Bull. Nov. Comp. Center, Comp. Science, 40 (2016), 53–74

An approach to context-dependent lexical and syntactic ambiguity resolution in ontology population*

N.O. Garanina, E.A. Sidorova

Abstract. We suggest an approach to the resolution of context-dependent lexical and syntactic ambiguity in a framework of ontology population from natural language texts. We show that a set of maximally determined ontology instances can be represented as a Scott information system with an entailment relation as a collection of information connections. Moreover, consistent primary lexical instances form FCA-concepts. These representations are used to justify the correctness of lexical disambiguation and to define syntactic ambiguity and its resolution. This information system generates a multi-agent system in which agents resolve the ambiguity of both types.

Keywords: ambiguity resolution, lexical ambiguity, syntactic ambiguity, ontology population, information retrieval, Scott information systems, formal concept analysis, multiagent systems.

1. Introduction

Currently, ontological databases are widely used for storing information obtained from a great number of sources. To complete such ontologies, formalisms and methods that allow one to automate the process are developed. Features of automatic information retrieval cause ontology population ambiguities. In linguistics, several kinds of ambiguities are considered: lexical, syntactic, semantic, and pragmatic [2]. In the process of ontology population from natural language texts, we use our algorithms [5] in which the following ambiguity types appear: (1) several ontology instances or data attributes correspond to the same text fragment, (2) some value is incorrectly assigned to some attribute of some instance, (3) some value is incorrectly assigned to attributes of several instances, (4) some value is incorrectly assigned to several attributes of some instance, (5) several values are assigned to a one-valued attribute of some instance. The first type corresponds to lexical ambiguity, and the other types are syntactic ambiguity. An algorithm for lexical disambiguation was represented in [6]. In this work, we suggest a modified algorithm for resolving lexical ambiguity and a new algorithm for syntactic disambiguation, and we justify the correctness of them both.

*Supported by Russian Foundation for Basic Research under Grant 15-07-04144 and Siberian Branch of Russian Academy of Science (Integration Grant n.15/10 "Mathematical and Methodological Aspects of Intellectual Information Systems").

In [6], we demonstrated that the process of information retrieval represented in the form of a set of ontology instances can be presented as a Scott information system [13]. This process produces maximally determined instances for ontology population. In this paper, we prove that consistent sets of instances and lexical objects whose values assign attributes of these instances form FCA concepts [3]. This fact guarantees that information states of ambiguous conflicting agents do not intersect. This implies the correctness of lexical disambiguation.

Besides, now we use a representation of ontologies which does not consider ontology relations as special structures. In these ontologies, only classes are allowed and relations are represented as special attributes of classes. The well-known ontology representation language OWL uses this kind of notation. This representation is a good solution for the specification of polyadic relations. It makes the algorithms for ontology population simpler because class and relation instances are packed in the same item.

Automatic techniques of disambiguation do not usually use an input data context in full. This can lead to incomplete and incorrect ambiguity resolution [1, 7, 8, 9, 11]. Our approach tries to get around these drawbacks. We use a distributed approach for disambiguation. Every retrieved instance is related to an agent. These agents detect and resolve ambiguities with the help of a special master agent. This approach takes polynomial time for disambiguation.

The rest of the paper is organized as follows. In Section 2, an approach to ontology population in the framework of information systems is discussed. Section 3 describes lexical and syntactic disambiguation in terms of the system defined in the previous section. The next section gives definitions for a multi-agent system of context-dependent ambiguity resolution. Section 5 describes agents of our systems, their action protocols, and the main conflict resolution algorithm. In the concluding Section 6, directions of future research are discussed.

2. Scott information systems in ontology population

Let us have an ontology of a subject domain, the ontology population rules, semantic and syntactic models for a sublanguage of the subject domain and a data format, and input data as a finite natural language text with information for the population of the ontology. We consider *an ontology O of a subject domain* which includes (1) a finite nonempty set C_O of classes for the concepts of the subject domain, (2) a finite set of attributes with names in $Dat_O \cup Rel_O$, each of which has values in some data domain (data attributes in Dat_O) or is some instance of the ontology (relation attributes in Rel_O, which model relations), and (3) a finite set D_O of data types. Every class $c \in C_O$ is defined by a tuple of typed attributes: $c = (Dat_c, Rel_c)$,

where every data attribute $\alpha \in Dat_c \subseteq Dat_O$ has a type $d_\alpha \in D_O$ with values in V_{d_α} and every relation attribute $\rho \in Rel_c \subseteq Rel_O$ is of a class $c_\rho \in C_O$. Let a set of all values of all attributes be $V_O = \cup_{d_\alpha \in D_O} V_{d_\alpha}$. *An information content* IC_O of the ontology O is a set of class instances, where every instance $a \in IC_O$ is of the form (c_a, Dat_a, Rel_a), where c_a is a class of the instance, every data attribute in Dat_a has a name $\alpha \in Dat_{c_a}$ with values in V_{d_α} and every relation attribute in Rel_a has a name $\rho \in Rel_{c_a}$ with a value as an instance of a class c_ρ. *The ontology population problem* is to compute an information content for a given ontology from given input data. Input data for the ontology population process are natural language texts. These data are finite and our algorithms for ontology-oriented text analysis can generate a finite set of ontology instances [5]. The finiteness of the set is guaranteed by the prohibition for the rules to generate infinite information items by one position. We suggest considering this process of forming ontology instances as working with Scott information systems. A *Scott information system* T is a triple $(T, Con, \vdash)$, where

- T is a set of tokens and $Fin(T)$ is a set of finite subsets;
- Con is a consistency predicate such that $Con \subseteq Fin(T)$, and

  1. $Y \in Con$ and $X \subseteq Y \Rightarrow X \in Con$,
  2. $a \in T \Rightarrow \{a\} \in Con$;

- $\vdash$ is an entailment relation such that $\vdash \subseteq Con \setminus \{\emptyset\} \times T$ and

  3. $X \vdash a \Rightarrow X \cup \{a\} \in Con$,
  4. $X \in Con$ and $a \in X \Rightarrow X \vdash a$,
  5. $\forall b \in Y : X \vdash b$ and $Y \vdash c \Rightarrow X \vdash c$.

The *information retrieval system* based on an ontology, finite input data, and on rules of the ontology population and data processing is defined as a triple $R = (A, Con, \vdash)$. The set of tokens A consists of a set of all (underdetermined) ontology *p-instances* formed by the rules in the determination process of initial p-instances which are retrieved from an input text by special preprocess. Every p-instance $a \in A$ has the form (c_a, Dat_a, Rel_a, P_a), where

- the class $c_a \in C_O$, and
- every data p-attribute $\alpha_a \in Dat_a$ is of the form (α, IV_α), where
  - the name $\alpha \in Dat_{c_a}$, where
  - its information values $\bar{v} \in IV_\alpha$ has the form $(v_{\bar{v}}, g_{\bar{v}}, s_{\bar{v}})$ with
    - the data value $v_{\bar{v}} \in d_\alpha$, a set of all values of α is $Val_{\alpha_a} = \{v_{\bar{v}} \mid \bar{v} \in IV_\alpha\}$,

  - $g_{\bar{v}}$ is grammar information (morphological and syntactic features), and
  - $s_{\bar{v}}$ is structural information (position in input data);

- every relation p-attribute $\rho_a \in Rel_a$ is of the form (ρ, O_{ρ_a}), where
  - the name $\rho \in Rel_{c_a}$, and
  - every relation object $\bar{o} \in O_{\rho_a}$ has the form (o, p_o), where
    - o is an instance of a class c_{ρ_a}, and
    - $p_o \in P_o$ is its position,

    a set of all relation instances of a is $O(Rel_a) = \{a\} \cup_{\rho_a \in Rel_a} \{o | \bar{o} \in O_{\rho_a}\}$;

- P_a is structural information (a set of positions in input data).

We consider a special set of tokens: a set of *lexical objects LO* corresponding to the values of data attributes retrieved from input data. Every lexical object is a p-instance which has only a single data attribute with a single information value. Every non-lexical p-instance is an *information object* in a set IO: $A = LO \cup IO$. P-instances correspond to ontology instances in a natural way. Let $a = (c_a, Dat_a, Rel_a, p_a)$ be a p-instance, then its corresponding ontology instance is $a' = (c_a, Dat_{a'}, Rel_{a'})$, where every $\alpha \in Dat_{a'}$ has its value(s) in Val_{α_a} and every $\rho \in Rel_{a'}$ has its value o with $(o, p_o) \in O_{\rho_a}$. Further we omit the prefix "p-" if there is no ambiguity. An *information order* relation $\prec$ is defined on ontology instances. Let $a, a' \in A$: $a \prec a'$, if $a = a'$ everywhere except for at least one attribute, with the number of values of this attribute in a being strictly less than that in a'. For $x, x' \in A$: if $x \prec x'$, then x' is *an information extension of* x.

Rules of ontology population and data processing $Rules = \{rule_1, \ldots, rule_n\}$ map the finite sets of instances of ontology classes to an instance which is an informational extension of some instance of the domain set or a new instance. These sets must be linguistically and ontologically compatible: specified sets of their attributes and some instances have to satisfy conditions on values, grammatical and structural information [10]:
$rule_i : Dom_i \mapsto A$, $Dom_i \subseteq 2^A$, such that
$\forall X \in Dom_i \ : \ LingCons_i(\cup_{x' \in X} Dat_{x'} \cup Rel_{x'}) = true \wedge \forall x' \in X : c_{x'} \in Class_i$,
where the predicate $LingCons_i$ and the set of classes $Class_i \subseteq C_O$ detect the linguistic and ontological compatibility of the instance set, correspondingly. Let for $X \in Dom_i, x \in A$:
$rule_i(X) = x$ iff $((\exists y \in X : y \prec x \wedge c_x = c_y) \vee (\forall y \in X : y \not\prec x \wedge c_x = gen_i(X))) \wedge$
$(Dat_x = \emptyset \vee Dat_x = \cup_\alpha(\alpha, \cup\{(f_i(\bar{V_\alpha}), g_i(\bar{V_\alpha}), s_i(\bar{V_\alpha})) \mid$
$\exists Y_\alpha \subseteq X : dat_\alpha \subseteq \cap_{y \in Y_\alpha} Dat_y \wedge \bar{V_\alpha} = \cup_{\beta \in dat_\alpha} \{\bar{v} | \bar{v} \in IV_\beta\}\})) \wedge$

$$(Rel_x = \emptyset \vee \forall o \in O(Rel_x) : o \in X \cup_{y \in X} O(Rel_y)),$$

where $gen_i(X)$ generates a new class for a new instance, $f_i(\bar{V})$ produces a value based on values in $(\bar{V})$ for an attribute of the instance x, and $g_i(\bar{V})$ and $s_i(\bar{V})$ inherit grammatical and structural information from the set of information values $\bar{V}$.

The consistency predicate Con and entailment relation $\vdash$ correspond to the rules of ontology population and data processing. Let $x, x' \in A$ and $X \subseteq A$. The entailment relation connects informationally associated tokens:

– $X \vdash x$, iff $x \in X$, or $x \notin X \wedge$

$$((\exists X' \subseteq X, X'' \subseteq A, rule_i \in Rules : rule_i(X' \cup X'') = x) \vee$$
$$(\exists x' \in A, X'' \subseteq A, rule_i \in Rules : X \vdash x' \wedge rule_i(\{x'\} \cup X'') = x)),$$

i.e., the instance x is entailed from X if it is in this set or information from tokens of this set is used for evaluating the attributes of x.

The consistency predicate defines informationally consistent sets of tokens:

– $X \in Con$, iff for some $rule_i \in Rules$ the following holds

$\forall X' \subseteq X(\cup_{x' \in X'} c_x \subseteq Class_i) \Rightarrow (\exists x \in A, X'' \subseteq A : rule_i(X' \cup X'') = x)$,

i.e., if there exists some rule which can find in a set of tokens some instances satisfying its class compatibility then these instances should be consistent with some other set of tokens with respect to the rule. The sets that are class compatible but linguistically incompatible cannot be processed by rules, hence we do not consider them consistent.

Let us prove the following theorem for the system R:

Theorem 1. *The triple $R = (A, Con, \vdash)$ is a Scott information system.*

Proof. Let us show that the consistency predicate Con and the entailment relation $\vdash$ satisfy properties 1–5 of information systems.

1. $Y \in Con$ and $X \subseteq Y \Rightarrow X \in Con$. This fact follows from the definition of the consistency predicate directly because the condition of the definition should hold for every subset of a consistent set.

2. $a \in A \Rightarrow \{a\} \in Con$. By the definition, for every $rule_i \in Rules$ the class of a is not included in $Class_i$ or a single $\{a\}$ can be complemented by some set of tokens in such a way that the $rule_i$ produces a new token.

3. $X \in Con$ and $X \vdash a \Rightarrow X \cup \{a\} \in Con$. $X \cup \{a\}$ is consistent because $a \in X$ or lexical information of a is inherited from X by definitions of the entailment relation and process rules.

4. $X \in Con$ and $a \in X \Rightarrow X \vdash a$ by the definition of the entailment relation.

5. $\forall b \in Y : X \vdash b$ and $Y \vdash c \Rightarrow X \vdash c$. Let $Y_b = Y \setminus \{b\}$. $X \vdash c$ iff $(\exists b \in Y, Y_b \subseteq A, rule \in Rules : X \vdash b \wedge rule(\{b\} \cup Y_b) = c))$ by the definition of the entailment relation.■

The proposition below directly follows from the monotonicity of the entailment relation and finiteness of input data.

Proposition 1. *Information retrieval process of ontology population terminates.*

For a token $x \in A$: $x^{\uparrow} = \{x\} \cup \{x' \mid x \prec x'\}$ and $x^{\downarrow} = \{x\} \cup \{x' \mid x' \prec x\}$ are *upper and down cones of* x. Let a set of *maximally determined* instances in a set X (maximal instances or tokens) be $X^{\uparrow} = \{x \in X \mid x^{\uparrow} = \{x\}\}$. We consider the tokens from LO to be always maximally determined: $LO = LO^{\uparrow}$. The result of the analysis of input data is $A^{\uparrow}$. These instances may populate an ontology. The following proposition is obvious.

Proposition 2. *The triple* $I = (A^{\uparrow}, Con, \vdash)$ *is a Scott information system.*

Information descendants of a token $a \in A^{\uparrow}$ are all maximal tokens (all information) that can be obtained from this token by the entailment relation: $Ds(a) = \{x \in A^{\uparrow} | \{a\} \vdash x\}$. *Information ancestors* of a token $a \in A^{\uparrow}$ are all maximal tokens from which a can be obtained: $An(a) = \{x \in A^{\uparrow} | \{x\} \vdash a\}$. In our framework, the following equality holds for lexical objects: $An(a) = \{a\}$ because ontology instances are based on retrieved lexical objects. Information descendants are a particular case of Scott information states [14]. Like in the cited paper, we show that tokens from LO and their information descendants form a concept lattice.

Proposition 3. *Consistent sets of lexical objects form FCA concepts. Every consistent set of instances is a base for FCA concepts.*

Proof. Let every set x of information descendants of LO be an object and every $l \in LO$ be an attribute. A lexical object l is an attribute of x iff $l \in x$. The extension of a set of attributes $L \subseteq LO$ is the set $L' = \{x | L \subseteq x\}$ and the intension of L' is the set $\{l | \forall x \in L', l \in x\}$. L is a concept iff the condition on the intension of the extension of L holds: $L = \{l | \forall x \in L', l \in x\}$ iff L is an information state of the information system I iff L is a consistent set. The intension of a set of infostates X is the set $X' = \{l | \forall x \in X, l \in x\}$ and the extension of X' is the set $\{x | X' \subseteq x\}$. X is a concept iff the condition on the extension of the intension of X holds: $X = \{x | X' \subseteq x\}$ iff a set of all instances in the set of infostates X forms an infostate too: $X^i = \{a \mid a \in x \in X\}$, hence X^i is a consistent set. Hence, every consistent set of instances is a base for FCA concepts.■

3. Ambiguity and resolution

(1) Lexical ambiguity.
Let $l, l' \in LO$ be in conflict $l \leftrightsquigarrow l'$ iff $s(l) \cap s(l') \neq \emptyset$. Let $AmbLO$ be a

set of conflict lexical objects and Lex be a set of their descendants. We consider that the rules in $Rules$ cannot generate instances which include inconsistent information. That is, for every $rule_i \in Rules$ it holds that $\forall X \in Dom_i, a, a' \in X, l, l' \in An(a) \cap An(a') \cap LO : \neg(l \leftrightsquigarrow l')$. Hence, for the lexical objects l and l' in conflict: $Ds(l) \cap Ds(l') = \emptyset$.

For the lexical disambiguation of two conflicting lexical objects, we prefer a lexical object which is more incorporated in an input text than its competitor. For $l, l' \in LO$, if $|Des(l)| > |Des(l')|$, we take l for evaluating attributes of ontology instances and ignore l'.

(2) Syntactic ambiguity.
Detecting syntactic ambiguity frequently requires the analysis of homogeneous groups. Syntactic ambiguity is defined for ontology instances, not for lexical objects. Our types of syntactic ambiguity can depend on an ontology specification, hence here we actually consider syntactic-semantic ambiguity. We omit "-semantic" for brevity. Syntactic ambiguity usually can be expressed by corresponding a single lexical object to several ontology items in various ways. For disambiguation, it is necessary to find in an input text an evidence of correctness of the correspondence. This could be performed using the following inequalities. Let us define several useful abbreviations.

- For every instance a and its data attribute $\alpha \in Dat_a$, *a set of information values equal to* $v \in d_\alpha$ is $EQ(a, \alpha, v) = \{\bar{v} \in IV_\alpha \mid v_{\bar{v}} = v\}$.
- For every instance a and its relation attribute $\rho \in Rel_a$, *a set of relation objects with an instance equal to* $e \in c_{\rho_a}$ is $EQ(a, \rho, e) = \{(o, p_o) \in O_\rho \mid o = e\}$.
- The power of these sets is *an evidence power*.
- A triple $(a, \alpha, \bar{v})$ denotes the information value $\bar{v} \in IV_\alpha$ of the data attribute $\alpha \in Dat_a$ of the instance a.
- A couple (a, ρ) denotes a value of the relation attribute $\rho \in Rel_a$ of the instance a.
- A set of *information values which effects* $(a, \alpha, \bar{v})$ is $V(a, \alpha, \bar{v}) = \{(c, \gamma, \bar{w}) | \exists rule_i \in Rules, X \subseteq A^\uparrow : rule_i(X) = a \wedge c \in X \wedge \gamma \in dat_\alpha \cap Dat_c \wedge \bar{w} \in IV_\gamma \wedge \bar{v} = (f(\bar{V_\alpha}), g(\bar{V_\alpha}), s(\bar{V_\alpha}))\}$.
- A set of *instances which effects* (a, ρ) is $I(a, \rho) = \{e \in A^\uparrow \mid \exists rule \in Rules, X \subseteq A^\uparrow : rule(X) = a \wedge e \in X \wedge O_\rho \cap O(Rel_e) \neq \emptyset\}$.

Now we define a method of syntactic disambiguation.
(1) *Some value is incorrectly assigned to some attribute of an instance (Synt11).*
An example: "The old men and woman sat on the bench." The woman may or may not be old. Hence, the attribute "age" of the instance "woman"

may not have the value "old". Let the set of instances with the ambiguity of this type be denoted as $Synt11$. Let in an instance a an information value $(c, \gamma, \bar{w})$ effect $(a, \alpha, \bar{v})$: $(c, \gamma, \bar{w}) \in V(a, \alpha, \bar{v})$. Then, in the case of this ambiguity, $(c, \gamma, \bar{w})$ is declared as effecting $(a, \alpha, \bar{v})$ iff $|EQ(a, \alpha, v_{\bar{v}})| > 1$. Let in an instance a an instance e effect (a, ρ): $e \in I(a, \rho)$. Then, in the case of the ambiguity, the instance e is declared as effecting (a, ρ) iff $|EQ(a, \rho, e)| > 1$.

(2) *Some value is incorrectly assigned to attributes of several instances (Synt12).*

An example: "Someone shot the maid of the actress who was on the balcony." Either the actress or the maid was on the balcony. Hence, either the attribute "place" of the instance "actress" or the attribute "place" of the instance "maid" may have the value "balcony." Let the set of instances with ambiguity of this type be denoted as $Synt12$. Let in instances a and b an information value $(c, \gamma, \bar{w})$ effect $(a, \alpha, \bar{v})$ and $(b, \beta, \bar{u})$: $(c, \gamma, \bar{w}) \in V(a, \alpha, \bar{v}) \cap V(b, \beta, \bar{u})$. Then, in the case of this ambiguity, $(c, \gamma, \bar{w})$ is declared as effecting $(a, \alpha, \bar{v})$ and not $(b, \beta, \bar{u})$ iff $|EQ(a, \alpha, v_{\bar{v}})| > |EQ(b, \beta, \bar{u})|$. Let in instances a and b, an instance e effect (a, ρ) and (b, o): $e \in I(a, \rho) \cap I(b, o)$. Then, in the case of the ambiguity, the instance e is declared as effecting (a, ρ) and not (b, o) iff $|EQ(a, \rho, e)| > |EQ(b, o, e)|$.

(3) *A value is incorrectly assigned to several attributes of an instance (Synt112).*

An example: "Cuban jazz band." A group of Cuban musicians performing jazz music or a group of musicians performing Cuban jazz. Hence, the attribute "country" or the attribute "style" of the instance "band" may have the value "Cuban". Let the set of instances with the ambiguity of this type be denoted as $Synt112$. Let in an instance a an information value $(c, \gamma, \bar{w})$ effect $(a, \alpha, \bar{v})$ and $(a, \beta, \bar{u})$: $(c, \gamma, \bar{w}) \in V(a, \alpha, \bar{v}) \cap V(a, \beta, \bar{u})$. Then, in the case of this ambiguity, $(c, \gamma, \bar{w})$ is declared as effecting $(a, \alpha, \bar{v})$ and not $(a, \beta, \bar{u})$ iff $|EQ(a, \alpha, v_{\bar{v}})| > |EQ(a, \beta, v_{\bar{u}})|$. Let in an instance a an instance e effect (a, ρ) and (a, o): $e \in I(a, \rho) \cap I(a, o)$. Then, in the case of the ambiguity, the instance e is declared as effecting (a, ρ) and not (a, o) iff $|EQ(a, \rho, e)| > |EQ(a, o, e)|$.

(4) *Several values are assigned to a one-valued attribute of an instance (Synt211).*

An example: "Shakespeare is the author of the piece." The gender of Shakespeare may be either male or female. Hence, the one-valued attribute "gender" of the instance "person" may have the value either "male" or "female." Let the set of instances with the ambiguity of this type be denoted as $Synt211$. Let in an instance a information values $(b, \beta, \bar{u})$ and $(c, \gamma, \bar{w})$ effect $(a, \alpha, \bar{v})$ and $(a, \alpha, \bar{v'})$, respectively: $(b, \beta, \bar{u}) \in V(a, \alpha, \bar{v})$ and $(c, \gamma, \bar{w}) \in V(a, \alpha, \bar{v'})$. Then, in the case of this ambiguity, $(b, \beta, \bar{u})$ is declared as effecting $(a, \alpha, \bar{v})$, and $(c, \gamma, \bar{w})$ is declared as not effecting α iff $|EQ(a, \alpha, v_{\bar{v}})| > |EQ(a, \alpha, v_{\bar{v'}})|$. Let in an instance a instances e and e' ef-

fect (a, ρ): $e, e' \in I(a, \rho)$. Then, in the case of the ambiguity, the instance e is declared as effecting (a, ρ) and e' is declared as not effecting (a, ρ) iff $|EQ(a, \rho, e)| > |EQ(a, \rho, e')|$.

In the case of equal evidence powers, the conflict is not resolved. We consider systems in which all these ambiguities are independent, i.e., the pairwise intersections of the sets Lex, $Synt11$, $Synt12$, $Synt112$ and $Synt211$ are empty. The following informal descriptions of action protocols for instance agents present resolution of independent lexical and syntactic ambiguities. These protocols work correctly if the coreference resolution and detection of syntactic ambiguities are correct.

4. Multi-agent ambiguity resolution

Let a set of lexical objects which effect some information value of a data attribute α of an instance a be $L(a, \alpha) = \{l \in LO \mid \exists rule_i \in Rules, X \subseteq A^{\uparrow} : rule_i(X) = a \wedge (\exists x \in X : x \in Ds(l) \wedge (\exists \beta \in dat_\alpha \cap Dat_x : l \in L(x, \beta)))\}$. For every $x \notin A^{\uparrow}$, the corresponding maximally determined instance is $\tilde{x}$ such that $\tilde{x} \in A^{\uparrow} \wedge x \prec \tilde{x}$. The entailment relation $\vdash$ generates *information connections* between maximally determined instances. Let $X \vdash x$ and $y \in X \wedge y \notin x^{\downarrow}$. Then *an information connection* between $\tilde{y}$ and $\tilde{x}$ is $\tilde{y} \xrightarrow{\tilde{\omega}} \tilde{x}$ iff $(\exists \alpha \in Dat_x, \beta \in Dat_y : \omega \in L(x, \alpha) \cap L(y, \beta)) \vee (\omega \in O(Rel_x) \cap O(Rel_y)$

– of *an updating type* $\tilde{y} \xrightarrow{\tilde{\omega}^u} \tilde{x}$ iff $\exists x' \in X \ : \ x' \prec x$,

– of *a generating type* $\tilde{y} \xrightarrow{\tilde{\omega}^g} \tilde{x}$ iff $\nexists x' \in X \ : \ x' \prec x$.

The information system of information retrieval R generates a multi-agent system with typed connections. Agents of the system resolve the ambiguities by computing and comparing the context cardinalities and evidence powers. The information system $(A, Con, \vdash)$ generates the *Multi-agent System of Ambiguity Resolution* (MASAR) as a tuple $S = (A, C, I, T)$, where

- $A = \{a_x \mid x \in A^{\uparrow}\}$ is a finite set of agents corresponding to maximally determined instances;
- $C = \{\tilde{\omega} \mid \exists x, y \in A : \tilde{x} \xrightarrow{\tilde{\omega}} \tilde{y}\}$ is a finite set of connections;
- a mapping $I : C \longrightarrow 2^{A \times A}$ is an interpretation function of ordered connections between agents: $I(c) = (a_x, a_y)$ iff $\tilde{x} \xrightarrow{c} \tilde{y}$;
- a mapping $T : C \times A \times A \longrightarrow \{gen, upd\}$ determines types of connections: $T(c, a_x, a_y) = gen$ iff $I(c) = (a_x, a_y) \rightarrow (\tilde{x} \xrightarrow{c^g} \tilde{y})$, and $T(c, a_x, a_y) = upd$ iff $I(c) = (a_x, a_y) \rightarrow (\tilde{x} \xrightarrow{c^u} \tilde{y})$.

Let *information agents* IA correspond to information objects from $IO^{\uparrow}$ and *lexical agents* LA correspond to lexical objects from LO.

Not every instance from $IO^{\uparrow}$ is used for ontology population. There is a set of utility instances Utl. They do not resolve ambiguities or populate an

ontology. They just transfer information to its descendants. Hence $IO^{\uparrow} = Ont \cup Utl$, where only instances from Ont may populate an ontology.

For every agent $a \in A$, we define the following sets of agents and connections. We omit symmetric definitions of ancestors $Anc*$ (for $Des*$) and utility predecessors $UtP*$ (for $UtS*$) for brevity:

- $C_a = \{c \in C | \exists a' \in A : (a, a') \in I_C(c) \bigvee (a', a) \in I_C(c)\}$ is the connections of a;
- $Cg_a = \{c \in C | \exists a' \in A : (a', a) \in I_C(c) \wedge T(c, a, a') = gen\}$ is the generating connections of a;
- $Cu_a = \{c \in C | \exists a' \in A : (a', a) \in I_C(c) \wedge T(c, a, a') = upd\}$ is the updating connections of a;
- $Scg_a^c = \{a' \in A \mid (a, a') \in I_C(c) \wedge c \in Cg_{a'}\}$ is a set of generated successors by the connection c;
- $Scu_a^c = \{a' \in A \mid (a, a') \in I_C(c) \wedge c \in Cu_{a'} = upd\}$ is a set of updated successors by the connection c;
- $Sc_a^c = Scg_a^c \cup Scu_a^c$ is a set of all successors by the connection c;
- $Pr_a^c = \{a' \in A \mid (a', a) \in I_C(c)\}$ is a set of predecessors by the connection c;
- $UtS_a^c = \{a' \in Utl \mid (a, a') \in I_C(c)\}$ is a set of utility successors by the connection c; and
- $Des_a^c = Sc_a^c \cup \bigcup_{a' \in Sc_a^c} Des_{a'}^c$ is descendants by the connection c.

MASAR is a multiagent system of information dependencies. In these systems, agents can use information from predecessors and can pass the processed information to successors. Hence $Des_a^c \cap Anc_a^c = \emptyset$, i.e., every connection has no cycle because of information transfer.

A weight of an agent corresponds to the number and quality (in the case of generation) of its non-utility ancestors and descendants. For every $a \in A$

- $wt_{Pr}^a(c) = 1 + \sum_{a' \in Pr_a^c} wt_{Pr}^{a'}(c)$ is *the weight of connection ancestors*,
- $wt_{Sc}^a(c) = 1 + \sum_{a' \in Scg_a^c} wt(a') + \sum_{a' \in Scu_a^c} wt_{Sc}^{a'}(c)$ is *the weight of connection descendants*,
- $wt_{UtP}^a(c) = 1 + \sum_{a' \in UtP_a^c} wt_{UtP}^{a'}(c)$ is *the weight of connection utility ancestors*,
- $wt_{UtS}^a(c) = 1 + \sum_{a' \in UtS_a^c} wt_{UtS}^{a'}(c)$ is *the weight of connection utility descendants*, and
- $wt(a) = 1 + \sum_{c \in C_a} (wt_{Pr}^a(c) + wt_{Sc}^a(c) - (wt_{UtP}^a(c) + wt_{UtS}^a(c)))$ is *the weight of information agents.*

The weight of the system S is $wt(S) = \sum_{a \in Ont} wt(a)$.

The problem of conflict resolution in MASAR is to obtain a conflict-free MASAR of the maximal weight. A multiagent algorithm below produces such a system.

5. Conflict resolution in MASAR

In this paper, we consider independent ambiguities only. In this case, order of their resolution is irrelevant. It is expedient, however, to resolve lexical ambiguity first, because this disambiguation effects the existence of ontology instances. Syntactic disambiguation refines information distribution among instances.

Action protocols for conflict resolution used by MASAR agents form a multi-agent system of conflict resolution MACR. The system MACR includes the set of MASAR agents and the agent-master. Note that a fully distributed version of our algorithm could be developed but it would be very ineffective. The result of agents' interactions by protocols described below is the conflict-free MASAR. All agents execute their protocols in parallel until the master detects termination. The system is dynamic because MASAR agents can be deleted from the system. The agents are connected by synchronous duplex channels. The master agent is connected with all agents, MASAR agents are connected with their successors and predecessors, and conflict lexical agents are connected too. Messages are transmitted via a reliable medium and stored in channels until they are read.

For correct lexical disambiguation, it is necessary to find the groups of lexical agents which impact the weights of each other in case one of them is removed. Let us denote these *groups of relatives* as *Relatives*. The agents of the groups from *Relatives* have common generated descendants: $\forall Rlt \in Relatives(\forall a_\omega \in Rlt(\exists a_\nu \in Rlt : Ds(a_\omega, a_\nu) \neq \emptyset \wedge \exists b \in Ds(a_\omega, a_\nu) : (\omega \in Cg_b \vee \nu \in Cg_b) \wedge \forall a \in AmbLO \setminus Rlt : Ds(a_\omega, a) = \emptyset))$, where $Ds(x, y) = Ds(x) \cap Ds(y)$ for $x, y \in LO$. Due to the mutual effect of relatives on their weights, it is necessary to resolve conflicts between the groups of relatives. Note that due to Proposition 3 the set $AmbLO$ can be disjoined to nonintersecting subsets of relatives. Let $Frn = \cup_{i=1}^{n} Rlt_i$ ($Rlt_i \in Relatives$ for $i \in [1..n]$) be *a group of friends* iff $\forall a, b \in Frn : \neg(a \nleftrightarrow b)$. The groups of friends Frn_1 and Frn_2 are *in conflict* $Frn_1 \nleftrightarrow Frn_2$ iff $(\forall a \in Frn_1 \exists b \in Frn_2 : a \nleftrightarrow b) \wedge (\forall b \in Frn_2 \exists a \in Frn_1 : b \nleftrightarrow a)$. *The conflict is resolved* for the benefit of the group with a greater weight, i.e., if $\sum_{a \in Frn_1} wt(a) > \sum_{b \in Frn_2} wt(b)$, then the agents of the group Frn_2 are removed from the system and their descendants delete their inherited values of attributes or the descendant removes itself if the lexical value from a lexical agent in Frn_2 is generating for this descendant.

Hence, for resolving all conflicts in the system, it is necessary to perform

the following steps: (1) to compute the weights of agents, (2) to detect relative groups, (3) to compute independent conflict groups of friends, (4) to resolve lexical conflicts between the groups, (5) to make the corresponding change in the system, and (6) to resolve all kinds of syntactic ambiguity. The agent-master coordinates MASAR agents. It computes conflict groups and detects agents to be removed. All other activities are performed by MASAR agents asynchronously. Due to parallel execution, all computations take polynomial time.

Let $A = \{a_1, ..., a_n\}$ be the MASAR agents set, and M be the master agent. Let `Ai` be an interface protocol of the agent a_i and `M` be the protocol of actions of the agent-master M. Then the multi-agent conflict resolution algorithm MACR can be presented in pseudocode as follows:

`MACR:: parallel {A1} ...{An} {M}`

Below we give informal descriptions of protocols of the system agents.

(1) An interface protocol for system agents

This protocol specifies agent's reactions to incoming messages. These messages include information about which actions should be performed by the agent:

(1) `Start`: to start;
(2) `CompWeight`: to compute its weight;
(3) `SendRlt` and `FindRlt`: to find relatives;
(4) `Remove`: to remove connections or itself;
(5) `Synt*` and `Res*`: to resolve some syntactic ambiguity.

Until an input message causes an agent to react, the agent stays in the wait mode. Messages for an agent are stored in the input channel `Input`. They include information about the names of actions `act` and information for performing these action `inf`: `msgA` = `act` × `inf`, where `act` = {`start`, `stop`, `Comp`, `Find`, `Rem`, `Synt`} and parameters `inf` are defined in the corresponding protocol descriptions. Let function `get(Set)` choose an arbitrary element of a nonempty set `Set` and remove it.

The interface protocol of a system agent a.

```
a.Ai ::
set of msgA Input; msgA mess = (start, ∅);
1. while ( mess.act != stop )
2.    if ( Input != ∅ ) then {
3.         mess = get( Input );
4.         if ( mess.act = Comp ) then a.CompWeight;
5.         if ( mess.act = Find ) then
               if(a ∈ IO ) a.SendRlt;
               if(a ∈ LO ) a.FindRlt;
6.         if ( mess.act = Rem  ) then a.Remove(mess.inf);
7.         if ( mess.act = Synt11 ) then a.Synt11;
```

```
8.          if ( mess.act = Res11  ) then a.Res11(mess.inf);
9.          if ( mess.act = Synt12 ) then a.Synt12;
10.         if ( mess.act = Res12  ) then a.Res12(mess.inf);
11.         if ( mess.act = Synt112 ) then a.Synt112;
12.         if ( mess.act = Synt211 ) then a.Synt211;}
```

(2) The main algorithm for conflict resolution

Let us give an informal description of the protocol `Master`. First, the agent-master computes the set of lexical agents *LO* (line 1) choosing for this set the agents without information predecessors. Then the master finds a set of conflict lexical agents *AmbLO* (line 2), using information about their position in the input text: agents are in conflict if their positions intersect. This set is stored as a structure `AgConfs {agent ag; set of agent CoList;}`, where `ag` is an agent in conflict and `CoList` is a set of its conflict partners. After that it sends `Start` to all agents and launches parallel computing of the agents' weights (line 4) and search for lexical agents' relatives (line 5). After all agents finish their job, the master computes conflict pairs of friend groups using the method `ConfGroups` described below (line 6). By comparing the weights of the conflict groups of friends, it forms a list of agents to be removed (line 7). After finishing this resolution of group conflicts, the master launches the corresponding system changes (line 8). After the termination of these changes, it initiates all kinds of syntactic disambiguation for instance agents (lines 10-17).

The protocol of the master agent for conflict resolution.

```
Master ::
 agent a, b;
 set of agent LexAgs, ToRem, F1, F2;
 AgConfs LexAmb;
 list of array [2] of set of agent ConfGroups;
 1. LexAgs = FindLex();
 2. LexAmb = FindLexAmb();
 3. forall a∈ A send ( start ) to a;
 4. forall a∈LexAgs send ( Comp ) to a; wait Finish;
 5. forall a∈ A send ( Find ) to a; wait Finish;
 6. ConfGroups = FindConfGroups();
 7. while ( ConfGroups≠ ∅ ){
      F1 = head(ConfGroups)[1];
      F2 = head(ConfGroups)[2];
      if (∑a∈F1 a.wt < ∑b∈F2 b.wt) then ToRem = ToRem ∪ F1;
      if (∑a∈F1 a.wt > ∑b∈F2 b.wt) then ToRem = ToRem ∪ F2;
      recalculate( ConfGroups );}
 8. forall a∈ToRem send ( Rem, ∅ ) to a; wait Finish;
 9.  A = A \ Rem_A;
```

```
10. forall a∈ A send ( Synt11 ) to a; wait Finish;
11. A = A \ Rem_A;
12. forall a∈ A send ( Synt12 ) to a; wait Finish;
13. A = A \ Rem_A;
14. forall a∈ A send ( Synt112 ) to a; wait Finish;
15. A = A \ Rem_A;
16. forall a∈ A send ( Synt211 ) to a; wait Finish;
17. A = A \ Rem_A;
18. forall a∈ A send ( stop ) to a;
```

When the master computes the conflict pairs of friend groups, it really constructs pair groups relative agents closed under the conflict relation. First, the master defines a set of current conflicts as a set of all conflict lexical agents (line 1). Next, all conflict agents detect the sets of their conflict relatives `a.Amb` (line 2). At the beginning, the group of friends `G1` consists of the conflict relatives of the first conflict lexical agent, including this agent (line 3). At the same time, its conflict group `G2` consists of the agents from the conflict list of the first agent and their conflict relatives (lines 4–5). Here we consider agents having only a one-element conflict list. In another case, comparing conflict groups is more difficult. Then, while the current conflict set is not empty, the master detects whether the groups G1 and G2 are in conflict using the predicate `IsConf` (line 7). If they are in conflict, it removes them from the current `Conf`, stores this pair in the list of such pairs `ConfGroups`, and forms the next pair of groups for conflict checking in the same way as in lines 3–5. If the groups G1 and G2 are not in conflict yet, i.e., they include lonely agents which do not have a conflict partner in the opposite group, the master extends the corresponding group with the conflict partners of lonely agents (line 8). After that, it checks them for conflictness again.

```
Master.FindConfGroups() ::
 agent a, b;
 set of agent A2, G1, G2;
 AgConfs Conf;
 1.  Conf = LexAmb;
 2.  forall a∈LexAmb a.Amb = Amb(a.Rlt);
 3.  G1 = Conf[1].ag.Amb;
 4.  A2 = Conf[1].CoList;
 5.  forall a∈A2 G2 = G2∪a.Amb;
 6.  while( Conf != ∅)
 7.      if ( IsConf( G1, G2 )) then
              Conf = Conf.Rem( G1, G2 );
              ConfGroups.Add( G1, G2 );
```

```
              G1 = Conf[1].ag.Amb;
              A2 = Conf[1].CoList;
              forall a∈A2 G2 = G2∪a.Amb;
 8.       else forall a∈notConf(G2)
                     A2 = Conf[a].CoList;
                     forall b∈A2 G2 = G2∪b.Amb;
              forall a∈notConf(G1)
                     A2 = Conf[a].CoList;
                     forall b∈A2 G1 = G1∪b.Amb;
 9. return ConfGroups;
```

(3) Computing the agents' weight

Following the definitions of the weights, a system agent a computes in parallel the weights of the descendants a.Sc(c) and ancestors a.Pr(c) by every connection $c \in C_a$, launching the corresponding subprocesses for each $c \in C_a$ (line 2 of a.CompWeight). These subprocesses send the integer weights of their descendants (ancestors) increased by 1 to predecessors (successors), respectively (line 4 of a.Pr(c) and lines 4,5 of a.Sc(c)). Utility subprocesses do not increase the weights (line 1 of a.Pr(c) and a.Sc(c)). If a parent connection c is of the type *gen*, then the corresponding descendants' subprocesses send the whole weight of a to the predecessors (line 5 of a.Sc(c)). When these parallel computations are finished, the agent computes its own weight (line 4 of a.CompWeight). The protocol of weights computing belongs to the class of wave echo algorithms [12].

The protocol of a system agent a for weight computing.

```
a.CompWeight ::
 array [C_a] of int: wPr, wSc;
// The own weight a.wt
 1. a.wt = null;
 2. parallel forall c_i ∈ C_a {Pr(c_i)} {Sc(c_i)}
 3. wait Finish;
 4. a.wt = 1 + Σ_{c_i∈C_a}(wPr[c_i] + wSc[c_i];
// Ancestors
 a.Pr(c_i) ::
   set of int Input; int wght;
   int NumP = |Pr_a^{c_i}|;
 1. if ( a.ut=true ) wPr[c_i] = 0; else wPr[c_i] = 1;
 2. while( NumP != 0 )
 3.        if ( Input != ∅ ) then {
                wght = get( Input );
                wPr[c_i] = wPr[c_i] + wght;
                NumP = NumP - 1;}
```

```
 4. forall ( b∈ Sc_a^{c_i} )
             send ( wPr[c_i] ) to b.Pr(c_i);
// Descendants
 a.Sc(c_i) ::
   set of int Input; int wght;
   int NumS = |Sc_a^{c_i}|;
 1. if ( a.ut=true ) wSc[c_i] = 0; else wSc[c_i] = 1;
 2. while( NumS != 0 )
 3.         if ( Input != ∅ ) then {
                 wght = get( Input );
                 wSc[c_i] = wSc[c_i] + wght;
                 NumS = NumS - 1;}
 4. if (c_i.type = upd ) then
       forall ( b∈ Pr_a^{c_i} ) send ( wSc[c_i] ) to b.Sc(c_i);
 5. if (c_i.type = gen ) then
       wait ( a.wt != null );
       forall ( b∈ Pr_a^{c_i} ) send ( a.wt ) to b.Sc(c_i);
```

(4) Computing the agents' relatives

In this algorithm, the agents construct a reflexive-transitive closure of the relative relation. Computing relatives consists of two stages. Agents act asynchronously.

(1) Preliminary search. Since the connections between the information agents from IA are labeled, they know some groups of relatives (possibly incomplete groups). In the protocol `SendRel` they share this knowledge with lexical agents.

(2) Merging. Lexical agents take relative lists from information agents until the latter stop sending them. Since the relative relation is transitive, lexical agents have to interchange their relative lists for constructing a reflexive-transitive closure of the relation. The interchange goes on until all relative lists become stable. Termination of all list transfers can be detected by the AB-algorithm from [4].

The protocol of an information agent a for the preliminary search.

```
a.SendRel ::
 int id;
 1. forall( id∈ Cg_a ) send(Rlt:Cg_a ∪ Cu_a) to ag(id);
 2. forall( id∈ Cu_a ) send(Rlt:Cg_a) to ag(id);
```

The protocol of a lexical agent a for relatives computing.

```
a.FindRlt ( Rlt ) ::
 int id, nOld, nNew;
```

```
mess msg;
set of mess Input;
1.  rltOld = {a.id};
2.  while( FromAgents )
3.      if( Input != ∅ ) then {
4.          msg = get( Input );
5.          a.Rlt = a.Rlt∪msg.Rlt; }
6.  forall(id∈a.Rlt) send(Rlt: a.Rlt) to ag(id);
7.  while( FromLex )
8.      if( Input != ∅ ) then {
9.          msg = get( Input );
10.         nOld = |a.Rlt|;
11.         a.Rlt = a.Rlt∪msg.Rlt;
12.         nNew = |a.Rlt|;
13.         if( nNew > nOld ) then
14.             forall(id∈a.Rlt) send(Rlt: a.Rlt) to ag(id);}
```

(5) Removing LO-agents from the system
If an agent a has to be removed from the system, then

1. all its predecessors remove all connections with it and delete a from the sets of successors;
2. its descendants remove
   (a) all connections with it,
   (b) the corresponding predecessors and
   (c) the corresponding attribute value;
   (d) if the removing connection is of the generating type, then the descendant has to be removed from the system.

An agent has a local variable a`.Rmvd`, which is true if a has performed an action *Remove* and is false otherwise. Let the method `a.RemVal(`c`)` remove attribute values corresponding to the connection c. In the protocol, after the agent checks the presence status if it detects that its successor has provoked it into changing, it removes the successor from the corresponding list (line 2). If changes come with a generating connection or from the master (line 3), the agent has to be removed from the system. If the connection is of the updating type, the agent just removes its corresponding data (line 4). After its own changes, the agent induces the corresponding changes of the connected agents (lines 5–7).

The protocol of a system agent a for changing the system.

a`.Remove(` x`,` c`) :: {`

```
agent b;
connection e;
set of connections Con;
1.  if( a.Rmvd ) then return;
2.  if( x ∈ Sc_a^c ) then Sc_a^c = Sc_a^c \ {x}; return;
3.  if( T(c, x, a) = gen | x = ∅) then
        Con = C_a;
        Rem_A = Rem_A ∪ {a};
        a.Rmvd = true;
4.  if( T(c, x, a) = upd ) then
        Con = {c};
        C_a = C_a \ {c};
        Pr_a^c = ∅;
        Sc_a^c = ∅;
        a.RemVal(c);
5.  forall e∈Con {
6.    if( a.Rmvd ) for b∈ Pr_a^e send( Rem, a, e ) to b;
7.    forall b∈ Sc_a^e send( Rem, a, e ) to b; } }
```

Let for every agent a and every connection $c \in C_a$ the set of partners by the c-connection Sc_a^c be ordered by positions in input data: $Sc_a^c = \{sc_1, \ldots, sc_m\}$. For simplicity, we also suggest that attribute values are not computed, i.e., they are just equal to effecting lexical objects. Protocols below can be easily generalized for computable attribute values. Resolution of syntactic ambiguities Synt11 and Synt12 consists of two steps.

(6) Synt11 resolution.

(1) Ambiguity detection. Every agent using the set of its successors checks if some attribute value effects values of several instances. If it is so and these instances form a homogeneous group and satisfy a predefined grammar condition (line 2), it sends a message with the type of the conflict and conflict value to every agent in the group excluding the first agent in the group (line 3).

The protocol of an agent a for detecting Synt11.

```
a.Synt11() :: {
1.  forall c∈ C_a {
2.          if( |Sc_a^c| > 1 & Hom(Sc_a^c) & GramHom(Sc_a^c)) then
3.              forall b∈ Sc_a^c\sc_1 send( Res11, a, c ) to b; } }
```

(2) Ambiguity resolution. The agents in the group resolve the ambiguities following the resolving formulas for Synt11. For this purpose, they compute the evidence power of the ambiguous value. If the value is not valid, the agent removes it from the system (line 2).

The protocol of an agent a for resolving Synt11.

```
a.Res11(x, c) :: {
 1.  forall ω ∈ Data ∪ Rela
 2.          if( |EQ(a,ω,c)| ≤ 1)) then
                 send( Rem, x, c ) to a;
```

(7) Synt12 resolution.
(1) The ambiguity detection a.`Synt12()`. Every agent, using the set of its successors, checks if some attribute value effects values of several instances. If it is so and these instances satisfy a predefined grammar condition and do not form a homogeneous group, it sends a message with the type of the conflict, conflict value, and ids of the competitors `Com` to the first agent in the group.
The protocol of an agent a for detecting Synt12.

```
a.Synt12() :: {
 array of agents Com;
 1.  forall c∈ Ca {
 2.          Com = Sc_a^c;
 3.          if( |Com| > 1 & NotHom(Com) & GramNotHom(Com) then
                 send( Synt12, a, c, Com, 1 ) to Com[1];}}
```

(2) The ambiguity resolution a.`Res12(`x`, c, Com, i)`. Agents resolve the ambiguities following the resolving formulas for Synt12. The resolving agent compares its evidence power with the evidence power of the nearest unexplored neighbor (line 6) after the neighbor has sent the power `msg.ep` in the message `msg` (line 5). If the agent's power is greater than the neighbor's power, then the neighbor removes its ambiguous value from the system and the agent moves to the next neighbor (line 6). In the other case, the neighbor continues to resolve the ambiguity and the agent removes its ambiguous value from the system (line 7).
The protocol of an agent a for resolving Synt12.

```
a.Res12(x, c, Com, i) :: {
 int ep; j = i+1;
 messEP msg;
 1.  forall ω ∈ Data ∪ Rela
 2.          ep = |EQ(a,ω,c)|;
 3.          if( ep ≥ 1) then
                 while( j < |Com|+1 )
 4.                  send( GiveEP, a, c ) to Com[j];
 5.                  wait( msg.act = TakeEP );
 6.                  if( msg.ep < ep ) then
                         send( Rem, x, c ) to Com[j];
                         j = j+1;
```

```
7.                  if( msg.ep > ep ) then
                        send( Res12, x, c, Com, j ) to Com[j];
                        send( Rem, x, c ) to a;
                        return;}
```

(8) Synt112 resolution.
If an agent finds attributes ω_1 and ω_2 with a value w (lines 2–4), then it compares the evidence powers $EQ(a, \omega_1, w)$ and $EQ(a, \omega_2, w)$ (lines 6, 7). The attribute value is removed from the values of the attribute with less power. Let the type `IIV` denote informational values and relation objects.
The protocol of an agent a for detecting and resolving Synt112.

```
a.Synt112() :: {
 IIV iv; set of IIV ivs;
 1. forall ω1,ω2 ∈ Data ∪ Rel_a
 2.    ivs = IV_ω1 ∩ IV_ω2; dat = true;
 3.    if( |ivs| = 0 ) ivs = O_ω1 ∩ O_ω2; dat = false;
 4.    if( |ivs| > 0 ) then
 5.        forall iv∈ivs
              if( dat ) then w = v_iv; else w = o_iv;
 6.           if( |EQ(a, ω1, w)| > |EQ(a, ω2, w|) ) then
                  send ( Rem, x, c^ω2 ) to a;
 7.           if( |EQ(a, ω1, w)| < |EQ(a, ω2, w)|) ) then
                  send ( Rem, x, c^ω1 ) to a;
```

(9) Synt211 resolution.
If an agent finds a one-valued attribute ω with different values w_1 and w_2 (line 3), it compares the evidence powers $EQ(a, \omega, w_1)$ and $EQ(a, \omega_1, w_2)$ (lines 5, 6). The attribute value with less power is removed from the values of the attribute. Let the type `VO` denote data values and relation instances.
The protocol of an agent a for detecting and resolving Synt211.

```
a.Synt112() :: {
 VO w1,w2; set of VO Set;
 1. forall ω ∈ Data ∪ Rel_a
 2.     if( ω ∈ Data_a ) then Set = Val_ω; else Set = O_ω;
 3.    if( |Set| > 1 & OneVld(ω)) then
 4.        forall w1,w2∈Set
 5.           if( |EQ(a, ω, w1)| > |EQ(a, ω, w2)|) ) then
                  send ( Rem, x, w1 ) to a;
 6.           if( |EQ(a, ω, w1)| < |EQ(a, ω, w2)|) ) then
                  send ( Rem, x, w2 ) to a;
```

6. Conclusion

In this paper, we show that the maximal instances of the ontology classes taking part in the process of population together with the rules of data processing and ontology population form a Scott information system. This result justifies the resolution of the context-dependent lexical ambiguity by calculating context cardinalities. The Scott information system is also a basis for our approach to syntactic context-dependent ambiguity resolution. This system generates a multi-agent system in which agents resolve the ambiguities by computing the cardinality of their contexts and evidence powers. The suggested algorithm of lexical ambiguity resolution chooses the most powerful group of agents and removes competitors. The choice is based on agents' weights and their impact on the system.

We have considered only independent lexical and syntactic ambiguities. In the near future, we plan to study the disambiguation of a combination of various types of syntactic and lexical ambiguities. In this work, it is useful to introduce a membership probability of attribute ambiguity values and a degree of their effect on other instances. Other direction of research is the study of non-binary conflict relations and development of various types of disambiguation corresponding to these relations.

References

[1] Alfawareh H.M., Jusoh S. Resolving ambiguous entity through context knowledge and fuzzy approach // Intern. J. on Comput. Sci. and Eng. (IJCSE). – 2011. – Vol. 3, No. 1. – P. 410–422. ISSN: 0975-3397.

[2] Berry, D.M., Kamsties, E., Krieger, M.M. From contract drafting to software specification: Linguistic sources of ambiguity. [2003]. URL: http://se.uwaterloo.ca/~dberry/handbook/ambiguityHandbook.pdf (31.01.2016)

[3] Ganter B., Wille R. Formal Concept Analysis. Mathematical Foundations. – Springer Verlag, 1996.

[4] N. O. Garanina, E. V. Bodin. Distributed termination detection by counting agent // Proc. 23nd Intern. Workshop on Concurrency, Specification and Programming / CS&P 2014, Chemnitz, Germany. September 29–October 1, 2014. – Humboldt-Universitat zu Berlin, 2014. – P. 69–79.

[5] Garanina N., Sidorova E., Bodin E. A Multi-agent approach to unstructured data analysis based on domain-specific onthology // Proc. 22nd Intern. Workshop on Concurrency, Specification and Programming / CS&P 2013, Warsaw, Poland, Sept. 25–27, 2013. – CEUR Workshop Proc., 2013. – Vol. 1032. – P. 122–132.

[6] Garanina N., Sidorova E. An approach to ambiguity resolution for ontology population // Proc. 24th Intern. Workshop CS&P / Rzeszow, Poland, Sep. 28-30, 2015. – Univ. of Rzeszow, 2015. – Vol. 1. – P. 134–145.

[7] Gleich B., Creighton O., Kof L. Ambiguity detection: towards a tool explaining ambiguity sources // Proc. 16th Intern. Working Conf. Requirements Engineering: Foundation for Software Quality / REFSQ 2010, Essen, Germany, June 30–July 2, 2010. – Lect. Notes Comput. Sci., 2010. – Vol. 6182. – P. 218–232.

[8] Kim D.S., Barker K., Porter B.W. Improving the quality of text understanding by delaying ambiguity resolution // Proc. 23rd Intern. Conf. on Computational Linguistics, Beijing, 2010. – P. 581–589.

[9] Navigli R. Word sense disambiguation: a survey // ACM Computing Surveys. – 2009. – Vol. 41, No. 2. – P. 1–69.

[10] Sidorova E., Kononenko I., Zagorulko Yu. Knowledge-based approach to document analysis // Intern. J. "Information Technologies and Knowledge". – 2008. – Vol. 2, No. 1. – P. 17–22.

[11] Spasic I., Zhao B., Jones C., Button K. KneeTex: an ontology-driven system for information extraction from MRI reports // J. Biomedical Sem. – 2015. – Vol. 6. – P. 6–34.

[12] Tel G. Introduction to Distributed Algorithms. – Cambridge University Press, 2000.

[13] Winskel G. The Formal Semantics of Programming Languages: An Introduction. – MIT Press, 1993.

[14] Zhang G.-Q. Chu spaces, concept lattices, and domains // Electronic Notes in Theor. Comput. Sci. (ENTCS). – 2013. – Vol. 83. – P. 287–302.