An Approach to Ambiguity Resolution for Ontology Population *

Natalia Garanina and Elena Sidorova

A.P. Ershov Institute of Informatics Systems,
Lavrent'ev av., 6, Novosibirsk 630090, Russia
{garanina,lsidorova}@iis.nsk.su

Abstract. We study a problem of information retrieval and ambiguity resolution for ontology population. The process for retrieval of information in a form of a set of ontology instances is presented as a Scott information system. This representation proves termination of the process. The Scott information system is a basis for our approach to context-dependent ambiguity resolution. This system generates a multi-agent system in which agents resolve the ambiguity by computing the cardinality of their contexts.

1 Introduction

Ontological databases are currently 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.

We propose to consider the process of ontology population as work with information systems, which are concepts of the domain theory [9]. An information system is a universal model for knowledge (ontologies, thesauri, databases, etc.) organization systems. Information systems are "syntactic" representations of the Scott domains. They are simple, well-studied, and, in the context of ontology population, possess, in particular, useful properties of the entailment relation. In the framework of the algebraic approach to work with ontologies, theory of Formal Concept Analysis is usually applied, which makes it possible to enrich the ontology with new concepts [2]. The Scott domain theory was used for enriching ontologies with topological relations [4].

In this paper, we show that an ontology population system is a Scott information system, which gives an idea of certain general properties of the population process, in particular, a simple proof of termination of the population process and justification of the resolution of context-dependent semantic ambiguity. Features of information retrieval cause ontology population ambiguities. These ambiguities arise when several ontology instances are formed for the same input data. We use the information system of ontology population for resolving context-depending ambiguities. We can evaluate the cardinality of the instance contexts, i.e. how much an instance is related with the

* The research has been supported by Russian Foundation for Basic Research (grant 13-01-00643, 15-07-04144, 13-07-00422) and the People Programme (Marie Curie Actions) of the European UnionŠs Seventh Framework Programme FP7/2007-2013/ under REA grant agreement n PIRSES-GA-2011-294962-COMPUTAL.

other retrieved instances via the information contained in it. Thereby, the information system of ontology population generates a multi-agent system in which agents are ontology instances informationally connected. They are in conflict when there is the ontology ambiguity for them. Agents resolve the conflict by removing the less integrated agents with all their connections from the system. Hence the system is dynamic.

The works on multi-agent systems usually focus on the behavior of agents, methods of communication between agents, knowledge and belief of an agent about environment and other agents, etc [3, 10]. Works about conflict resolution process usually consider the process in terms of the behavior of the agent depending on its internal state, reasoning and argumentation methods etc. [7]. The dynamics of the agents connections is not a subject of these researches. On the other hand, there are studies on social networks, in which agents are connected by typed connections, but their weight is irrelevant [1].

The allied approach to ontology-driven disambiguation which uses a context of a given input text is suggested in [6]. Sequential sentence-by-sentence ambiguity resolution is performed. For every sentence, a set of semantic interpretation is constructed. Each interpretation is represented by a graph with vertices as ontology entities and edges as ontology relations. Every next sentence may increase scores of some interpretation of the previous ones. Hence the preference of alternative meaning of words is based on ontology relations. This approach has low accuracy because such on-the-fly choice of alternative takes into account the nearest context mainly. Our approach uses the maximal context available from an input text for choosing alternative. Connections between ontology objects (class and relation instances) correspond to informational dependance of the text. Since the approach does not require complete sentence parsing, this considerably simplify the text analysis for ontology population.

We propose a multi-agent algorithm for the conflict resolution in the generated multi-agent system of ontology instances. This algorithm is based on an idea of connectivity cardinality of instance agents. On a run of the algorithm agents compute their system weight corresponding to this connectivity. According to these weights, conflicts are resolved by a special agent. This agent adjusts the weight computing processes and constructing a conflict-free set of agents which is an optimal solution of a conflict resolution problem for a given system.

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. The next Section 3, gives definitions for a multi-agent system for ambiguity resolution. Section 4 describes agents of our systems, their action protocols, and the main conflict resolution algorithm. In the concluding Section 5, directions of future researches are discussed.

2 Ontology Population and Scott Information Systems

Let we are given by an ontology of a subject domain, the ontology population rules, semantic and syntactic model for a sublanguage of the subject domain and a data format, and input data with information for population of the ontology. We consider an *ontology of a subject domain* which includes (1) a finite nonempty set of classes for concepts of the subject domain, (2) a finite set of named binary relations on the classes, (3) a finite set of attributes, and (4) a finite set of data types. Every class is defined by a tuple of

typed attributes. Every relation is defined by a couple of classes and a tuple of typed attributes. *Information content* of an ontology is a set of class instances and relation instances defined on these class instances.

Ontology population problem is to compute information content for a given ontology from given input data. A mapping of input data to the ontology has to be a one-to-one correspondence, i.e. different ontology instances correspond to different data fragments. Rules for ontology population and data processing determine ontology instances in input data, and evaluate their attributes. Determination of the information content includes the following stages. First, input data are matched to characters of attributes, classes, and relations of the ontology. These matchings are predefined and depend on an input data format. Initial ontology instances are formed using obtained ontology characters independently from each other. At this stage the instances are not determined completely, because information, which is necessary for evaluating their attributes and establishing relations, frequently places at isolated parts of input data. This information is usually stored in different ontology instances. At the second stage initial instances are formed consistently and completely (as much as possible), i.e. information from some instances could be used for evaluating other and for defining new ontology instances. Rules for ontology population and data processing define informationally connected instances. In paper [5] multiagent algorithm for the second stage is presented. In a process of instance determination, some ambiguities are possible when several ontology instances are formed for the same data fragment. These ambiguities can be caused by homonymy, references, etc. The problem of ambiguity resolution is solved at the third stage. This paper suggests an approach for the solving. At the forth stage ontology population itself is performed. Computed instances could be included into the (possibly nonempty) ontology informational content. At this stage possible inconsistences and ambiguities related to instances in the content should be processed.

We suggest to consider the second stage of the ontology population process as work 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 $Con \subseteq Fin(T)$, $\emptyset \vdash Con$ 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 $\vdash \subseteq Con \times T$ and
  3. $X \in Con$ and $X \vdash a \Rightarrow X \cup \{a\} \in Con$,
  4. $X \in Con$ and $a \in X \Rightarrow X \vdash a$,
  5. $\forall b \in Y : X \vdash b$ and $Y \vdash c \Rightarrow X \vdash c$.

An ontology population system based on an ontology, input data, and the ontology population and the data processing rules, could be defined as a triple $(A_{op}, Con_{op}, \vdash_{op})$. Set of tokens A_{op} is a set of all (underdetermined) ontology instances formed by the rules in the determination process of initial instances:

– class instances $a, \ldots$ of form $(Class_a, Atr_a)$,
  – Atr_a are values of attributes of class $Class_a$ containing:
    grammar information (morphological and syntactic features), and
    structural information (position in input data);

– relation instances $r, \ldots$ of form $(Relation_r, (o_1, o_2)_r, Atr_r)$
  – $(o_1, o_2)_r$ are instances of classes related by the relation $Relation_r$,
  – Atr_r are values of attributes of the relation $Relation_r$ containing grammar and structural information.

For every $X \subseteq A$ let $X = X_C \cup X_R$, where X_C is a set of class instances and X_R is a set of relation instances.

Consistency predicate Con_{op} and entailment relation $\vdash_{op}$ correspond to the rules of ontology population and data processing. Let $x, x' \in A_{op}$, $o \in A_C$, and $X \subseteq A_{op}$. The entailment relation connects informationally associated tokens:

– $X \vdash_{op} x$, iff $x \in X$, or $x \notin X \wedge \exists y \in X_R : x \in O_y$, or $x \notin X \wedge (Atr_x = \emptyset \vee Atr_x = \cup\{\alpha \mid \exists x' \in X : \alpha \subseteq Atr_{x'}\}) \wedge (x \in A_R \rightarrow (O_x = \emptyset \vee \forall o \in O_x : X \vdash_{op} o))$, i.e. instance x is entailed from X, if (1) it is in this set, or (2) it is an object of some relation from this set, or (3) information from tokens of this set is used for evaluating attributes or objects of x, and x does not include other information.

The consistency predicate determines informationally bound sets of tokens:

– $X \in Con_{op}$, if $\exists x \in A_{op} : X \vdash_{op} x$, i.e. *a set of instance-tokens is consistent*, if it entails (new) instance;

– $\emptyset \in Con_{op}$.

Set X is *nontrivial consistent*, if $\exists x \notin X : X \vdash_{op} x$, i.e. the set entails not only its elements and x is *nontrivially entailed*. An *information order* relation $\prec$ is defined on class and relation instances. Let $a, a' \in A_C$ and $r, r' \in A_R$:

– $a \prec a'$, 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';

– $r \prec r'$, if $r = r'$ everywhere except for (1) at least one object and/or (2) at least one attribute, with the number of values of this attribute in r being strictly less than that in r'.

For token $x \in A_{op}$: if $x \prec x'$, then x' is *information extension of* x, and $x^{\uparrow} = \{x\} \cup \{x' \mid x \prec x'\}$, $x^{\downarrow} = \{x\} \cup \{x' \mid x' \prec x\}$ are *upper and down cones of* x.

Theorem 1. *Triple* $(A_{op}, Con_{op}, \vdash_{op})$ *is a Scott information system.*

Proof. Let us show that the consistency predicate Con_{op} and the entailment relation $\vdash_{op}$ satisfy properties 1–5 of information systems.

1. $Y \in Con_{op}$ and $X \subseteq Y \Rightarrow X \in Con_{op}$. For trivial consistent sets the proposition is obvious. Let Y is nontrivial consistent: $Y \vdash_{op} z$, $z \notin Y$. Then $\exists z' \in z^{\downarrow} : X \vdash_{op} z$ or X is trivial consistent, hence $X \in Con_{op}$.

2. $a \in A_{op} \Rightarrow a \in Con_{op}$. $\{a\} \vdash_{op} a$, hence $\{a\} \in Con_{op}$.

3. $X \in Con_{op}$ and $X \vdash_{op} a \Rightarrow X \cup \{a\} \in Con_{op}$. $X \cup \{a\}$ is trivial consistent.

4. $X \in Con_{op}$ and $a \in X \Rightarrow X \vdash_{op} a$. By the def. of the entailment relation.

5. $\forall b \in Y : X \vdash_{op} b$ and $Y \vdash_{op} c \Rightarrow X \vdash_{op} c$. For trivial consistent sets the proposition is obvious. Let sets X and Y are trivial consistent. Set Y includes information from instance-tokens of X only, hence token c does not include information more than in X, hence $X \vdash_{op} c$. Formally: $Y \vdash_{op} c$, if $c \notin Y \wedge (Atr_c = \emptyset \vee Atr_c = \cup\{\alpha \mid \exists b \in Y, X' \subseteq X : \alpha \sqsubseteq Atr_b \sqsubseteq \cup_{x \in X'} Atr_x\}) \wedge (c \in A_R \rightarrow (O_c = \emptyset \vee \forall o \in O_c : o \in Y \vee X \vdash_{op} Y \vdash_{op} o))$, hence $X \vdash_{op} c$.■

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

Proposition 1. *Ontology population process terminates.*

Let us define a closure of an entailment relation: $X \vdash^* x$ iff $\exists y \in A, Y \subseteq A : X \vdash y \wedge Y \cup \{y\} \vdash^* x$. *An information state* of set of token $X \subseteq A$ is all tokens (all information) that can be obtained from this set by the entailment relation: $F(X) = \{x | \forall Y \subseteq Fin(X) \text{ and } Y \vdash^* x)\}$. *A projection of information state* of set X to token $a \in X$ is all tokens that can be entailed from this set using this token: $F(X, a) = \{x | \forall Y \subseteq Fin(X) : a \in Y \text{ and } Y \vdash^* x)\}$. Let a *context* of token a be $F(a) = \{X \mid a \in F(X)\} \cup_{a \in X} F(X, a)$. Intuitively, a context of a is a set of token which includes all tokens necessary for the token entailment and all tokens entailed using the token. In the ontology population framework a context of an instance-token can be used for resolution of context-depending ambiguity. Suppose that there are two instance-tokens that are at the same position of input data. The ontology can be populated with only one of them. Obviously, we prefer the token that is more closely related to the other information obtained from the input data. This is the token that possesses a more powerful context which uses nontrivial entailment relations only.

Hence, the problem of context-depending ambiguity resolution is reduced to the problem of computing the cardinality of contexts for competing instance-tokens. We show that an information system of ontology population generates a special multi-agent system with typed connections. Agents of the system resolve the ambiguities by computing and comparing the context cardinalities.

3 A Multi-agent System of Ambiguity Resolution

Let a set of *maximal determined* instances, which is the result of the analysis of input data, be $A^{\uparrow}_{op} = \{x \in A_{op} \mid x^{\uparrow} = \{x\}\}$. For every $x \notin A^{\uparrow}_{op}$ the corresponding maximal determined instance is $\tilde{x}$ such that $\tilde{x} \in A^{\uparrow}_{op} \wedge x \prec \tilde{x}$.

Entailment relation $\vdash_{op}$ generates *information connections* between maximal determined instances. Let $X \vdash_{op} x$ and $y \in X \wedge y \not\prec x$. Then

– *attribute connections* between $\tilde{y}$ and $\tilde{x}$ are
  – $\tilde{y} \xrightarrow{\alpha} \tilde{x}$ iff $\exists \alpha \in Atr_y : \alpha \in Atr_x$;
  – of *tutorial type* $\tilde{y} \xrightarrow{\alpha^{tut}} \tilde{x}$ iff $\exists x' \in X \ : \ x' \prec x$;
  – of *parental type* $\tilde{y} \xrightarrow{\alpha^{par}} \tilde{x}$ iff $\nexists x' \in X \ : \ x' \prec x$;

– *object connections* between $\tilde{y}$ and $\tilde{x}$ are
  – $\tilde{y} \xrightarrow{y} \tilde{x}$ iff $x \in A_R \wedge (y \in X_C \wedge y \in O_x \vee y \in X_R \wedge \exists o_y \in O_y : o_y \in O_x)$;
  – of *parental type* $\tilde{y} \xrightarrow{y^{par}} \tilde{x}$.

Information system $(A_{op}, Con_{op}, \vdash_{op})$ generates *Multi-agent System of Ambiguity Resolution* (MASAR) as a tuple $S = (A, C, I_C, T_C)$, where

– $A = \{a_x \mid x \in A^{\uparrow}_{op}\}$ is a finite set of agents corresponded to maximal determined instances;

– $C = \{\alpha \mid \exists x, y \in A_{op} : \tilde{x} \xrightarrow{\alpha} \tilde{y}\} \cup \{x \mid \exists x, y \in A_{op} : \tilde{x} \xrightarrow{x} \tilde{y}\}$ is a finite set of connections;

– mapping $I_C : C \longrightarrow 2^{A \times A}$ is an interpretation function of (ordered) connections between agents: $I_C(c) = (a_x, a_y)$ iff $\tilde{x} \xrightarrow{c} \tilde{y}$;

– mapping $T_C : C \times 2^{A \times A} \longrightarrow \{tut, par\}$ is types of connections: $T(c, a_x, a_y) = r$ iff $I_C(c) = (a_x, a_y) \rightarrow (\tilde{x} \xrightarrow{c^r} \tilde{y})$, $r \in \{tut, par\}$.

For every agent $a \in A$ we define the following sets of agents and connections. The similar definitions are used in the graph theory, but we would like to reformulate them for the clarity. We omit symmetric definitions of ancestors $Anc*$ (for $Des*$) and predecessors $Pred*$ (for $Succ*$) for the brevity:

– $C_a = \{c \in C | \exists a' \in A : (a, a') \in I_C(c) \bigvee (a', a) \in I_C(c)\}$ is connections of a;
– $Des_a^c = \{a' \in A \mid (a, a') \in I_C(c)\}$ is a set of descendants by c connection;
– $Des_a = \bigcup_{c \in C_a} Des_a^c$ is a set of descendants;
– $Succ_a^c = Des_a^c \cup \bigcup_{a' \in Des_a^c} Succ_{a'}^c$ is successors by c connection;
– $Succ_a = \bigcup_{c \in C_a} Succ_a^c$ is a set of successors.

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 $Succ_a^c \cap Pred_a^c = \emptyset$, i.e. every connection has no cycle because of information transfer. Note, that in MASAR every agent has one ancestor at the most. The weight functions for agents should correspond to information worth of agents and their connectivity with other agents. Let $Prt \in \{Des, Anc\}$. For every $a \in A$ mapping

– $wt_{Prt}^a : C \longrightarrow \mathbb{N}$ is *the weight function of connection descendants (ancestors)* $wt_{Prt}^a(c) = \sum_{a' \in Prt_a^c} wt_{Prt}^{a'}(c) + 1$, and $wt_{Prt}(a) = \sum_{c \in C_a} (wt_{Prt}^a(c) - 1)$;

– $wt : A \longrightarrow \mathbb{N}$ is *the weight function of information agents* defined by $wt(a) = wt_{Des}(a) + wt_{Anc}(a) + 1$.

Weight of system S is $wt(S) = \sum_{a \in A} wt(a)$.

Let *conflict set* $Conf \subseteq A \times A$ be a set of unordered conflict pairs of agents corresponded to informational objects formed for the same data fragments. For every pair of agents from $Conf$ we say that *a conflict is resolved* if one of the agents in the pair called *the minor agent* deletes itself from A (with all its connections) and involves all its descendants. Every descendant involved performs the following *conflict actions*:

(1) if the involving connection is of the parental type then it deletes itself from A (with all its connections) and involves all its descendants;

(2) if the involving connection c is of the tutorial type then it deletes all outgoing connections c, and involves its descendants connected by the connections.

Note that all successors of the minor agents are involved in its conflict resolution. A conflict pair of agents is deleted from $Conf$ after conflict resolution. Conflict actions decrease the weight of all successors and predecessors of a conflicting agent. The first conflict action reduces the conflict set and the set of agents in MASAR. Hence the system is dynamic due to conflict resolution. Change of the system weight with the fixed weight function depends on a policy of conflict actions for every agent. *Problem of conflict resolution in MASAR* is to get a conflict-free MASAR of the maximal weight. We develop a multiagent algorithm that produces such system.

4 Conflict Resolution in MASAR

For constructing the conflict-free multiagent system of the maximal weight by resolving a chain of conflicts we should know how much each conflict resolution step affects to

the system weight. For every agent in conflict, it is necessary to compute its *conflict weight* which is the difference between the system weight before and after the agent conflict resolution. Distributed computing of this weight takes polynomial time.

Action protocols for conflict resolution used by MASAR agents forms a multi-agent system of conflict resolution MACR. The system MACR includes set of MASAR agents and an agent-master. Note, that a fully distributed version of our algorithm could be developed but it should be very ineffective. The result of agent 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 ancestors and descendants. Messages are transmitted instantly via a reliable medium and stored in channels until being read.

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

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

Our algorithm for constructing a conflict-free MASAR of the maximal weight is a greedy algorithm. At every step it chooses for resolution a conflict which has the maximal effect to the system weight. This effect depends on conflict actions of involved agents. Hence the following algorithms should be implemented: calculating of agents' weights, calculating of agents' conflict weights, the main algorithm for constructing a conflict-free set of agents of the maximal weight. Calculating the weights should be performed by MASAR agents, but constructing a conflict-free set should be conducted by the master agent.

We define an interface protocol `Ai` for system agents, which specifies agent's reactions for incoming messages. These messages include information which protocol `Act` (to perform a conflict action) or `ChangeWeight` (to change its weight) should be run and their parameters described below at the protocols' definitions. Until an input message cause an agent to react the agent stays in a wait mode.

Interface protocol of agent a.

```
Ai (a) ::
   set of msg Input; msg mess=(start,∅);
1.   while ( mess.act != Stop )
2.     if( Input != null ) then {
3.       mess = get (Input);
4.       if( mess.act = ToAct ) then Act(mess);
5.       if( mess.act = ToChange ) then ChangeWeight(mess);}
```

(1) The main algorithm for conflict resolution

Let us give an informal description of protocol `Master`. Let Del_A be set of agents removed from A due to particular conflict resolution, $PartConf_a = \{b \in A | (a, b) \in Conf\}$ be a set of agents in conflict with a and $Conf_A = \{a \in A | \exists b \in A (a, b) \in Conf\}$ be a set of agents in a conflict. Until the latter set becomes empty the following

steps repeat: (1) to compute of agents' weights by launching agents to perform protocols `WeightCount` in parallel, (2) to compute of agents' conflict weights by launching conflict agents to perform protocols `Start` in sequence, (3) to find the minor partner of the agent of maximal impact for the system weight, with the maximal difference in their conflict weights, (4) to change the weights of agents involved by this agent by launching the minor agent to perform protocol `Start`, (5) to remove the conflict of the agent and conflicts of deleted agents from the conflict set and (6) to recalculate the set of agents in conflicts. We consider the master can detect termination moments of other agents' parallel computations at every step. The protocol of conflict weights computing and weights changing belongs to the class of wave echo algorithms [8]. Let function `max_wConf(`X`)` (`min_wConf(`X`)`) returns the agent of the maximal (minimal) conflict weight in set of agents X.

Protocol of the master agent for conflict resolution.

```
Master ::
   agent a, b;
1.   while ( Conf_A ≠ ∅ ){
2.      forall a∈ A in_parallel WeightCount(a);
3.      forall a∈ Conf_A in_sequence Start(a, true);
4.      a = max_wConf( Conf_A ); b = min_wConf( PartConf_a );
5.      Start(b, false);
6.      Conf = Conf \ {(a, b)};
7.      forall c∈ Del_A Conf = Conf \ {(c, d) | d ∈ PartConf_c};
8.      recalculate(Conf_A);}
9.   forall a∈ A in_parallel send ( Stop ) to a;
```

(2) Computing agents' weight

Let the set of connection of agent a is $C_a = \{c_1, \ldots, c_n\}$. Following the definitions of the weights the agent launches in parallel calculations of the sum weight by every connection c_i for successors `CiDes` and for predecessors `CiAnc` (line 1) and stores calculated weights in arrays `w_Des` and `w_Anc` respectively. When these parallel calculations are finished, the agent computes its own weight (lines 2–3). The calculation processes have local channels `Input` for messages with integer weights of successors (predecessors). They send the weights increased by 1 to predecessors (successors) respectively. We omit the similar description of predecessors' processes `CiAnc` for the brevity. All these agent's weights are accessible to the agent for changing in its other protocols.

Protocol of agent a for weight compute.

```
WeightCount (a) ::
   array [n] of int: w_Des, w_Anc; int w_Des_own, w_Anc_own;
1.   parallel {C1Des} {C1Anc} ... {CnDes} {CnAnc}
2.   w_Des_own = ∑_{i∈[1..n]} w_Des[i]; w_Anc_own = ∑_{i∈[1..n]} w_Anc[i];
3.   wt_A = w_Des_own + w_Anc_own + 1;
CiDes() ::
   set of int Input; NumD = |Des_a^{c_i}|;
1.   w_Des[i] = 0;
2.   while( NumD != 0 )
```

```
3.     if ( Input != null ) then {
4.           w_Des[i] = w_Des[i] + get( Input ); NumD = NumD - 1;}
5.   forall (b in Anc_a^{c_i}) send w_Des[i]+1 to b;
```

(3) Computing agents' conflict weight

The next protocol is performed by the agent which starts to compute its conflict weight (`wc=true`) or is the minor agent (`wc=false`). It performs conflict action 1 launching a wave of weight changing of successors and predecessors and initiates actions of involved agents in protocol `Act` in line 2. Before the wave starts, agents restore its initial weights and the presence status in line 1. If there is real system change (not conflict weight computing) then all agents change their weight in line 3. Let $\mathtt{wt}^a$ be integer temporal weight of agent a, and boolean variable a.`Rmvd` be true if a is removed by its conflict action 1, and false in other case.

Protocol of initial agent a

```
Start(a, wc) ::
1.   forall b ∈ A wt^b=wt(b); b.Rmvd = false;
2.   Act(1, ∅, ∅, wc);
3.   if (not wc) forall b ∈ A wt(b)=wt^b;
```

An input for protocol `Act` is a message of the form `mess = (ct,` x`,` c`, wc)`, where `ct` is conflict action type, x is an agent which activate this action, c is a connection with this agent, and `wc` is true if a conflict weight is computing. In this protocol (lines 1–2) an agent depending on the type of its conflict action (1) determines the difference of own weight, (2) forms sets of descendants and ancestors which weights are changing due to this action, and (3) specifies the amount of these changes in variable `w`. Then the agent sends the corresponding messages to the partners (lines 7) launching a wave of weight changing of its successors and predecessors and waits when it finishes (line 8). Further the agent depending on its conflict type launches a wave of conflict actions of its successors (lines 9,10) and waits when it finishes (line 11). The agent has local channel `Input` for messages with integer conflict weights of involved successors. In line 13 the agent sums these weights. Let function `i(c)` returns index of connection c in set C_a for agent a, $ActId = \{1, 2\}$ be a set of indexes of conflict actions, and $C_1^a = \{a' \mid \exists c : T(c, a, a') = par\}$ and $C_2^a = \{a' \mid \exists c : T(c, a, a') = tut\}$ be sets of agents immediately involved in a conflict by agent a to perform action 1 or 2.

Protocol of agent a **for conflict actions.**

```
Act( mess = (ct, x, c, wc) ) :: {
   int w, wConf, wConfTmp; agent b; connection c;
   set of agents Desa, Anca; set of int Input; ActId i;
1.   if ( ct = 1 ) then Desa = Des_a; Anca = Anc_a; wConf = wt_A;
2.   if ( ct = 2 ) then Desa = Des_a^c; Anca = Anc_a^c;
                        wConf = w_Des[i(c)]+w_Anc[i(c)];
3.   forall b∈Desa∪Anca {
4.        c = (a,b);
5.        if b∈Desa then Rel = Anc; w = w_Anc[i(c)];
6.                  else Rel = Des; w = w_Des[i(c)];
7.        send (ToChange, delMe, a, c, w, Rel) to b;}
8.   wait (doneWt) fromall b;
```

```
9.   while( Input != ∅ ) wConf = wConf + get( Input );
10.  if( ct = 1 ) then forall i∈ActId forall b∈ C_i^a
                              send (ToAct, i, a, (a,b), wc) to b;
11.  if( ct = 2 ) then forall i∈ActId forall b∈ C_i^a ∩ Des_a^c
                              send (ToAct, i, a, c, wc) to b;
12.  wait (doneAct) fromall b;
13.  if ( wc ) then {
14.       while( Input != null ) wConf = wConf + get( Input );
15.       if ( x != null ) then send (doneAct, wConf) to x;}
16.  else A = A \ {a}; }
```

An input for the next protocol is a message of the form `mess = (act, x, c, w, Rel)`, where `act` specifies should the agent a remove (`act=delMe`) agent x from its ancestors (`Rel = Anc`) or descendants (`Rel = Des`) by connection `c` (lines 2,5). The agent decreases its corresponding weights by `w`. Decreasing of the weight affects weights of its successors and predecessors. The agent initiates the changing of these weights in line 8 and waits when it finishes (line 9).

Protocol of agent a for its weight changing.

```
ChangeWeight( mess = (act, x, c, w, Rel) ) :: {
   int w; agent b; set of agents Parts;
1.   if( Rel = Anc ) then w_Anc[i(c)] = w_Anc[i(c)] - w;
2.                        if( act = delMe ) then Anc_a^c = Anc_a^c \ {x};
3.                        Parts = Des_a^c;
4.                   else w_Des[i(c)] = w_Des[i(c)] - w;
5.                        if( act = delMe ) then Des_a^c = Des_a^c \ {x};
6.                        Parts = Anc_a^c;
7.   wt^a = wt^a - w;
8.   forall b∈Parts send (ToChange, decMe, a, c, w, Rel);
9.   wait (doneWt) fromall b∈Parts;
10.  while( Input != ∅ ) wt^a = wt^a + get( Input );
11.  send (doneWt, wt^a) to x; } }
```

Consider now the performance of the conflict resolution algorithm in a particular case of ambiguity in the following sentence:

On October 22, 2013, an official ceremony was held in the Nenets Autonomous District to mark the start of pilot oil production at the A. Titov field.

We consider *Energetics* as an ontology subject domain. Thesaurus of this subject area among others should contain single-word terms *pilot, oil* and *production* together with multi-word terms *pilot oil* and *oil production*. Thus the ambiguity in the example above is the following:

[[pilot [oil production]] ⟷ [[pilot oil] production]]

During the multiagent algorithm initialization for the above sentence the following lexical objects L1–L5 is created with semantic attributes from the thesaurus (see Fig. 1). As a result of main stage of multiagent algorithm by the means of rule-agents implementing search of information concerning activities related to the oil production, an informational agents I1–I4 and R1–R3 corresponding to the ontological classes and relations is created.

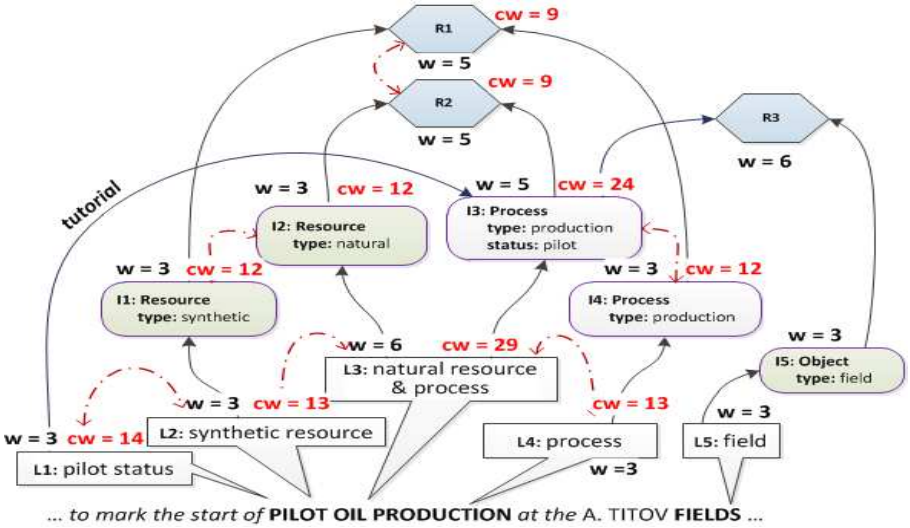

Fig. 1. An example of conflicting agents.

Thus the main stage of the analysis in our example results in the following list of ambiguity conflicts: (L1,L2), (L2,L3), (L3,L4), (I1,I2), (I3,I4), (R1,R2). Calculated weights of agents are also depicted at Fig.1. The conflict resolution algorithm deletes agents L2, I1, and R1 at the first iteration, and L4 and I4 at the second one. The result of the algorithm is the set of information agents I2, I3, I5, R2, and R3. Thereby all remaining conflicts are resolved automatically.

5 Conclusion

In this paper, we show that instances of the ontology classes and relations that take part in the process of population form, together with the rules of data processing and ontology population, a Scott information system. This results in a simple proof of termination of the population process and justification of the resolution of context-dependent ambiguity of instances by calculating their context cardinalities. The Scott information system is a basis for our approach to context-dependent ambiguity resolution. This system generates a multi-agent system in which agents resolve the ambiguity by computing the cardinality of their contexts. The suggested algorithm of ambiguity resolution chooses the most powerful agents and removes their competitors. The choice is based on agents' weights and their impact on the system in the process of ambiguity resolution.

In the near future we plan to give formal proofs of correctness of the algorithm proposed and to estimate its time complexity. In a development process of our multi-agent system of the semantic analysis of natural language texts for ontology population, we intend to carry out integrated testing and to rate quality of these algorithms in terms of completeness and soundness.

References

1. Bergenti F., Franchi E., Poggi A.: *Selected models for agent-based simulation of social networks* // In: Procs. 3rd Symposium on Social Networks and Multiagent Systems (SNAMAS 2011) 2011, pp. 27-32.
2. Cimiano P., Hotho A., Stumme G., Tane J.: *Conceptual Knowledge Processing with Formal Concept Analysis and Ontologies* // Proc. of 2nd Conference on Formal Concept Analysis (ICFCA 2004). LNCS Vol. 2961, 2004, pp 189-207.
3. Fagin R., Halpern J.Y., Moses Y., Vardi M.Y.: *Reasoning about Knowledge.* MIT Press, 1995.
4. Fernhdez-Breis J.T., et all: Towards Scott Domains-Based Topological Ontology Models: An Application to a Cancer Domain // In Proc. of the Conference on Formal Ontology in Information Systems (FOIS '01). ACM, 2001, p. 127–138.
5. Garanina N., Sidorova E., Bodin E.: *A Multi-agent Approach to Unstructured Data Analysis Based on Domain-specific Onthology* // Proc. of the 22nd International Workshop on Concurrency, Specification and Programming, Warsaw, Poland, Sept. 25-27, 2013. CEUR Workshop Proceedings, Vol-1032, p. 122-132
6. Kim D.S., Barker K., Porter B.W.: *Improving the Quality of Text Understanding by Delaying Ambiguity Resolution* // Proc. of the 23rd International Conference on Computational Linguistics (Coling 2010), Beijing, August 2010. pp. 581Ű-589
7. Huhns M. N., Stephens L. M.: *Multiagent Systems and Societies of Agents* // In: Multiagent Systems, MIT Press, 1999 pp. 79–120.
8. Tel G.: *Introduction to Distributed Algorithms.* Cambridge University Press, 2000.
9. Winskel G.: *The Formal Semantics of Programming Languages: An Introduction.* MIT Press, 1993.
10. Wooldridge, M.: *An Introduction to Multiagent Systems.* Willey&Sons Ltd, 2002.