_f is the corresponding labelling associations set. A sequence of applications of inference steps is said a *derivation*. A derivation is *successful* if it ends in a final state, *failing* if it ends in a non-final state where no inference step is possible.

Proposition 2. *Let $p(\widetilde{x})$ be an atom, v a valuation on $\mathscr{H}$ such that $v(\widetilde{x}) = \widetilde{t}$ where $\widetilde{t}$ are ground terms, and $\widetilde{\ell}$ a list of labels. Then there is a successful derivation for $\langle p(\widetilde{t}), v, \langle v(\widetilde{x}), \widetilde{\ell} \rangle \rangle$ iff $\langle p(\widetilde{t}), \widetilde{\ell} \rangle \in T_P \uparrow \omega$.*

Proof (In the following we provide a proof sketch, deferring the full proof to a future paper for the sake of brevity).
($\rightarrow$). We prove the proposition by induction on the length k of the derivation. If $k = 0$ the result holds trivially.

For the inductive case, let us suppose that there is a successful derivation for $\langle p(\widetilde{t}), v, \langle v(\widetilde{x}), \widetilde{\ell} \rangle \rangle$ of $k + 1$ steps. Let us focus on the first step: there is a rule r: $s_0 \leftarrow \Lambda', s_1, s_2, \dots, s_m$ such that $\theta(p(\widetilde{t})) = \theta(s_0)$ leading to the new state $\sigma = \langle \theta(s_1, s_2, \dots, s_m), v \circ \theta, \Lambda'' = \widetilde{f}_{\theta L}(\Lambda, \theta(\Lambda') \rangle$, where $f_C(\Lambda'') = \mathsf{true}$, that admits a successful derivation of k steps.

Consider now the states $\sigma_1, \dots, \sigma_m$ defined as $\sigma_i = \langle \theta(s_i), v \circ \theta, \Lambda''_i \rangle$ where Λ''_i is the restriction of Λ'' to the variables in s_i. Since σ admits a successful derivation of $k+1$ steps, each σ_i should admit a successful derivation of at most k steps.

If for all $i \in \{1, \dots, m\}$, $\theta(s_i)$ is ground, then, by inductive hypothesis we have that $\langle \theta(s_i), \ell_i \rangle \in T_P \uparrow \omega$ where $\ell_i = \pi_2(\Lambda''_i)$, and hence that there are h_is such that $\langle \theta(s_i), \ell_i \rangle \in T_P \uparrow h_i$. Since T_p is monotonic, all of them belong to $T_P \uparrow h$ where $h = \max_{i=1,\dots,m} h_i$. Then, by applying T_P considering the rule r, since we already know that $\Lambda'' = \widetilde{f}_{\theta L}(\Lambda, \theta(\Lambda'))$, and $f_C(\Lambda'') = \mathsf{true}$, we have that $\langle p(\widetilde{t}), \widetilde{\ell} \rangle \in T_P \uparrow h + 1$, hence to $T_P \uparrow \omega$.

If for some i, $\theta(s_i)$ is not ground, the above property holds for any ground instantiation of the remaining variables and again the results follows.

($\leftarrow$). Now, let us analyse the converse direction. If $\langle p(\widetilde{t}), \widetilde{\ell} \rangle \in T_P \uparrow \omega$ this means that there is a $k \geq 0$ such that $\langle p(\widetilde{t}), \widetilde{\ell} \rangle \in T_P \uparrow k$.

Let us prove by induction on k. Again, if $k = 0$ the result holds trivially. Let us suppose now that $\langle p(\widetilde{t}), \widetilde{\ell}\rangle \in T_P \uparrow k+1$. This means (by definition of T_P) that there is a rule r: $s_0 \leftarrow \Lambda', s_1, s_2, \ldots, s_m$ such that $s_0 = p(\widetilde{x})$ and there is a valuation u on $\mathscr{H}$ such that $u(s_0) = p(\widetilde{t})$ and that, in particular, $\langle s_1, \ell_1\rangle, \ldots, \langle s_m, \ell_m\rangle \in T_P \uparrow k$ (and f_L can be computed and f_C is true on these arguments). By inductive hypothesis, for $i \in \{1, \ldots, m\}$ there is a derivation, say, of h_i steps for $\sigma_i = \langle u(s_i), v \circ u, \langle u(s_i), \ell_i\rangle\rangle$

Since f_C is true on such arguments and f_L can be computed, the same holds for T_P: so, we have a derivation of $\sum_{i=1}^{m} h_i + 1$ steps for $\langle p(\widetilde{t}), v \circ u \circ \theta, \langle \widetilde{t}, \widetilde{\ell}\rangle\rangle$.

This completes the proof of the inductive step. □

5 Conclusions & Future Work

The primary result of this paper is the definition of the LVLP theoretical framework, where different domain-specific computational models can be expressed via *labelled variables*, capturing suitably-tailored labelled models. The framework is aimed at extending LP to face the challenges of today's pervasive systems, by providing the models and technologies required to effectively support distributed situated intelligence, while maintaining the features of a declarative program-

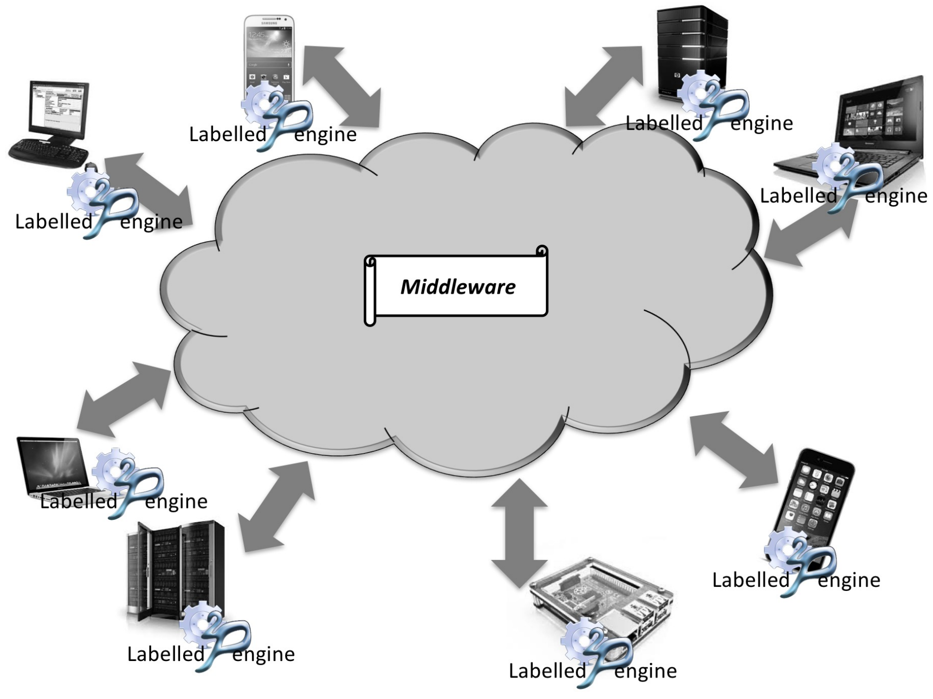

Fig. 1. Envisioning tuProlog / LVLP middleware for distributed (labelled) LP

ming paradigm. We also present the fixpoint and operational semantics, discuss correctness, completeness, and equivalence.

Beside completing our LVLP prototype [3], currently implemented over tuProlog [5], a focal point of our forthcoming activity is the design and implementation of a full-fledged *middleware* for tuProlog and LVLP, which could be exploited to test the effectiveness of distributed LP (and its extensions) in real-world pervasive intelligence scenarios—a sketch of the forthcoming middleware is reported in Fig. 1.

Future work will be devoted to a more complete proof of the result sketched in Subsection 4.1 as well as to the deeper exploration and better understanding of the consequences of applying labels to formulas, as suggested by Gabbay [8]. Other research lines will possibly include the further development of the Labelled tuProlog [5] prototype, as well as the application of the LVLP framework to different scenarios and approaches—such as probabilistic LP [17], the many CLP approaches [4], distributed ASP reasoning [7], and action languages [6].

Acknowledgements. Agostino Dovier is partially supported by INdAM-GNCS 2015 and 2016 projects.

References

1. Alsinet, T., Chesñevar, C.I., Godo, L., Simari, G.R.: A logic programming framework for possibilistic argumentation: Formalization and logical properties. Fuzzy Sets and Systems 159(10), 1208–1228 (May 2008), `http://dx.doi.org/10.1016/j.fss.2007.12.013`
2. Barany, V., ten Cate, B., Kimelfeld, B., Olteanu, D., Vagena, Z.: Declarative probabilistic programming with Datalog. In: Martens, W., Zeume, T. (eds.) 19th International Conference on Database Theory (ICDT 2016). LIPIcs, vol. 48, pp. 7:1–7:19. Dagstuhl, Germany (2016), `http://dx.doi.org/10.4230/LIPIcs.ICDT.2016.7`
3. Calegari, R., Denti, E., Omicini, A.: Labelled variables in logic programming: A first prototype in tuProlog. In: Bellodi, E., Bonfietti, A. (eds.) AI*IA 2015 DC Proceedings. CEUR Workshop Proceedings, vol. 1485, pp. 25–30. AI*IA, Ferrara, Italy (23–24 Sep 2015), `http://ceur-ws.org/Vol-1485/proceedings.pdf#page=30`
4. Cohen, J.: Constraint Logic Programming languages. Communications of the ACM 33(7), 52–68 (Jul 1990), `http://dx.doi.org/10.1145/79204.79209`
5. Denti, E., Omicini, A., Ricci, A.: Multi-paradigm Java-Prolog integration in tuProlog. Science of Computer Programming 57(2), 217–250 (Aug 2005), `http://dx.doi.org/10.1016/j.scico.2005.02.001`
6. Dovier, A., Formisano, A., Pontelli, E.: Autonomous agents coordination: Action languages meet CLP(FD) and Linda. Theory and Practice of Logic Programming 13(2), 149–173 (Sep 2013), `http://dx.doi.org/10.1017/S1471068411000615`
7. Dovier, A., Pontelli, E.: Present and future challenges for ASP systems. In: Erdem, E., Lin, F., Schaub, T. (eds.) Logic Programming and Nonmonotonic Reasoning. 10th International Conference, LPNMR 2009, Potsdam, Germany, September 14-18, 2009. Proceedings, pp. 622–624. Springer (2009), `http://dx.doi.org/10.1007/978-3-642-04238-6_70`

8. Gabbay, D.M.: Labelled Deductive Systems, Volume 1. Clarendon Press, Oxford Logic Guides 33 (Sep 1996)
9. Hofstedt, P.: Multiparadigm Constraint Programming Languages. Cognitive Technologies, Springer (2011), `http://dx.doi.org/10.1007/978-3-642-17330-1`
10. Holzbaur, C.: Metastructures vs. attributed variables in the context of extensible unification. In: Bruynooghe, M., Wirsing, M. (eds.) Programming Language Implementation and Logic Programming, Lecture Notes in Computer Science, vol. 631, pp. 260–268. Springer (1992), `http://dx.doi.org/10.1007/3-540-55844-6_141`, 4th International Symposium (PLILP'92) Leuven, Belgium, August 26–28, 1992 Proceedings
11. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation. Pearson / Addison-Wesley, 3rd edn. (2007)
12. Imbert, J.L., Cohen, J., Weeger, M.D.: An algorithm for linear constraint solving: Its incorporation in a Prolog meta-interpreter for CLP. The Journal of Logic Programming 16(3), 235–253 (1993), `http://dx.doi.org/10.1016/0743-1066(93)90044-H`
13. Jaffar, J., Maher, M.J.: Constraint Logic Programming: A survey. The Journal of Logic Programming 19–20(Supplement 1), 503–581 (May–Jul 1994), `http://dx.doi.org/10.1016/0743-1066(94)90033-7`, Special Issue: Ten Years of Logic Programming
14. Mariani, S., Omicini, A.: Coordinating activities and change: An event-driven architecture for situated MAS. Engineering Applications of Artificial Intelligence 41, 298–309 (May 2015), `http://dx.doi.org/10.1016/j.engappai.2014.10.006`
15. Omicini, A., Zambonelli, F.: MAS as complex systems: A view on the role of declarative approaches. In: Leite, J.A., Omicini, A., Sterling, L., Torroni, P. (eds.) Declarative Agent Languages and Technologies, Lecture Notes in Computer Science, vol. 2990, pp. 1–17. Springer (May 2004), `http://dx.doi.org/10.1007/978-3-540-25932-9_1`, 1st International Workshop (DALT 2003), Melbourne, Australia, 15 Jul. 2003. Revised Selected and Invited Papers
16. Russo, A.: Generalising propositional modal logic using labelled deductive systems. In: Baader, F., Schulz, K.U. (eds.) Frontiers of Combining Systems, Applied Logic Series, vol. 3, pp. 57–73. Springer (1996), `http://dx.doi.org/10.1007/978-94-009-0349-4_2`
17. Skarlatidis, A., Artikis, A., Filippou, J., Paliouras, G.: A probabilistic logic programming event calculus. Theory and Practice of Logic Programming 15(Special Issue 02 (Probability, Logic and Learning)), 213–245 (3 2015), `http://dx.doi.org/10.1017/S1471068413000690`
18. WordNet: Home page. `http://wordnet.princeton.edu`