Gerald Plesniewicz*, Baurzhan Karabekov, and Dmitry Masherov

A logical specification language for ontologies in the system “Binary Model of Knowledge”

https://doi.org/10.1515/eng-2018-0017
Received June 28, 2017; accepted November 10, 2017

Abstract: We describe the language LLS-1 for logical specifying ontologies written in “Binary Model of Knowledge” (BMK). BMK is the system intended for specifying heavy-weight ontologies by means of conceptual-type languages. The language LLS-1 has a user-friendly syntax in style of the Manchester syntax for OWL. We also determine a deduction method for LLS-1.

Keywords: knowledge bases, ontologies, specification languages, conceptual-type languages, logical specification languages, deduction

1 Introduction

Ontologies are seen as the key technology used to describe the semantics of information. An ontology is a formal description (in terms of concepts, entities, their properties and relationships) of knowledge for solving a given class of problems [1, 5].

“Binary Model of Knowledge” (BMK) is a platform intended for specifying ontologies by means of conceptual-type languages and for manipulating ontologies, in particular, for deduction and query answering over ontologies [3]. BMK is under development in the National Research University “MPEI”, Applied Mathematics Department (Moscow, Russia) and in the Institute of Information and Computing Technologies (Almaty, Kazakhstan). The architecture of BMK includes the ontology editor and knowledge management system Protégé 5.0 with plugins to it, that defines the functionality of BMK.

Semantics of BMK languages is defined using formal concepts. A formal concept is constructed of names which are members of the data type Name (subtype of the standard type String), and has the following components:
- name C;
- universe U^C – the set of all possible names for instances of the concept C;
- set I^C of all points of reference – they can be interpreted as worlds, states of affaire, contexts, timepoints, and so on;
- subset E^C,γ ⊆ U^C for each point γ ∈ I^C – the set of all names for instances of C at the point of reference γ;
- coreferentiality relation ~C,γ ⊆ E^C,γ × E^C,δ for each point γ ∈ I^C – two names are coreferential if they denote the same object of modelled problem domain.

The pair (E^C,γ × E^C,δ) is the extension Ext^C,γ of the concept C at the point of reference γ. The family Ext^C,γ = {Ext^C,γ | γ ∈ I^C} is the full extension of C.

It is clear that coreferentiality is an equivalence relation, and the quotient E^C,γ/~E^C,γ is in one-to-one correspondence with the set of the objects – instances of C. We will assume that every equivalence class contains, as a representative, some surrogate (or OID – object identifier). So, there is in BMK the data type Surr = {#1, #2, #3,…} of surrogates.

A concept system is a finite set of formal concepts with same points of reference. An ontology is a specification of a concept system.

Let O be any ontology and O_str, O_log, O_tran be its parts such that
- statements of O_str specify the universes of the concepts i.e. factually the concepts structure;
- statements of O_log specifies the extensions of the concepts uniformly and independently of points of reference;
- statements of O_tran specify the changes E^C,γ ∆ E^C,δ = (E^C,γ \ E^C,δ) ∪ (E^C,δ \ E^C,γ) for (γ, δ) ∈ R where R is some binary relation on I^C.

For writing O_str, O_log and O_tran languages for structural, logical and transitional specification are used (respectively).

In BMK, there are several specification languages. In particular, there is the language LSS for structural specification and the language LLS-1 for logical specification. The
main goal of the present paper is to describe the language LLS-1 and to present a method deduction for this language.

There are two types of concepts that can be specified by means of LSS – classes and links (binary relations). For example, consider the following LSS statements determining the universes of the classes and link

Employee[Name:String,Position:String,
Works_at:Department],
Project[ID:String,Budget:Integer,
Supervisor:Employee,
Team:Employee(*)],
(1.1)
(Employee Works_for Project)[Starts:Date],
Department[Name:String,From:University].

The statement (1.1) determines the universe $U_{Project}$ as the set of all tuples $[ID: x, Budget: y, Supervisor: s, Team: \{s_1, s_2, \ldots, s_n\}]$ united with the set $\text{Surr} = \{#1, #2, \ldots\}$ of surrogates (OIDs – object identifiers) and the set $\text{Name}$ of individual object names. Here $x \in \text{String}, y \in \text{Integer}$ and $s, s_1, s_2, \ldots, s_n \in \text{Surr} (n = 0, 1, 2, \ldots)$.

We can define new concepts using constrains on attributes. For example, the following term defines the class of all employees from the department “Applied mathematics” that work for a project with the budget less than 200000 dollars:

Employee(Works_at='applied mathematics';

This term can be considered as composite name of that class.

We use statements of logical specification languages for defining concept extensions. Also we can specify directly some tuples as instances or counter-instances of a given concept; for this purpose the tabular representation is used in BMK. For example, the head of tables for the link $\text{Works_for}$ has the standard attributes $\text{Por}$ (point of reference), $\text{Surr}$ (surrogate), $\text{Coref}$ (list of coreferential names), $\text{Sign}$ (signs “+” and “–” for truth and false) and the specific attributes $\text{Employee}$, $\text{Project}$ and $\text{Starts}$.

If the table for $\text{Works_for}$ contains the row

(1.2) then at point of reference $\text{por2}$ we have that

\[ \#210 - \text{Employee: #17, Project: #107, Starts: ?} \]

and these two coreferential names don’t belong to the extension $\text{E}_{\text{Works_for}}$ of the link $\text{Works_for}$. Informally (1.2) and (1.3) means that at the point of reference $\text{por2}$ the employee with the surrogate #17 don’t work for the project with the surrogate.

Remark. We aimed at such definition of syntax for LLS-1 that this syntax would be close to the Manchester syntax for OWL. (The “Manchester syntax” is a compact, user-friendly syntax with a style close to frame languages.)

In Section 2 we will take the known problem “Steamroller” and will write it as an ontology in the language LLS-1. In Section 3 we will describe exactly syntax and semantics of LLS-1. In Section 4 the deduction method for LLS-1 ontologies is presented.

## 2 Problem “Steamroller”

In 1978, Leonard Schubert set up the following problem for logical proving [6]. The name “Steamroller” of the problem is caused by the fact that in spite of its apparent simplicity, it turned out to be too hard for existing resolution-based theorem provers.

(a) Wolves, foxes, birds, caterpillars, and snails are animals.
(b) There are some of each of them.
(c) Grains are plants.
(d) There are some grains.
(e) Every animal either likes to eat all plants or all animals much smaller than itself that like to eat some plants.
(f) Caterpillars and snails are much smaller than birds, which are much smaller than foxes, which in turn are much smaller than wolves.
(g) Wolves do not like to eat foxes or grains, while birds like to eat caterpillars but not snails.
(h) Caterpillars and snails like to eat some grains. Therefore,
(i) There is an animal that likes to eat a grain-eating animal.

The problem “Steamroller” is to prove automatically that (a), (b),..., (g) logically imply (i).

Beginning to write “Steamroller” in the language LSL-1, we introduce the following concept names: Animal, Plant, Wolf, Fox, Bird, Caterpillar, Snail, Grain (for classes) and LikesToEat, MuchSmaller (for binary relations). All these classes have no structure, and so we do not write for them any LSL-1 statements besides the following two SSL statements defining the binary relations:

(Animal LikesToEat Animal | Plant).
(Animal Smaller Animal).

We formalize the sentences (a) – (i) by the following LSL-1 statements.

(a): Wolf ISA Animal. (2.1)
Fox ISA Animal. (2.2)
Bird ISA Animal. (2.3)
Caterpillar ISA Animal. (2.4)
Snail ISA Animal.

(2b): EXIST Wolf.
EXIST Fox.
EXIST Bird.
EXIST Caterpillar.
EXIST Snail.
(c): Grain ISA Plant.
(d): EXIST Grain.
(e): EACH Animal:S LikesToEat SOME Plant OR LikesToEat EACH Animal THAT Smaller S AND LikesToEat SOME Plant.
(f): EACH Caterpillar Smaller EACH Bird.
EACH Bird Smaller EACH Fox.
EACH Fox Smaller EACH Wolf.
(g): EACH Wolf NOT LikesToEat EACH (Caterpillar OR Plant).
(h): (Caterpillar OR Snail)
LikesToEat SOME Plant.
(i): EXIST Animal THAT LikesToEat SOME Animal THAT LikesToEat SOME Grain.

Here, in (2.13), the symbol S indicates on sameness of concept instances: S denotes the same instance of Animal.

We consider the set of statements [(2.1), (2.2), ..., (2.19)] as an example of an ontology written in the language LLS-1.

The solution of the problem “Steamroller” consists in the proof that (2.20) logically follows from [(2.1), (2.2), ..., (2.19)]. This is equivalent that the ontology [(2.1), (2.2), ..., (2.19), (2.20)] is inconsistent, where (2.20) denotes negation of the statement (2.20) i.e.

NULL Animal THAT LikesToEat SOME Animal THAT LikesToEat SOME Grain.

3 Syntax and semantic of the language LLS-1

We introduce three types of terms in LLS-1: C-terms, L-terms and P-terms.

C-terms denote classes of objects. They are interpreted as subsets of SurR of surrogates. The examples of C-terms are

Wolf, Animal, Animal:S, NOT Wolf, Wolf OR Fox,

L-terms denote binary relations (links). They are interpreted as subsets of SurR X SurR. The examples of L-terms are

MuchSmaller, LikesToEat, NOT LikesToEat, MuchSmaller AND Not LikesToEat.

P-terms denote unary predicates defined on the set SurR. Here are the examples of P-terms:

Smaller, Smaller EACH (Wolf Fox), LikesToEat SOME Plant.

Describing syntax and semantics of the language LLS-1, we will use the following syntactic classes:

• c for individual objects;
• N for concept names;
• C, D, Ci for C-terms (i = 1, 2, 3, ...);
• L, M, Li for L-terms;
• P, Q, Pi for P-terms;
• S, T for sentences;
• I for sameness indicators S, S1, S2, ..., .

By “a” denote the value of an expression α of LSL-1 under an arbitrary fixed interpretation “ “.

SYNTAX of C-terms:

C1 := NOT C1 | (C1 AND C1) | (C1 OR C1) | N;
C := C1 | C1 THAT P | THOSE P | Ci: I

SEMANTICS of C-terms:

“C AND D” = “C” \ “D”, “C OR D” = “C” ∪ “D”,
“C THAT P” = {x ∈ “C” | “P”(x)}, “THOSE P” = {x | “P”(x)}.

SYNTAX of L-terms:

L1 := NOT L1 | L1 AND L1 | L1 OR L1 | N;
L := L1 | INV(L1);

SEMANTICS of L-terms:

“L AND M” = “L” ∩ “M”, “C OR D” = “C” ∪ “D”,
“INV(M)” = {(y, x) | “M”(y)}.

SYNTAX of P-terms:

P1 := NOT P1 | (P1 AND P1) | P1 OR P1 | N;
P := P1 | L SOME C | L EACH C | L ONLY C | (L I) | (L c).

SEMANTICS of P-terms:

“(x P)” <-> “P(x)”, “(x NOT Q)” <-> “Q(x),
“(x P AND Q)” <-> “P(x) \ “Q”(x),
“(x P OR Q)” <-> “P”(x) \ “Q”(x),
“(x L SOME C)” <-> (∃y“C”) (xy) “L”
“(x L EACH C)” <-> (∃y“C”) (xy) “L”,
“(x L ONLY C)” <-> ∀y (xy) “L” \ “C”,
“(x (L V))” <-> (xy) “L”,
“(x (L c))” <-> (xy) “c”, “L”.

SYNTAX of statements:

S, T := NOT T | S AND T | S OR T | S IMP T |
S = T | EXIST C | EXIST L | NULL C |
NULL L | C ISA D | L ISA M | C = D |
\[ L = M \mid C = \text{NOT } D \mid L = \text{NOT } M \]
\[ \text{EACH } C \, P \mid \text{SOME } C \, P. \]

**SEMANTICS of statements:**

\[ \text{"NOT } T \text{" } \iff \text{"} T \text{"}, \text{"} S \text{ AND } T \text{" } \iff \text{"} S \text{ } \land \text{ } T \text{"}, \]
\[ \text{"} S \text{ OR } T \text{" } \iff \text{"} S \text{ } \lor \text{ } T \text{"}, \text{"} S \text{ IMPL } T \text{" } \iff \text{"} S \text{ } \rightarrow \text{ } T \text{"}, \]
\[ \text{"} S \text{ = } T \text{" } \iff \text{"} S \text{ } = \text{ } T \text{"}, \text{"} \text{EXIST } C \text{" } \iff \text{"} C \text{" } \neq \emptyset, \]
\[ \text{"} \text{NOT } L \text{" } \iff \text{"} L \text{" } \neq \emptyset, \text{"} \text{NULL } C \text{" } \iff \text{"} C \text{" } = \emptyset, \]
\[ \text{"} \text{NULL } L \text{" } \iff \text{"} L \text{" } = \emptyset, \text{"} C \text{ ISA } D \text{" } \iff \text{"} C \text{ } \subseteq \text{ } D \text{"}, \]
\[ \text{"} L \text{ ISA } M \text{" } \iff \text{"} L \text{ } \subseteq \text{ } M \text{"}, \text{"} C \text{ } \equiv \text{ } D \text{" } \iff \text{"} C \text{ } = \text{ } D \text{"}, \]
\[ \text{"} C \text{ } = \text{ } \text{ NOT } D \text{" } \iff \text{"} C \text{" } = \text{ } \text{ NOT } D \text{"}, \]
\[ \text{"} L \text{ = } \text{ NOT } M \text{" } \iff \text{"} L \text{" } = \text{ Surr } X \text{ Surr } \text{\"} \text{M \"}, \]
\[ \text{"} \text{EACH } C \, P \text{" } \iff \text{"} (\forall x \in \text{"} C \text{"} ) \text{"} P \text{"} (x), \]
\[ \text{"} \text{SOME } C \, P \text{" } \iff \text{"} (\exists y \text{"} C \text{"} ) \text{"} P \text{"} (x). \]

We consider the following expressions as **patterns** for simple statements that consist of no more than three simple names (i.e. members of the data type `Name`):

\begin{align*}
\text{EXIST } C, \text{ NULL } C, \text{ EXIST } L, \text{ NULL } L, C \text{ ISA } D, \\
C = D, C = \text{ NOT } D, C = \text{ D AND } E, C = D \text{ OR } E, \\
C = D \text{ THAT } P, C = \text{ THOSE } P, L = M, L = \text{ NOT } M, \\
L = M \text{ AND } N, L = M \text{ OR } N, L = \text{ INV } (M), P = Q, \\
P = \text{ NOT } P, Q = \text{ Q AND } Q', P = Q \text{ OR } Q', P = L \text{ SOME } C, \\
P = L \text{ EACH } C, P = L \text{ ONLY } C, P = (L) V, P = (L) c, \\
S = T, S = \text{ NOT } T, S = T \text{ AND } T', S = T \text{ OR } T', \\
S = T \text{ IMPL } T', S = (T = T'), S = \text{ EXIST } C, \\
S = \text{ NULL } C, S = \text{ EXIST } L, S = \text{ NULL } L, S = C \text{ ISA } D, \\
S = (C = D), S = (L = M), S = (P = Q). 
\end{align*}

Particular simple statements are obtained by replacing the parameters \( C, D, E, L, M, N, P, Q, Q', S, T, T' \) with simple names from `Name`. The names chosen for the parameters \( C, D \) and \( E \) are interpreted as classes; the names chosen for the parameters \( L, M \) and \( N \) are interpreted as binary relations; the names chosen for the parameters \( P, Q \) and \( Q' \) are interpreted as classes; the names chosen for the parameters \( S, T \) and \( T' \) are interpreted as classes.

As every logic, LLS-1 have the relation “\(|\text{=}\)" of **logical consequence**. Let \( O \) be an ontology in LLS-1 (i.e. a finite set of statements from LLS-1) and \( \phi \) is a LLS-1 statement. Then \( \phi \) **logically follows from** \( O \) if and only if there is no interpretation such that all statements from \( O \) are true and the statement \( \phi \) is false.

A **fact** for an ontology \( O \) is an expression of one of the forms


where \( u, v \in \text{ Surr } \cup \text{ Name } \cup \text{ Var. } \) Facts with signs “+” and “−” are called **positive** and **negative** (correspondingly). For any interpretation “\( |\)" we define:

\[ + : u : C \iff u \in \text{ "} C \text{"}, \]
\[ - : u : C \iff u \notin \text{ "} C \text{"}, \]
\[ + : u : L : v \iff (u, v) \in \text{ "} L \text{"}, \]
\[ - : u : L : v \iff (u, v) \notin \text{ "} L \text{"}, \]

**A fact base** for \( O \) is a finite set \( F \) of facts for \( O \). A **knowledge base** is a set \( K = O \cup F \) where \( F \) is a fact base for \( O \).

The relation “\(|\text{=}\)" of logical consequence is extended to fact bases and knowledge bases.

### 3.1 Equivalence of ontologies

Two ontologies \( O \) and \( O' \) are **equivalent** if for any interpretation, all statements from \( O \) are true if and only if all statements from \( O' \) are true.

It is easy to see that each LLS-1 ontology is equivalent to an ontology consisting of only simple statements. Consider an example.

**Example 1.** Take the statement.

\[ \text{NULL Animal THAT LikesToEat SOME Grain.} \]

Let us define:

\[ P1 = \text{LikesToEat SOME Grain}, \]
\[ C1 = \text{Animal THAT P1}, \]
\[ P2 = \text{LikesToEat SOME C1}, \]
\[ C2 = \text{Animal THAT P2}, \]
\[ \text{NULL C2}. \]

Clearly, two ontologies \( \{ (3.20) \} \) and \( \{ (3.1), (3.2), (3.3), (3.4), (3.5) \} \) are equivalent.

In general, let \( O \) be an LSS-1 ontology, and let us transform its statements to simple statements as we did it for statement (3.20). By \( O^* \) denote the resulting ontology.

**Example 2.** Let \( O-St \) be the ontology for the Steamroller problem, \( O-St = \{ (2.1), (2.2), \ldots, (2.19), (2.21) \} \). Then we obtain

\[ O-St^* = \{ (2.1): \text{Wolf ISA Animal}, \]
\[ (2.2): \text{Fox ISA Animal}, \]
\[ (2.3): \text{Bird ISA Animal}, \]
\[ (2.4): \text{Caterpillar ISA Animal}, \]
\[ (2.5): \text{Snail ISA Animal}, \]
\[ (2.6): \text{EXIST Wolf}, \]
\[ (2.7): \text{EXIST Fox}, \]
\[ (2.8): \text{EXIST Bird}, \]
\[ (2.9): \text{EXIST Caterpillar}, \]
\[ (2.10): \text{EXIST Snail}, \]
\[ (2.11): \text{Grain ISA Plant}, \]
\[ (2.12): \text{EXIST Grain}, \]
\[ (2.13): P1 = \text{LikesToEat SOME Plant}, \]
\[ P2 = \text{Smaller X}, \]
\[ P3 = P1 \text{ AND } P2, \]
\[ C1 = \text{Animal THAT P3}, \]
\[ P4 = \text{LikesToEat EACH C1}, \]
\[ P5 = \text{LikesToEat SOME Plant}, \]
\[ P6 = P4 \text{ OR } P5, \]
\[ \text{EACH Animal:X P6}, \]
\[ (2.14): P7 = \text{Smaller EACH Bird}, \]
\[ \text{EACH Caterpillar P7}. \]
4 Deduction for the language LLS-1

We define a deduction method based on applying productions to fact bases of LLS-1 ontologies.

A production has the form \( \alpha \Rightarrow \beta \) where \( \alpha \) and \( \beta \) are conjunctions of facts (possibly, with variables). Below is the list of patterns with associated productions for the language LLS-1.

We associate productions with each pattern. Below is the list of these productions. It easy to prove that the productions are sound. For example, take the pattern 13: \( C = D \) or \( E \). Clearly, for any surrogate \( s : (i) \) if \( s \in \{ C \} \) and \( s \notin \{ D, E \} \) then \( s \in \{ E \} \); (ii) if \( s \in \{ C \} \) and \( s \notin \{ D, E \} \) then \( s \in \{ D \} \); (iii) if \( s \notin \{ C \} \) then \( s \notin \{ D, E \} \); (iv) if \( s \notin \{ D \} \) and \( s \notin \{ E \} \) then \( s \notin \{ C \} \); (v) if \( s \in \{ D \} \) then \( s \in \{ E \} \). Thus, all productions for the pattern 13 are sound.

1. Pattern \( \text{EXIST } C \).
   
   \( \Rightarrow +s_2: C \)

   Here \( s \) is new Skolem’s constant.

2. Pattern \( \text{NULL } C \).
   
   \( \Rightarrow +v_2: C \)

   Here \( v \) is new variable.

3. Pattern \( \text{EXIST } L \).
   
   \( \Rightarrow +s_3: L +s(j+1) \)

   Here \( s \) and \( j+1 \) are new Skolem’s constants.

4. Pattern \( \text{NULL } L \).
   
   \( \Rightarrow +v_3: L +v(j+1) \)

   Here \( v \) and \( v(j+1) \) are new variables.

5. Pattern \( C \) \( ISA \) \( D \).
   
   \( +x: C \Rightarrow +x: D, -x: D \Rightarrow -x: C \)

6. Pattern \( L \) \( ISA \) \( M \).
   
   \( +x: C \Rightarrow +x: D, -x: D \Rightarrow -x: C \)

7. Pattern \( C \) \( ISA \) \( D \).
   
   \( +x: C \Rightarrow +x: D, -x: D \Rightarrow -x: C \)

8. Pattern \( L \) \( ISA \) \( M \).
   
   \( +x: C \Rightarrow +x: D, -x: D \Rightarrow -x: C \)

9. Pattern \( \text{EACH } C \).
   
   \( +x: C \Rightarrow +(x P), -(x P) \Rightarrow -x: C \)

10. Pattern \( \text{SOME } C \).
    
    \( +x: C \Rightarrow +(x P), -(x P) \Rightarrow -x: C \)

11. Pattern \( C \) \( ISA \) \( D \).
    
    \( +x: C \Rightarrow +x: D, -x: D \Rightarrow -x: C \)

12. Pattern \( L \) \( ISA \) \( M \).
    
    \( +x: C \Rightarrow +x: D, -x: D \Rightarrow -x: C \)

13. Pattern \( L \) \( ISA \) \( M \).
    
    \( +x: C \Rightarrow +x: D, -x: D \Rightarrow -x: C \)

14. Pattern \( C \) \( ISA \) \( D \).
    
    \( +x: C \Rightarrow +x: D, -x: D \Rightarrow -x: C \)

15. Pattern \( C \) \( ISA \) \( D \).
    
    \( +x: C \Rightarrow +x: D, -x: D \Rightarrow -x: C \)

16. Pattern \( L \) \( ISA \) \( M \).
    
    \( +x: C \Rightarrow +x: D, -x: D \Rightarrow -x: C \)

17. Pattern \( L \) \( ISA \) \( M \).
    
    \( +x: C \Rightarrow +x: D, -x: D \Rightarrow -x: C \)

18. Pattern \( C \) \( ISA \) \( D \).
    
    \( +x: C \Rightarrow +x: D, -x: D \Rightarrow -x: C \)
23. Pattern \( P_j \mid L \) SOME \( C \).
\[ + (X \ P_j) \Rightarrow + X \ L : \ sk \ + \ sk : \ C, \]
\[ - (X \ P_j) ; + Y : C \Rightarrow - X \ L : Y, \]
\[ - (X \ P_j) ; + X \ L : Y \Rightarrow - Y : C. \]

24. Pattern \( P_j \mid L \) EACH \( C \).
\[ + (X \ P_j) ; + Y : C \Rightarrow + X \ L : Y, \]
\[ - (X \ P_j) ; - X \ L : Y \Rightarrow - Y : C, \]
\[ + (X \ P_j) \Rightarrow - X \ L : sk \ + \ sk : C. \]

21. Pattern \( P_j \mid L \) ONLY \( C \).
\[ + (X \ P_j) ; + X \ L : Y \Rightarrow + Y : C, + (X \ P_j) ; \]
\[ - X : L : Y \Rightarrow - Y : C, \]
\[ - (X \ P_j) \Rightarrow + X : L : sk \ - sk : C. \]

22. Pattern \( L_j \mid \) INV \( L \).
\[ + X : L : Y \Rightarrow + Y : L : X, \]
\[ - X : L : Y \Rightarrow - Y : L : X, \]
\[ + (X \ Q) \Rightarrow + (X \ P), \]
\[ - (X \ P) \Rightarrow + (X \ R). \]

23. Pattern \( R \mid P \) AND \( Q \).
\[ + (X \ R) \Rightarrow + (X \ P) ; + (X \ Q), \]
\[ - (X \ R) ; + (X \ P) \Rightarrow - (X \ Q), \]
\[ - (X \ R) ; + (X \ Q) \Rightarrow + (X \ P), \]
\[ + (X \ R) \Rightarrow - (X \ P), \]
\[ - (X \ Q) \Rightarrow + (X \ R). \]

24. Pattern \( R \mid P \) OR \( Q \).
\[ -(X \ R) \Rightarrow -(X \ P) ; -(X \ Q), \]
\[ + (X \ R) ; - (X \ P) \Rightarrow + (X \ Q), \]
\[ + (X \ R) ; -(X \ Q) \Rightarrow + (X \ P), \]
\[ + (X \ R) \Rightarrow + (X \ P), \]
\[ -(X \ Q) \Rightarrow -(X \ R). \]

25. Pattern \( L \mid \) INV \( L \).
\[ + (X \ P) \Rightarrow + X : L : V. \]

Here \( V \) is a variable.

26. Pattern \( P \mid L \) c.
\[ + (X \ P) \Rightarrow + X : L : c. \]

27. Pattern \( S \mid T \).
\[ + : S \Rightarrow + : T, - : T \Rightarrow - : S, + : T \Rightarrow + : S, \]
\[ - : S \Rightarrow - : T. \]

28. Pattern \( S \mid T \) AND \( T_1 \).
\[ + : S \Rightarrow + : T, + : T_1, - : S ; + : T \Rightarrow - : T_1, \]
\[ - : S ; + : T_1 \Rightarrow - : T, - : T \Rightarrow - : S, - : T_1 \Rightarrow - : S. \]

29. Pattern \( S \mid T \) OR \( T_1 \).
\[ - : S \Rightarrow - : T, - : T_1, + : S ; - : T \Rightarrow + : T_1, \]
\[ + : S ; - : T_1 \Rightarrow + : T, + : T \Rightarrow + : S, + : T_1 \Rightarrow + : S. \]

30. Pattern \( S \mid T \) IMP \( T_1 \).
\[ - : S \Rightarrow + : T, - : T_1, + : S ; - : T \Rightarrow + : T_1, \]
\[ + : S ; - : T_1 \Rightarrow + : T, + : T \Rightarrow + : S, + : T_1 \Rightarrow + : S. \]

31. Pattern \( S \mid \) EXIST \( C \).
\[ + : S \Rightarrow + : C; \]
\[ - : S \Rightarrow - v j ; \]
\[ C, + : C \Rightarrow + : S, \]
\[ - v : C \Rightarrow - : S. \]

Here \( s_j \) is new Skolem’s constant, \( v_j \) is new variable, \( C \) is an object constant, \( v \) is a variable.

32. Pattern \( S \mid \) EXIST \( L \).
\[ + : S \Rightarrow + v k : L : \ s (k + 1), - : S \Rightarrow - v k : L : v (k + 1), \]
\[ + c : L : d \Rightarrow + : S, - u : L : v \Rightarrow - : S. \]

Here \( s \) and \( s (k + 1) \) are new Skolem’s constants, \( v_k \) and \( v (k + 1) \) are new variables, \( c \) and \( d \) are object constants, \( u \) and \( v \) are variables.

33. Pattern \( S \mid \) NULL \( C \).
\[ + : S \Rightarrow - v k : L : v (k + 1), - : S \Rightarrow + s k : L : s (k + 1), \]
\[ + u : L : v \Rightarrow + : S, - c : L : d \Rightarrow - : S. \]

Here \( s \) is new Skolem’s constant, \( v_k \) is new variable, \( C \) is an object constant, \( v \) is a variable.

34. Pattern \( S \mid \) NULL \( L \).
\[ + : S \Rightarrow - v k : L : v (k + 1), - : S \Rightarrow + s k : L : s (k + 1), \]
\[ + u : L : v \Rightarrow + : S, - c : L : d \Rightarrow - : S. \]

Here \( c \) and \( d \) are object constants, \( u \) and \( v \) are variables.

35. Pattern \( S \mid C \) ISA \( D \).
\[ + : S ; + s k : C \Rightarrow + x : D, + : S ; - x : D \Rightarrow - x : C, \]
\[ - : S \Rightarrow - c j : C ; + c j : D, - : v : C \Rightarrow + : S, \]
\[ + v : D \Rightarrow + : S, + c : C ; - c : D \Rightarrow - : S. \]

36. Pattern \( S \mid L \) ISA \( M \).
\[ + : S ; + x : L \Rightarrow + x : M, + : S ; - x : M \Rightarrow - x : L, \]
\[ - : S \Rightarrow - s j : L : c (j + 1), + c j : M : c (j + 1), \]
\[ - u : L : v \Rightarrow + : M, + u : M : v \Rightarrow + : S, \]
\[ + c : C ; - c : D \Rightarrow - : S. \]

37. Pattern \( S \mid \) EACH \( C \).
\[ + : S \Rightarrow + v j : C ; + (v j P), - : S \Rightarrow + c j : C ; + (+ c j P). \]

38. Pattern \( S \mid \) SOME \( C \).
\[ + : S \Rightarrow + c j : C ; + (+ c j P). \]

39. Pattern \( S \mid (C = D) \).
\[ + : S ; + x : L \Rightarrow + x : M, + : S ; - x : D \Rightarrow - x : C, \]
\[ - : S \Rightarrow - c j : C ; + c j : D, - : v : C \Rightarrow + : S, + v : D \Rightarrow + : S, \]
\[ + c : D ; - c : C \Rightarrow - : S. \]

40. Pattern \( S \mid (L = M) \).
\[ + : S ; + x : L \Rightarrow + x : M, + : S ; - x : L \Rightarrow - x : M, \]
\[ - : S \Rightarrow - c j : L : c (j + 1), + c j : M : c (j + 1), \]
\[ - u : L : v \Rightarrow + : S, + u : M : v \Rightarrow + : S, \]
\[ + c : C ; - c : D \Rightarrow - : S, + c : S ; + x : M \Rightarrow + x : L, \]
\[ + : S ; - x : L \Rightarrow - x : M, - u : M : v \Rightarrow + : S, \]
\[ - : S \Rightarrow - c j : M : c (j + 1), + c j : L : c (j + 1), \]
\[ + u : L : v \Rightarrow + : S, + c : C ; - c : D \Rightarrow - : S. \]

41. Pattern \( S \mid (P = Q) \).
\[ + : S ; + (P X) \Rightarrow + (Q X), \]
\[ + : S ; - (Q X) \Rightarrow - (P X), \]
\[ - : S \Rightarrow - (P c j), + (Q c j), \]
\[ - (P v) \Rightarrow + s, \]
\[ + (Q v) \Rightarrow + : S, + (Q c) \Rightarrow - (P c) \Rightarrow - : S. \]
and we obtain empty. At step 1 the productions (4.1) and (4.4) are applied, written in the j-th row of the table. The initial fact base $F_0$ is inconsistent, when substituting $v_1:= s_2$ and facts are translated to RDFS. It is sufficient to consider primitive statements. Take, for example, the statement $P_1 = \text{LikesToEat} \text{ SOME Plant}$. This statement can be unify with the pattern $17: P_j = L \text{ SOME C}$. Thus, we have the productions $+(X P_1) => +X: \text{LikesToEat: s2} ; +s_2: \text{Plant}$, $-(X P_1); +Y: \text{Plant} => -X: \text{LikesToEat: Y}$, $-(X P_1); +X: \text{LikesToEat: Y} => -Y: \text{Plant}$, $+X: \text{Bird} => +(X P_1)$, $-Y: \text{Plant}$ and facts are translated to PDFS as follows:

We have applied the deduction method to the ontology $O-St$ for the problem “Steamroller”. The deduction had 11 steps before receiving a contradiction.

### 4.1 About implementation of the deduction method for LLS-1

For an implementation of the deduction method for LLS-1, we used the languages RDF, RDFS and SPARQL [1].

In the system BMK, there is the translator of LSS ontologies and fact bases into RDF schemes and RDF graphs. For example, information contained in (1.2) and (1.3) is written in RDF (in Turtle notation) as follows

Table 1: Deduction for the ontology from Example 3

<table>
<thead>
<tr>
<th>Bird</th>
<th>Plant</th>
<th>LikeToEat</th>
<th>P1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>+s1</td>
<td>-v1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>+s1</td>
<td>-v1</td>
<td>+s1</td>
</tr>
<tr>
<td>3</td>
<td>+s1</td>
<td>+s2</td>
<td>+s1</td>
</tr>
</tbody>
</table>

We see that $F_3$ contains the facts $-v1: \text{Plant}$ and $+s2: \text{Plant}$ which give the contradiction $-v1: \text{Plant}$ and $+s2: \text{Plant}$ when substituting $v1:= s2$. Thus, the fact base is inconsistent.
(:P1 :sign :-).
(:LikesToEat :sign :+).
(:LikesToEat :first ?X).
(:LikesToEat :second ?Y).

5 Conclusion

We have described briefly the language LLS-1 for logical specification of ontologies in the system “Binary Model of Knowledge” (BMK). BMK is a platform intended for specifying ontologies by means of conceptual-type languages and for manipulating ontologies, in particular, for deduction and query answering over ontologies. We will implement BMK as the set of plugins to the system Protégé 5.0 [4].

Acknowledgement: This work was supported by Russian Foundation for Basic Research (project 14-07-0387) and Ministry of Education and Science of Kazakhstan (project 0115 RK 00532).

References