Bibliography

1. Bolshakov, I.A. Getting One's First Million…Collocations. In: A. Gelbukh (Ed.). Computational Linguistics and Intelligent Text Processing. Proc. 5th Int. Conf. on Computational Linguistics CICLing-2004, Seoul, Korea, February 2004. LNCS 2945, Springer, 2004, p. 229-242.
2. Bolshakov, I.A., A. Gelbukh. Paronyms for Accelerated Correction of Semantic Errors. International Journal on Information Theories & Applications. V. 10, N 2, 2003, p. 198-204.
3. Bolshakov, I.A., A. Gelbukh. On Detection of Malapropisms by Multistage Collocation Testing. In: A. Düsterhöft, B. Talheim (Eds.) Proc. 8th Int. Conference on Applications of Natural Language to Information Systems NLDB´2003, June 2003, Burg, Germany, GI-Edition, LNI, V. P-29, Bonn, 2003, p. 28-41.
4. Bolshakova, E.I. Towards Computer-aided Editing of Scientific and Technical Texts. International Journal on Information Theories & Applications. V. 10, N 2, 2003, p. 204-210.
5. Hirst, G., D. St-Onge. Lexical Chains as Representation of Context for Detection and Corrections of Malapropisms. In: C. Fellbaum (ed.) WordNet: An Electronic Lexical Database. MIT Press, 1998, p. 305-332.
6. Kilgarriff, A., G. Grefenstette. Introduction to the Special Issue on the Web as Corpus. Computational linguistics, V. 29, No. 3, 2003, p. 333-347.
7. Manning, Ch. D., H. Schütze. Foundations of Statistical Natural Language Processing. MIT Press, 1999.

Authors' Information

Elena I. Bolshakova – Moscow State Lomonosov University, Faculty of Computational Mathematics and Cybernetic, Algorithmic Language Department; Leninskie Gory, Moscow State University, VMK, Moscow 119899, Russia; e-mail: bolsh@cs.msu.su

Igor A. Bolshakov – Center for Computing Research (CIC), National Polytechnic Institute (IPN); Av. Juan Dios Bátiz esq. Av. Miguel Othon Mendizabal s/n, U.P. Adolfo Lopez Mateos, Col. Zacatenco, C.P. 07738, Mexico D.F., Mexico; e-mail: igor@cic.ipn.mx

Alexey P. Kotlyarov – Moscow State Lomonosov University, Faculty of Computational Mathematics and Cybernetic, Algorithmic Language Department; Leninskie Gory, Moscow State University, VMK, Moscow 119899, Russia; e-mail: koterpillar@yandex.ru

A MATHEMATICAL APPARATUS FOR DOMAIN ONTOLOGY SIMULATION. AN EXTENDABLE LANGUAGE OF APPLIED LOGIC[1]

Alexander Kleshchev, Irene Artemjeva

Abstract: A mathematical apparatus for domain ontology simulation will be described in the series of the articles. This article is the first one of the series. The paper is devoted to means for representation of domain models and domain ontology models, so here a logical language is used only as a means for formalizing ideas. The chief requirement to such a language is that it must have such a semantic basis that would allow us to determine the most exact approximation of a set of intended interpretation functions as often as possible. Another requirement closely connected with the foregoing one is that the awkwardness of expressing ideas in such a language must not considerably exceed the complexity of their expressing in natural language. There are two ways to meet the requirements. The first one is to define and fix a wide semantic basis of the language. In this case the semantic basis nonetheless can be insufficient for some applications of the language. Extending applications of the language can lead from time to time to the necessity of further extending its semantic basis, i.e. to the

[1] This paper was made according to the program of fundamental scientific research of the Presidium of the Russian Academy of Sciences «Mathematical simulation and intellectual systems», the project "Theoretical foundation of the intellectual systems based on ontologies for intellectual support of scientific researches".

necessity of defining new and new versions of the language. The second way is to make the kernel of the language being as nearer to the semantic basis of the classical language as possible and to allow us to make necessary extensions of the kernel for particular applications. In this article the second way is used to define the extendable language of applied logic. The goal of this article is to define the kernel of the extendable language of applied logic and its standard extension. The standard extension of the language defines elements of the semantic basis that are supposed to be useful practically in all the applications.

Keywords: Extendable language of applied logic, ontology language specification, kernel of extendable language of applied logic, the standard extension of the language of applied logic.

ACM Classification Keywords :I.2.4 Knowledge Representation Formalisms and Methods, F4.1. Mathematical Logic

Introduction

At present the importance of studying properties of domain ontologies is generally recognized [Guarino, 1998] [Studer et al, 1998]. As a rule, a language of predicate calculus of the first order, other languages of mathematical logic [Guarino, 1998] [Wielinga et al, 1994] or ontology description languages [van Heijst et al, 1996] are used for formal representation of ontologies.

Languages of mathematical logic were created for aims that were not connected directly with describing domain ontologies. Therefore, they do not allow us to represent formally all the properties of domain ontologies. In particular, the definition of values and sorts for names of a signature, distinctions between the propositions representing domain knowledge and ontological agreements are beyond the syntax and semantics of these languages. Some poorness of the semantic basis for the most of such languages leads to the fact that the meaning of ontological agreements is obscured by awkwardness of technical details which are necessary to express the meaning in these languages. In addition, the languages of the first order do not allow us to introduce terms of a high level of generality and not taking part in description of situations (states of affairs in terms of the paper [Guarino, 1998]) into ontology descriptions. As a consequence, an ontology description representing properties of all domain terms turns out immense even if these terms can be divided into classes of terms with the same properties.

Semantics of ontology description languages and of predicate calculus languages is equivalent. Both ontologies and domain models can be described in these languages. Therefore, it is not clear in what measure such languages are specific for formalizing exactly domain ontologies and in what way this specificity is represented by their syntax and semantics. Operational aspects of semantics of these languages are not connected with domain ontologies but can be important only for describing task and method ontologies.

In papers [Artemjeva et al, 1995, 1996, 1997a, 1997b], [Kleshchev et al, 1998] an attempt was made to suggest a mathematical apparatus - logic relationship systems - for domain simulation. This apparatus cleared up some of the above troubles. But direct application of this apparatus for simulation of domain ontologies is impossible because domain ontology models are sets of domain models. For such an application it is necessary to make an appropriate generalization of the apparatus.

The goal of the article series is to build a mathematical apparatus for domain ontology simulation. In the article an extendable language of applied logic is defined and also the standard extension of the language. This language will be used for representation of mathematical models of domain ontologies.

1. The Necessity of an Extendable Language of Applied Logic

In this article an extendable language of applied logic is defined. The necessity of the language is motivated by the following circumstances. The language of classical mathematical logic - the language of predicate calculus - was developed to attain two aims conflicting with one another in many respects. First, it is a means for description of generative process states in a predicate calculus (of sets of formulas). Second, it is a means for formalizing ideas.

Any language for description of generative process states (of sets of formulas) in a predicate calculus has to have such a semantic basis (a set of language symbols whose semantics does not depend on an interpretation of

the signature symbols) that the predicate calculus being complete contains a finite set of inference rules. The semantic basis of the classical language of predicate calculus is formed by propositional connectives and quantifiers. In addition, sets (domains of variable values), Cartesian product of sets (domains of definitions of functions and predicates are Cartesian products) and also functional maps (the interpretations of functional and predicate symbols are elements of the functional map set) belong to the semantic basis implicitly. The inference rules of the predicate calculus usually correspond to the semantics of propositional connectives and quantifiers.

With the use of a predicate calculus language as a means for formalizing ideas, every logic theory can be considered as a way of intensional determining a set of interpretation functions having the same finite domain of definition - the signature of the language. In this case, the language has to have such a semantic basis that the most exact approximation of any intended set of interpretation functions can be determined by a finite set of propositions. It is clear, the wider such a semantic basis, the more often this aim can be achieved. In real applications, when declarative models (systems of algebraic, differential and other equations, and also systems of inequalities and optimization tasks) are determined this semantic basis contains arithmetic at the minimum. However, it is well known that a predicate calculus being complete and based on such a language cannot contain a finite set of inference rules.

In this manner, there are, as they are, a few logic languages. Some of them (for describing formulas) have a restricted semantic basis but the others (for formalizing ideas) have an extendable semantic basis. The languages of the first group are means for representation of generative process states in predicate calculus studied within mathematical logic. But the others are means for representation of declarative models studied within abstract and applied mathematics.

This paper is devoted to means for representation of domain models and domain ontology models, so here a logic language is used only as a means for formalizing ideas. The chief requirement to such a language is that it must have such a semantic basis that would allow us to determine the most exact approximation of a set of intended interpretation functions as often as possible. Another requirement closely connected with the foregoing one is that the awkwardness of expressing ideas in such a language must not considerably exceed the complexity of their expressing in natural language. There are two ways to meet the requirements. The first one is to define and fix a wide semantic basis of the language. In this case the semantic basis nonetheless can be insufficient for some applications of the language. Extending applications of the language can lead from time to time to the necessity of further extending its semantic basis, i.e. to the necessity of defining new and new versions of the language. The second way is to make the kernel of the language being as nearer to the semantic basis of the classical language as possible and to allow us to make necessary extensions of the kernel for particular applications.

In this article the second way is used to define the language of applied logic. The definition of the language consists of the kernel of the language only. When the semantic basis is extended for particular applications the following two classes of elements are possible. The elements of the first class can be impossible or undesirable to be defined by means of the kernel of the language and by extensions built. On the contrary, the elements of the second class can be naturally defined by means of the kernel and extensions built. The elements of the first class are described in the standard extension and in specialized extensions in the same form that is used in the description of the kernel of the language. The standard extension of the language defines elements of the semantic basis that are supposed to be useful practically in all the applications. A specialized extension of the language defines elements of the semantic basis that are necessary for a comparatively narrow class of applications. Because the same specialized extensions can be used in different applications such extensions have names. Every particular language of applied logic contains the kernel and usually the standard extension and possibly some specialized extensions. By this means, every particular language of applied logic is characterized by a set of extension names rather than a signature. A signature is introduced by a particular logical theory represented in such a language. Therewith, propositions of the theory can associate values (interpretation) or sorts with names (elements of the signature) or can restrict possible functions of interpretations for these names according to the interpretation of other names. In turn, every theory has a name. The parameters of the name are the names of the extensions of the language that are used for describing the theory. Other theories represented by their names also can be elements of a theory.

In this article the syntax and semantics of auxiliary constructions of the language (terms and formulas) and its basic constructions (propositions and logic theories) are defined.

2. The Kernel of the Applied Logic Language. The Syntax of Terms, Formulas, Propositions and Applied Logic Theory

The syntax of terms. The terms are:

1. a name n;
2. a variable v;
3. N and L;
4. $t_1 \to t_2$, where t_1 and t_2 are terms;
5. $(\times t_1,..., t_k)$, where t_1, ..., t_k are terms;
6. $t(t_1, ..., t_k)$, where t , t_1, ..., t_k are terms;
7. j(t), where t is a term.

The syntax of formulas. The formulas are:

1. $t(t_1, ..., t_k)$, where t , t_1, ..., t_k are terms;
2. $\neg f_1$, f_1 & f_2, $f_1 \vee f_2$, $f_1 \Rightarrow f_2$, $f_1 \Leftrightarrow f_2$, where f_1 and f_2 are formulas.

If a variable is not bound in a term or in a formula then it is considered as free in the term or in the formula. If a variable is bound in a term or in a formula then it is bound also in the proposition including this term or this formula. There are no bound variables in the kernel of the language.

The syntax of propositions. A proposition consists of a prefix and a body. A prefix is a set of variable descriptions $(v_1: t_1)...(v_m: t_m)$ (bounded universal quantifiers), where $(v_i: t_i)$ is a variable description, v_i is a variable, t_i is a term for all i=1, ...,m. For I = 1,...,m only the variables v_1, ..., v_m can be free variables of the term terms t_1, ..., t_m. A set of variable descriptions can be empty. All the variables v_1, ..., v_m are mutually different.

The body of a proposition depends on the type of the proposition. The types of propositions are a value description for a name, a sort description for a name, a restriction on the interpretation of names. Any free variable which is a part of the body of a proposition must be described in its prefix. If a variable is bound in the body of a proposition then it cannot be a part of the prefix of the proposition.

The body of a value description for a name has a form $t_1 \equiv t_2$, where t_1 and t_2 are terms.

The body of a sort description for a name has a form sort $t_1 : t_2$, where t_1 and t_2 are terms.

The body of a restriction on the interpretation of names is a formula.

The syntax of applied logic theories. An applied logic theory named $T(E_1, ..., E_k)$, where E_1, ..., E_k are the names of extensions of the language used for representing the theory is a pair <TS, SS>, where TS is a finite set (perhaps empty) of names of other theories, SS is a finite set (perhaps empty) of propositions. Any applied logic theory T = <TS, SS> by definition is equivalent to an applied logic theory <∅, SS'>, where SS' is the result of the following process. Let us denote ts(T) = TS, ss(T) =SS. Let TS_1= ts(T) and SS_1= ss(T). For every i = 1,2, ... let $TS_{i+1} = \bigcup_{t \in TS_i} ts(t)$, $SS_{i+1} = SS_i \cup \bigcup_{t \in TS_i} ss(t)$. If $TS_n = \varnothing$ on a recurrent step n then SS' = SS_n. The theory <∅, SS'> will be called the reduction of the theory <TS, SS>.

An applied logic theory will be called *syntactically correct* if

- TS contains only syntactically correct applied logic theories;
- SS contains propositions written by means of the kernel of the language and of its extensions E_1, E_2, ..., E_k only;
- the above process for building the reduction of the logic theory is completed in a finite number of steps;
- the reduction of the theory T contains a nonempty set of propositions.

It is evident that the set of propositions of the reduction of any syntactically correct applied logical theory is a finite set.

3. The Kernel of the Applied Logic Language. Semantics of Terms, Formulas, Propositions and Applied Logic Theory

Semantics of terms and formulas. Semantics of terms and formulas determines the values of terms and formulas and also the conditions under which these values exist. In this case it is suggested that a function α is given on the set of names. For every name the value of the function is an interpretation of the name. The values of terms and formulas will be defined in relation to an interpretation function α and an arbitrary admissible substitution θ of values for all the free variables in the term or in the formula. If a variable being free in a term or in a formula is also free in the proposition including the term or the formula then in an admissible substitution θ the value for the variable is determined by the semantics of the proposition. But if a variable being free in a term or in a formula is bound in the proposition including the term or the formula then in an admissible substitution θ the value for the variable is determined by the semantics of the term or of the formula in that the variable is bound. Let $J_{\alpha\theta}(t)$ denote the value of a term t for an interpretation function α and an admissible substitution θ, $J_{\alpha\theta}(f)$ denote the value of a formula f for an interpretation function α and an admissible θ, $\theta(v)$ denote the value of a variable v in the substitution θ.

The values of terms are defined by the following way.

1. $J_{\alpha\theta}(n) = \alpha(n)$, where n is a name; $J_{\alpha\theta}(n)$ does not depend on θ; the value $J_{\alpha\theta}(n)$ exists if n is an element of the set $J_{\alpha\theta}(N)$;

2. $J_{\alpha\theta}(v) = \theta(v)$, where v is a variable;

3. $J_{\alpha\theta}(N)$ is the infinite set of all possible names; $J_{\alpha\theta}(N)$ does not contain all the names that are described in the standard and in any used specialized extension of the language and also "N", "L", "≡","=", "→", "×", "⇒", "∨", "&", "¬", "⇔", "(", ")", ":", "true", "false", ",", "sort", "j"; $J_{\alpha\theta}(N)$ does not depend on α and θ;

4. $J_{\alpha\theta}(L)$ is the set consisting of two elements true and false; $J_{\alpha\theta}(L)$ does not depend on α and θ;

5. $J_{\alpha\theta}(t_1 \to t_2)$ is the set of all possible completely defined functions from the set $J_{\alpha\theta}(t_1)$ to the set $J_{\alpha\theta}(t_2)$; the value of the term exists if the both values $J_{\alpha\theta}(t_1)$ and $J_{\alpha\theta}(t_2)$ are sets;

6. $J_{\alpha\theta}(\times t_1, ..., t_k)$ is the Cartesian product of the sets $J_{\alpha\theta}(t_1)$, ..., $J_{\alpha\theta}(t_k)$; the value of the term exists if all the values $J_{\alpha\theta}(t_1)$, ..., $J_{\alpha\theta}(t_k)$ are sets; the operation "×" has all the properties of Cartesian product but associativity $J_{\alpha\theta}(\times(\times t_1, t_2), t_3) \neq J_{\alpha\theta}(\times t_1, (\times t_2, t_3))$;

7. $J_{\alpha\theta}(t(t_1, ..., t_k)) = \varphi(J_{\alpha\theta}(t_1), ..., J_{\alpha\theta}(t_k))$ is the value of the function φ which is the interpretation of the name $J_{\alpha\theta}(t)$ (i.e. $\varphi = \alpha(J_{\alpha\theta}(t))$), applied to the arguments $J_{\alpha\theta}(t_1)$, ..., $J_{\alpha\theta}(t_k)$; the value of the term exists if the value $J_{\alpha\theta}(t)$ is a name, having a sort $(s' \to s)$, where s' is the Cartesian product of the sets s_1, ..., s_k or a subset of the Cartesian product, s is a set, with $s \neq J_{\alpha\theta}(L)$, $<J_{\alpha\theta}(t_1), ..., J_{\alpha\theta}(t_k)> \in s'$; in this case $J_{\alpha\theta}(t(t_1, ..., t_k)) \in s$; let us notice that if t' is such a term that $J_{\alpha\theta}(t') = <J_{\alpha\theta}(t_1), ..., J_{\alpha\theta}(t_k)>$ then $J_{\alpha\theta}(t(t')) = J_{\alpha\theta}(t(t_1, ..., t_k))$;

8. $J_{\alpha\theta}(j(t)) = \alpha(J_{\alpha\theta}(t))$ is the interpretation of the name $J_{\alpha\theta}(t)$; the value of the term exists if $J_{\alpha\theta}(t)$ is a name.

The values of formulas are defined in the following way.

1. $J_{\alpha\theta}(t(t_1, ..., t_k)) \Leftrightarrow \rho(J_{\alpha\theta}(t_1), ..., J_{\alpha\theta}(t_k))$ is the value of the predicate ρ, which is the interpretation of the name $J_{\alpha\theta}(t)$ (i.e. $\rho = \alpha(J_{\alpha\theta}(t))$) applied to the arguments $J_{\alpha\theta}(t_1)$, ..., $J_{\alpha\theta}(t_k)$; the formula has a value if the value $J_{\alpha\theta}(t)$ is a name having a sort $(s' \to L)$, where s' is the Cartesian product of the sets s_1, ..., s_k or a subset of the Cartesian product, $<J_{\alpha\theta}(t_1), ..., J_{\alpha\theta}(t_k)> \in s'$; let us notice that if t' is such a term that $J_{\alpha\theta}(t') = <J_{\alpha\theta}(t_1), ..., J_{\alpha\theta}(t_k)>$ then $J_{\alpha\theta}(t(t')) \Leftrightarrow J_{\alpha\theta}(t(t_1, ..., t_k))$;

2. $J_{\alpha\theta}(\neg f) \Leftrightarrow \neg J_{\alpha\theta}(f)$, i.e. the value of the formula $\neg f$ is true if and only if the value $J_{\alpha\theta}(f)$ is false; the formula has a value if the formula f has a value for the interpretation function α and the substitution θ;

3. $J_{\alpha\theta}(f_1 \,\&\, f_2) \Leftrightarrow J_{\alpha\theta}(f_1) \,\&\, J_{\alpha\theta}(f_2)$, i.e. the value of the formula $f_1 \,\&\, f_2$ is true if and only if the both values $J_{\alpha\theta}(f_1)$ and $J_{\alpha\theta}(f_2)$ are true; the formula has a value if the both formulas f_1 and f_2 have values for the interpretation function α and the substitution θ;

4. $J_{\alpha\theta}(f_1 \vee f_2) \Leftrightarrow J_{\alpha\theta}(f_1) \vee J_{\alpha\theta}(f_2)$, i.e. the value of the formula $f_1 \vee f_2$ is true if and only if at least one of the values $J_{\alpha\theta}(f_1)$ or $J_{\alpha\theta}(f_2)$ is true; the formula has a value if the both formulas f_1 and f_2 have values for the interpretation function α and the substitution θ;

5. $J_{\alpha\theta}(f_1 \Rightarrow f_2) \Leftrightarrow J_{\alpha\theta}(f_1) \Rightarrow J_{\alpha\theta}(f_2)$, i.e. the value of the formula $f_1 \Rightarrow f_2$ is true if and only if either the value $J_{\alpha\theta}(f_1)$ is false or the both values $J_{\alpha\theta}(f_1)$ and $J_{\alpha\theta}(f_2)$ are true; the formula has a value if the both formulas f_1 and f_2 have values for the interpretation function α and the substitution θ;

6. $J_{\alpha\theta}(f_1 \Leftrightarrow f_2) \Leftrightarrow J_{\alpha\theta}(f_1) \Leftrightarrow J_{\alpha\theta}(f_2)$, i.e. the value of the formula $f_1 \Leftrightarrow f_2$ is true if and only if either the both values $J_{\alpha\theta}(f_1)$ and $J_{\alpha\theta}(f_2)$ are false or the both values $J_{\alpha\theta}(f_1)$ and $J_{\alpha\theta}(f_2)$ are true; the formula has a value if the both formulas f_1 and f_2 have values for the interpretation function α and the substitution θ;

Semantics of propositions. Semantics of propositions determines the meaning of the propositions and also the conditions under which propositions have meaning.

The set of admissible substitutions θ for free variables of a proposition is formed in the following way. If the prefix of the proposition is empty then the set of admissible substitutions of the proposition consists of the only empty substitution. Let the prefix of the proposition be of the form $(v_1: t_1)...(v_m: t_m)$, then the set of admissible substitutions is the set of all the substitutions $\theta = (v_1/c_1, ..., v_m/c_m)$, where $c_1 \in J_{\alpha\theta 1}(t_1), ..., c_m \in J_{\alpha\theta m}(t_m)$.

A value description for a name with the body $t_1 \equiv t_2$ has the following meaning: for every admissible substitution θ the interpretation of the name $J_{\alpha\theta}(t_1)$ is $J_{\alpha\theta}(t_2)$. The proposition has meaning if for all the admissible substitutions the value $J_{\alpha\theta}(t_1)$ is a name, the value of the term t_2 exists for the interpretation function α and for the substitution θ and also it does not follow from the logical theory that the name $J_{\alpha\theta}(t_1)$ has more than one value. A set of value descriptions for names can contain recursive value definitions for names.

A sort description for a name with the body sort $t_1 : t_2$ has the following meaning: for every admissible substitution θ the name $J_{\alpha\theta}(t_1)$ has the sort $J_{\alpha\theta}(t_2)$. The proposition has meaning if for all the admissible substitutions $J_{\alpha\theta}(t_1)$ is a name, $J_{\alpha\theta}(t_2)$ is a set and it does not follow from the logical theory that the name $J_{\alpha\theta}(t_1)$ has more than one sort. A set of sort descriptions for names can contain recursive sort definitions for names.

If a sort description for a name has the body sort $t_1 : t_2 \rightarrow t_3$ and $J_{\alpha\theta}(t_3) \neq J_{\alpha\theta}(L)$ then we will say that the name $J_{\alpha\theta}(t_1)$ is a functional name; if $J_{\alpha\theta}(t_3) = J_{\alpha\theta}(L)$ then we will say that the name $J_{\alpha\theta}(t_1)$ is a predicative name; otherwise we will say that the name $J_{\alpha\theta}(t_1)$ is an objective name.

A restriction on the interpretation of names has the following meaning: an interpretation function α is admissible if $J_{\alpha\theta}(f)$ = true for all the admissible substitutions θ, where f is a formula that is the body of this proposition. The proposition has meaning if there is such an interpretation function that the formula f is true for all the admissible substitutions θ.

Semantics of applied logic theories. The set of names being parts of an applied logic theory can be divided into two nonintersecting subsets: a set of uniquely interpreted names and a set of ambiguously interpreted names. A name is uniquely interpreted if one of the following conditions is met:

- the applied logic theory determines neither any sort nor any value for a name n; in this case for any α the interpretation $\alpha(n) = n$;

- the applied logic theory determines a value e for a name n and the value does not depend on the interpretations of other names; in this case for any α the interpretation $\alpha(n) = e$;

- the applied logic theory determines a value e for a name n and the value is uniquely determined by the interpretations of other names.

All the other names are ambiguously interpreted. For every such a name the applied logic theory determines a sort s but does not determine any value. In this case any interpretation function α must meet the restriction $\alpha(n) \in s$.

An interpretation function α is admissible for an applied logic theory if all the propositions of the theory reduction have meaning for this interpretation function. An applied logic theory is semantically correct if there is an admissible interpretation function α. Since for every proposition the set of admissible substitutions is determined

uniquely but the admissible interpretation function is determined ambiguously then a semantically correct applied logic theory determines a set of admissible interpretation functions. It is easily seen that under these conditions the set of ambiguously interpreted names of any semantically correct applied logic theory is finite for any admissible interpretation function.

The constriction of an admissible interpretation function α to the set of ambiguously interpreted names of an applied logic theory will be called a model of the theory. A model of an applied logic theory can be represented by such a set of value descriptions for names that after adding the set to the theory all the names of the new theory built in such a way will be uniquely interpreted.

4. The Standard Extension of the Language of Applied Logic

An extension of the language is a description of syntax and semantics for terms and formulas. These terms and formulas are added to the kernel of the language of applied logic. The standard extension ST of the language of applied logic introduces syntactic constructions for some special languages of mathematical logic and also arithmetic and set-theoretic constants, operations and relations. The syntax of many constructions is usual for mathematical expressions. All the quantifier constructions have a unified syntax $A(v_1: t_1)\ldots(v_m: t_m)$ t Ω or $A(v_1: t_1)\ldots(v_m: t_m)$ f Ω, where A and Ω are quantifier brackets (unique for every quantifier), $(v_1: t_1)\ldots(v_m: t_m)$ is a set of variable descriptions, t is a term, and f is a formula. The variables $v_1, \ldots, v_m$ are bound in this construction.

The terms are:

1. a quantifier construction $(\iota(v_1: t_1)\ldots(v_m: t_m)$ f) (iota-operator); $J_{\alpha,\theta}((\iota(v_1: t_1)\ldots(v_m: t_m)$ f)) is equal to such an element of the set of admissible substitutions for $(v_1: t_1)\ldots(v_m: t_m)$ that if it was substituted for the variables $v_1,\ldots, v_m$ then the value $J_{\alpha\theta}(f)$ would be true; the value of the term exists if the values $J_{\alpha\theta}(t_1), \ldots, J_{\alpha\theta}(t_m)$ are sets and such an element of the Cartesian product is unique that if it was substituted for the variables then the value $J_{\alpha,\theta}(f)$ would be true;

2. a quantifier construction $(\lambda(v_1: t_1)\ldots(v_m: t_m)$ t) (lambda-term determining a function); $J_{\alpha\theta}((\lambda(v_1: t_1)\ldots(v_m: t_m)$ t)) is a function φ of m arguments; for any element $<q_1, .., q_m>$ of the set of admissible substitutions for $(v_1: t_1)\ldots (v_m: t_m)$ the value $\varphi(q_1,q_2,..,q_m) = J_{\alpha\theta}(t)$;

3. a quantifier construction $(\lambda(v_1: t_1)\ldots(v_m: t_m)$ f) (lambda-term determining a predicate); $J_{\alpha\theta}((\lambda(v_1:t_1)\ldots(v_m:t_m)$ f)) is a predicate ρ of m arguments; for any element $<q_1,q_2,..,q_m>$ of the set of admissible substitutions for $(v_1: t_1)\ldots (v_m: t_m)$ the value $\rho(q_1,q_2,..,q_m) \Leftrightarrow J_{\alpha\theta}(f)$;

4. $/(f_1 \Rightarrow t_1), \ldots, (f_m \Rightarrow t_m)/$ (conditional term), where $t_1, .., t_m$ are terms and $f_1, .., f_m$ are formulas; $J_{\alpha\theta}(/(f_1 \Rightarrow t_1), \ldots, f_m \Rightarrow t_m)/) = J_{\alpha\theta}(t_k)$ under the condition that $J_{\alpha\theta}(f_k)$ is true; the value of the term exists if all the terms $t_1, \ldots, t_m$ and all the formulas $f_1, \ldots, f_m$ have values for the interpretation function α and the substitution θ and also there is the only k such that $J_{\alpha\theta}(f_k)$ is true;

5. numerical constants r; $J_{\alpha\theta}(r)$ has the value of the number corresponding to the numerical constant r; $J_{\alpha\theta}(r)$ does not depend on α and θ;

6. R and also $J_{\alpha\theta}(R)$ is the set of all the real numbers; $J_{\alpha\theta}(R)$ does not depend on α and θ;

6. $t_1 + t_2$, or $t_1 - t_2$, or $t_1 * t_2$, or t_1 / t_2 where t_1 and t_2 are terms; $J_{\alpha\theta}(t_1 + t_2) = J_{\alpha\theta}(t_1) + J_{\alpha\theta}(t_2)$, i.e. $J_{\alpha\theta}(t_1 + t_2)$ is the sum of $J_{\alpha\theta}(t_1)$ and $J_{\alpha\theta}(t_2)$; the value of the term exists if both $J_{\alpha\theta}(t_1)$ and $J_{\alpha\theta}(t_2)$ are numbers; $J_{\alpha\theta}(\tau)$ where τ is $t_1 - t_2$ or $t_1 * t_2$ or t_1 / t_2 is defined in such a way;

11. $t_1 \uparrow t_2$, where t_1 and t_2 are terms; $J_{\alpha\theta}(t_1 \uparrow t_2) = J_{\alpha\theta}(t_1) \uparrow J_{\alpha\theta}(t_2)$, i.e. $J_{\alpha\theta}(t_1 \uparrow t_2)$ is $J_{\alpha\theta}(t_2)$-th power of the number $J_{\alpha\theta}(t_1)$; the value of the term exists if both $J_{\alpha\theta}(t_1)$ and $J_{\alpha\theta}(t_2)$ are numbers and $J_{\alpha\theta}(t_2)$-th power of the number $J_{\alpha\theta}(t_1)$ exists;

12. $\varnothing$ and also $J_{\alpha\theta}(\varnothing)$ is the empty set; $J_{\alpha\theta}(\varnothing)$ does not depend on α and θ;

13.$\{t_1, \ldots, t_k\}$, where $t_1, \ldots, t_k$ are terms; $J_{\alpha,\theta}(\{t_1, \ldots, t_k\}) = \{J_{\alpha\theta}(t_1), \ldots, J_{\alpha\theta}(t_k)\}$, i.e. the value of the term is the set those elements are $J_{\alpha\theta}(t_1), \ldots, J_{\alpha\theta}(t_k)$; the value of the term exists if for the interpretation function α and the substitution θ the terms $t_1, \ldots, t_k$ have values;

14.{ }t, where t is a term; $J_{\alpha\theta}(\{\ \}t)$ is the set of all the finite subsets (including the empty set and maybe $J_{\alpha\theta}(t)$) of the set $J_{\alpha\theta}(t)$; the value of the term exists if $J_{\alpha\theta}(t)$ is a set;

15. a quantifier construction $\{(v_1: t_1)\ldots(v_m: t_m)\ f\}$ (intensionality quantifier); $J_{\alpha\theta}(\{(v_1: t_1)\ldots(v_m: t_m)\ f\})$ is the subset of the set of admissible substitutions for $(v_1: t_1)\ldots(v_m: t_m)$ that $J_{\alpha\theta}(f)$= true; the value of the term exists if the formula f has a value for all the admissible substitutions;

16. a quantifier construction $\{(v_1: t_1)\ldots(v_m: t_m)\ t\}$ (quantifier of set transformation); $J_{\alpha\theta}(\{(v_1: t_1)\ldots(v_m: t_m)\ t\})$ is the set of all the values of the term $J_{\alpha\theta}(t)$, where θ belongs to the set of admissible substitutions for $(v_1: t_1)\ldots(v_m: t_m)$; the value of the term exists if $J_{\alpha\theta}(t)$ is a set and if for the interpretation function α and for any admissible substitution θ the term t has a value;

17. $t_1 \cup t_2$, or $t_1 \cap t_2$, or $t_1 \setminus t_2$, where t_1 and t_2 are terms; $J_{\alpha\theta}(t_1 \cup t_2) = J_{\alpha\theta}(t_1) \cup J_{\alpha\theta}(t_2)$, i.e. $J_{\alpha\theta}(t_1 \cup t_2)$ is the union of the sets $J_{\alpha\theta}(t_1)$ and $J_{\alpha\theta}(t_2)$; the value of the term exists if both $J_{\alpha\theta}(t_1)$ and $J_{\alpha\theta}(t_2)$ are sets; $J_{\alpha\theta}(\tau)$ where τ is $t_1 \cap t_2$ or $t_1 \setminus t_2$ is defined in such a way;

18. $\mu(t)$, where t is a term; $J_{\alpha\theta}(\mu(t))$ is the cardinality of the set $J_{\alpha\theta}(t)$; the value of the term exists if $J_{\alpha\theta}(t)$ is a finite set;

19. $t_1 \Uparrow t_2$, where t_1 and t_2 are terms; $J_{\alpha\theta}(t_1 \Uparrow t_2) = J_{\alpha\theta}(t_1) \Uparrow J_{\alpha\theta}(t_2)$, i.e. $J_{\alpha\theta}(t_1 \Uparrow t_2)$ is the set $J_{\alpha\theta}(t_1)$ raised to the Cartesian power $J_{\alpha\theta}(t_2)$; the value of the term exists if $J_{\alpha\theta}(t_1)$ is a set and $J_{\alpha\theta}(t_2)$ is a positive integer; the operation "$\Uparrow$" has all the properties of the Cartesian power but associativity: $J_{\alpha\theta}((t_1 \Uparrow t_2) \Uparrow t_3) \neq J_{\alpha\theta}(t_1 \Uparrow (t_2 * t_3))$;

20. $\langle t_1, \ldots, t_m\rangle$, where $t_1, \ldots, t_m$ are terms; $J_{\alpha\theta}(\langle t_1, \ldots, t_m\rangle) = \langle J_{\alpha\theta}(t_1), \ldots, J_{\alpha\theta}(t_m)\rangle$, i.e. the value of the term is the m-tuple composed of the values $J_{\alpha\theta}(t_1), \ldots, J_{\alpha\theta}(t_m)$; the value of the term exists if for the interpretation function α and the substitution θ the terms $t_1, \ldots, t_m$ have values;

21. $\pi(t_1, t_2)$, where t_1 and t_2 are terms; $J_{\alpha\theta}(\pi(t_1, t_2)) = \pi(J_{\alpha\theta}(t_1), J_{\alpha\theta}(t_2))$, i.e. the value of the term is the $J_{\alpha\theta}(t_1)$-th projection of the tuple (of an element of a Cartesian product) $J_{\alpha\theta}(t_2)$; the value of the term exists if $J_{\alpha\theta}(t_2)$ is a m-tuple and $J_{\alpha\theta}(t_1)$ is a positive integer not greater than m;

22. length(t), where t is a term; $J_{\alpha\theta}(\text{length}(t))$ is the number of elements in the tuple $J_{\alpha\theta}(t)$; the value of the term exists if $J_{\alpha\theta}(t)$ is a tuple.

The formulas are:

1. $t_1 = t_2$, or $t_1 \neq t_2$, where t_1 and t_2 are terms; $J_{\alpha\theta}(t_1 = t_2) \Leftrightarrow J_{\alpha\theta}(t_1) = J_{\alpha\theta}(t_2)$, i.e. the value of the formula is true if and only if the values $J_{\alpha\theta}(t_1)$ and $J_{\alpha\theta}(t_2)$ are the same; the formula has a value if for the interpretation function α and the substitution θ the both terms t_1 and t_2 have values; $J_{\alpha\theta}(t_1 \neq t_2)$ is defined in such a way;

2. $t_1 > t_2$, or $t_1 < t_2$, or $t_1 \leq t_2$, or $t_1 \geq t_2$, where t_1 and t_2 are terms; $J_{\alpha\theta}(t_1 > t_2) \Leftrightarrow J_{\alpha\theta}(t_1) > J_{\alpha\theta}(t_2)$, i.e. the value of the formula is true if and only if the value $J_{\alpha\theta}(t_1)$ is greater than the value $J_{\alpha\theta}(t_2)$; the formula has a value if both $J_{\alpha\theta}(t_1)$ and $J_{\alpha\theta}(t_2)$ are numbers; $J_{\alpha\theta}(\phi)$ where ϕ is $t_1 < t_2$, or $t_1 \leq t_2$, or $t_1 \geq t_2$ is defined in such a way;

3. $t_1 \in t_2$, or $t_1 \notin t_2$, where t_1 and t_2 are terms; $J_{\alpha\theta}(t_1 \in t_2) \Leftrightarrow J_{\alpha\theta}(t_1) \in J_{\alpha\theta}(t_2)$, i.e. the value of the formula is true if and only if the value $J_{\alpha\theta}(t_1)$ belongs to the set $J_{\alpha\theta}(t_2)$; the formula has a value if for the interpretation function α and the substitution θ the term t_1 has a value and $J_{\alpha\theta}(t_2)$ is a set; $J_{\alpha\theta}(t_1 \notin t_2)$ is defined in such a way;

4. $t_1 \subset t_2$, or $t_1 \subseteq t_2$, or $t_1 \not\subset t_2$, where t_1 and t_2 are terms; $J_{\alpha\theta}(t_1 \subset t_2) \Leftrightarrow J_{\alpha\theta}(t_1) \subset J_{\alpha\theta}(t_2)$, i.e. the value of the formula is true if and only if the set $J_{\alpha\theta}(t_1)$ is a proper subset of the set $J_{\alpha\theta}(t_2)$; the formula has a value if both $J_{\alpha\theta}(t_1)$ and $J_{\alpha\theta}(t_2)$ are sets; $J_{\alpha\theta}(\phi)$ where ϕ is $t_1 \subseteq t_2$, or $t_1 \not\subset t_2$ is defined in such a way.

Conclusions

In this article the kernel of the extendable language of applied logic has been introduced. Any applied logic theory is characterized by a set (perhaps empty) consisting of the standard extension and specialized extensions of the language. The article also defines the standard extension of the language. The standard extension introduces syntactic constructions for some special languages of mathematical logic and also arithmetic and set-theoretic constants, operations and relations.

References

[Guarino, 1998] Guarino N. Formal Ontology and Information Systems. In Proceeding of International Conference on Formal Ontology in Information Systems (FOIS'98), N. Guarino (ed.), Trento, Italy, June 6-8, 1998. Amsterdam, IOS Press.

[Studer et al, 1998] Studer R., Benjamins V.R., Fensel D. Knowledge Engineering: Principles and Methods. In Data & Knolwedge Engineering, 1998, 25, p. 161-197.

[Wielinga et al, 1994] Wielinga, B., Schreiber A.T., Jansweijer W., Anjewierden A. and van Harmelen F. Framework and Formalism for Expressing Ontologies (version 1). ESPRIT Project 8145 KACTUS, Free University of Amsterdam deliverable, DO1b.1, 1994.

[van Heijst et al, 1996] van Heijst G., Schreiber A.Th., Wielinga B.J. Using Explicit Ontologies in KBS Development. In Intern. Jornnal of Human and Computer Studies, 1996, 46 (2-3), pp. 183-292.

[Artemjeva et al, 1995] Artemjeva I.L., Gavrilova T.L., Kleshchev A.S. Domain Models with Elementary Objects. In Scientific-Technical Information, Series 2, 1995, № 12. P. 8-18 (in Russian).

[Artemjeva et al, 1996] Artemjeva I.L., Gavrilova T.L., Kleshchev A.S. Logical Relationship Systems with Elementary Objects. In Scientific-Technical Information, Series 2, 1996, № 1, pp. 11-18 (in Russian).

[Artemjeva et al, 1997a] Artemjeva I.L., Gavrilova T.L., Kleshchev A.S. Logical Domain Models of the Second Order. In Scientific-Technical Information, Series 2, 1997, № 6, pp. 14-30 (in Russian).

[Artemjeva et al, 1997b] Artemjeva I.L., Gavrilova T.L., Kleshchev A.S. Logical Relationship Systems with Parameters. In Scientific-Technical Information, Series 2, 1997, № 7, pp. 19-23 (in Russian).

[Kleshchev et al, 1998] Kleshchev A.S., Artemjeva I.L., Gavrilova T.L., Surov V.V. Application of Logical Relationship Systems for Expert System Development. In Appl. of Advanced Information Technologies: Proc. of the Forth World Congress on Expert Systems, 16-20 March 1998, Mexico City Cognisant Communication Corporation, 1998, vol.1: 500-510.

Authors' Information

Alexander S. Kleshchev – kleschev@iacp.dvo.ru

Irene L. Artemjeva – artemeva@iacp.dvo.ru

Institute for Automation & Control Processes, Far Eastern Branch of the Russian Academy of Sciences

5 Radio Street, Vladivostok, Russia

INFORMATION PROCESSING IN A COGNITIVE MODEL OF NLP

Velina Slavova, Alona Soschen, Luke Immes

Abstract: A model of the cognitive process of natural language processing has been developed using the formalism of generalized nets. Following this stage-simulating model, the treatment of information inevitably includes phases, which require joint operations in two knowledge spaces – language and semantics. In order to examine and formalize the relations between the language and the semantic levels of treatment, the language is presented as an information system, conceived on the bases of human cognitive resources, semantic primitives, semantic operators and language rules and data. This approach is applied for modeling a specific grammatical rule – the secondary predication in Russian. Grammatical rules of the language space are expressed as operators in the semantic space. Examples from the linguistics domain are treated and several conclusions for the semantics of the modeled rule are made. The results of applying the information system approach to the language turn up to be consistent with the stages of treatment modeled with the generalized net.

Keywords: Cognitive model, Natural Language Processing, Generalized Net, Language Information System

ACM Classification Keywords: I.2.7 Natural Language Processing;