УДК 004.052, 519.179.2

A Verification Method for a Family of Multi-agent Systems of Ambiguity Resolution»

Natalia Garanina (A.P. Ershov Institute of Informatics Systems),

Elena Sidorova (A.P. Ershov Institute of Informatics Systems)

In the paper we describe a verification method for families of distributed systems generated by context-sensitive network grammar of a special kind. The method is based on model checking technique and abstraction. A representative model depends on a specification grammar for family of systems. This model simulates a behavior of the systems in such a way that properties which hold for the representative model are satisfied for all these systems. We show using this method for verification of some properties of multiagent system for resolution of context-dependent ambiguities in ontology population.

Keywords: model checking, context-sensitive network grammar, multi-agent systems, abstraction

1. Introduction

The motivation of our work is the ambiguity resolution problem in the frame of ontology population from natural language texts. In [6] we describe text analysis algorithms producing a system of information agents. But features of natural language cause ontology population ambiguities, which these agents have to resolve. We proposed to evaluate the cardinality of agents' contexts, i.e. how much an agent is related with the other agents of the resulting system via the information contained in it, and to mark the agents the most integrated in the text. We developed an ambiguity resolution algorithm [5], removing the less integrated agents from the system.

All agents in parallel perform rather complicate protocols with periodic local synchronizations. Hence, it is reasonable to use formal verification methods for proving correctness of the algorithm. We choose model checking technique for a particular multi-agent system. We verify rather specific multi-agent system of conflict resolution. 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 [4, 9]. 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. [8]. The dynamics of the agents connections is not a subject of these researches. There are papers related to the dynamics of weighted connections, but they are not the typed and their changes does not affect the internals of the agent [7]. On the other hand there are works on the study of social networks, in which the agents are connected by the typed connections, but their weight does not matter [1]. To the best of our knowledge, there are no works on model checking for a conflict resolution algorithm of the suggested type.

Model checking technique is widely used for verification of distributed and multiagent systems [2]. In our case we would like to verify not a particular agent network, but infinite family of such systems. For verification of infinite network families the model checking method was suggested in [3]. This method is based on using a context-free network grammar generating families of distributed systems, and on abstraction by finite automata. The idea of the method is to construct an invariant network based on a given grammar. This invariant simulates behavior of all systems in the family and is consistent with abstract functions associated with properties to be verified which are expressed by branching time logic $\forall CTL$. Due to consistent simulation, properties holding for the representative invariant also holds for all systems in the family. But authors studied context-free grammars only, while our model of the multiagent system is generated by a context-sensitive grammar of a special kind. In the paper we define such network grammar by adding notions of a quasi-terminal and a merging operator to the standard definition. We show that this verification method still can be used for network families generated by the new grammar.

The rest of the paper is organized as follows. The next section 2 gives base definitions. Section 3 presents results on a new merging operator, used in our context-sensitive network grammar. Section 4 describes using our method for the multiagent system of ambiguity resolution. We conclude in the last section 5 with a discussion of further research.

Acknowledgments. The research has been supported by Russian Foundation for Basic Research (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”).

2. Base Definitions

Let us give necessary definitions from [3] in a modified form. Modification concerns a merging operator and quasi-terminals in a network grammar.

Definition 1.

A Labeled Transition System (LTS) is a structure $M = (S, R, ACT, S_0)$, where

- S is the set of states,
- $S_0 \subseteq S$ is the set of initial states,
- ACT is the set of actions, and
- $R \subseteq S \times ACT \times S$ is the total transition relation, such that for every $s \in S$ there is action a and state s' for which $(s, a, s') \in R$ (denote as $s \xrightarrow{a} s'$).

Let L_{ACT} be the class of LTSs whose set of actions is a subset of ACT. Let $L_{(S,ACT)}$ be the class of LTSs whose state set is a subset of S and whose action set is the subset of ACT. Let $ACT_1, ACT_2 \subseteq ACT$. Let we are given two LTSs $M_1 = (S_1, R_1, ACT_1, S_0^1)$ and $M_2 = (S_2, R_2, ACT_2, S_0^2)$ in the class L_{ACT}.

Definition 2.

A function $\|: L_{ACT} \times L_{ACT} \mapsto L_{ACT}$ is called a *composition function* iff $M_1 \parallel M_2$ has the form $(S_1 \times S_2, R', ACT_1 \cup ACT_2, S_0^1 \times S_0^2)$.

A function $\cup : L_{ACT} \times L_{ACT} \mapsto L_{ACT}$ is called a *merging function* iff $M_1 \cup M_2$ has the form $(S_1 \cup S_2, R', ACT_1 \cup ACT_2, S_0^1 \cup S_0^2)$.

The definition of R' depends upon the exact semantics of the composition and merging function. Let S^i be words of length i with S as the alphabet.

Definition 3.

Given a state set S and a set of actions ACT, any subset of $\bigcup_{i=1}^{\infty} L_{(S^i,ACT)}$ is called a network on the tuple (S, ACT).

We give a definition of a *context-sensitive network grammar with quasi-terminals (CSNQ-grammar)* to describe networks, which is the modified definition of a context-free network grammar from [3]. The set of all LTSs derived by a network grammar forms a network which is an LTS also. Let S be a state set and ACT be a set of actions. *CSNQ-grammar* $G = (T, Qt, t, N, P, S)$ is a grammar, where

- T is a set of terminals, each of which is an LTS in $L_{(S,ACT)}$, these LTSs are sometimes referred to as basic processes,

- Qt is a set of quasi-terminals, each of which is an LTS in $L_{(S,ACT)}$, their merging gives an LTS,
- a mapping $t : Qt \mapsto T$ associates quasi-terminals to terminals,
- N is a set of nonterminals, each nonterminal defines a network,
- P is a set of production rules of the following forms:
  - $A \longrightarrow B \parallel_i C$, where $A \in N$, and $B, C \in T \cup Qt \cup N$, and $\parallel_i$ is a composition function.
  - $\{A_1, ...A_n\} \longrightarrow t(A_1) \cup_i ... \cup_i t(A_n)$, where $A_j \in Qt$, and $\cup_i$ is a merging function.
- $S \in N$ represents the network generated by the grammar.

Note, that this grammar is context-free with respect to composition functions and context-sensitive with respect to merging functions.

In order to express properties of a model composed from finite, but unspecified number of LTSs, we define a finite automaton on alphabet S.

Definition 4.

$D = (Q, q_0, \delta, F)$ is a *deterministic automaton* over S, where

- Q is the set of automaton states,
- $q_0 \in Q$ is the initial state,
- $\delta \subseteq Q \times S \times Q$ is the transition relation,
- $F \subseteq Q$ is the set of accepting states, and
- $L(D) \subseteq S^*$ is the set of words accepted by D.

We use finite automata over S for specification of atomic state properties. Let D be an automaton over S. State s satisfies D ($s \models D$) iff $s \in L(D)$. A specification language is a universal branching temporal logic $\forall CTL$ [3] with finite automata over S as the atomic formulas. Syntax of $\forall CTL$ consists of formulas that are composed of Boolean constants, atomic formulas, connectives $\neg$, $\vee$, $\wedge$, and branching time modalities $\mathbf{AX}\varphi$, $\mathbf{AG}\varphi$, and $\varphi\mathbf{AU}\psi$ with standard semantics.

Recall definitions for abstract LTS from [3]. For the simplicity, here the specification language contains a single atomic formula D. Given an automaton $D = (Q, q_0, \delta, F)$ and a word $w \in S^*$ the function induced by w on Q, $f_w : Q \mapsto Q$, is defined by $f_w(q) = q'$ iff $q \xrightarrow{w} q'$. Note that $w \in L(D)$ if and only if $f_w(q_0) \in F$. Two states s and s' are *equivalent* $s \equiv s'$ iff $f_s = f'_s$. The function f_s is called the *abstraction* of s and is denoted by $h(s)$. Relation $\models$ is extended to abstract states: $h(s) \models D$ iff $f_s(q_0) \in F$. Hence $s \models D$ iff $h(s) \models D$.

Let F_D be the set of functions corresponding to the deterministic automaton D. The abstraction function h extended to F_D is defined by $h(f) = f$ for $f \in F_D$ and extension the function h to $(S \cup F_D)$ is $h((a_1, a_2, ..., a_n)) = h(a_1) \circ ... \circ h(a_n)$. From now on we consider LTSs in the network N on the tuple $(S \cup F_D, ACT)$.

Definition 5. *(of abstract LTS)*
Given an LTS $M = (S^i, R, ACT, S_0)$ in the network N, the corresponding abstract LTS is defined by $h(M) = (S^h, R^h, ACT, S_0^h)$, where

- $S^h = \{h(s) | s \in S^i\}$ is the set of abstract states,
- $S_0^h = \{h(s) | s \in S_0\}$, and
- the relation R^h is defined as follows. For any $h_1, h_2 \in S^h$, and $a \in ACT$:
  $(h_1, a, h_2) \in R^h \Leftrightarrow \exists s_1, s_2 [h_1 = h(s_1) \wedge h_2 = h(s_2) \wedge (s_1, a, s_2) \in R]$.

M' *simulates* M (denoted $M \preceq M'$) iff there is a simulation preorder $E \subseteq S \times S'$ ($(s, s') \in E$ denoted $s \preceq s'$) that satisfies the following conditions: for every $s_0 \in S_0$ there is $s_0' \in S_0'$ such that $s_0 \preceq s_0'$. For every s, s', if $s \preceq s'$ then

- $h(s) = h(s')$, and
- for every s_1 such that $s \xrightarrow{a} s_1$ there is s_1' such that $s' \xrightarrow{a} s_1'$ and $s_1 \preceq s_1'$.

3. The Merging Operator in the Verification Framework

The first two propositions of the following lemma were proved in [3], the last is proved below:

Lemma 1.

1. $M \preceq h(M)$, i.e., $h(M)$ simulates M.
2. If $M \preceq M'$, then $h(M) \preceq h(M')$.
3. $M \cup M' \preceq h(M) \cup h(M')$

Proof of (3) is obvious: $M \cup M' \preceq h(M \cup M')$ due to (1), and $h(M \cup M') = h(M) \cup h(M')$.□

The following theorem about satisfiability of properties in an LTS and its simulator was proved in [3] and holds for our new framework.

Theorem 1.
Let φ be a formula in $\forall CTL$ over the atomic formula D. Let M and M' be two LTSs such that $M \preceq M'$. Let $s \preceq s'$. Then $s' \models \varphi$ implies $s \models \varphi$.

Definition 6.

A merging or composition operator $\bullet \in \{\cup, \|\}$ is called *monotonic* with respect to a simulation preorder $\preceq$ if and only if given LTSs such that $M_1 \preceq M_2$ and $M_1' \preceq M_2'$, we have that $M_1 \bullet M_1' \preceq M_2 \bullet M_2'$. A network grammar G is called monotonic if and only if all rules in the grammar use only monotonic composition and merging operators.

We modify a synchronous framework from [3] with results for the merging operator. Let models be a form of LTSs, Moore machines $M = (S, R, I, O, S_0)$ such that inputs I and outputs O must be disjoint. In addition, they have a special internal action denoted by τ. The set of actions is $ACT = \{\tau\} \cup 2^{I \cup O}$, where each noninternal action is a set of inputs and outputs. A transition $s \xrightarrow{a} t$ from s in a machine M with $a = i \cup o$ such that $i \subseteq I$ and $o \subseteq O$ occurs only if the environment supplies inputs i and the machine M produces the outputs o.

Naturally, for the merging operator inputs and outputs of merging machines must be disjoint also. Let $I \cap O' = \emptyset$ and $O \cap I' = \emptyset$. The merging of M and M', $M'' = M \cup M'$ is defined by

- $S'' = S \cup S'$,
- $S_0'' = S_0 \cup S_0'$,
- $I'' = I \cup I'$ and $O'' = O \cup O'$, and
- $s'' \xrightarrow{a''} s_1''$ is a transition in R'' iff the following holds: $s'' \xrightarrow{a} s_1''$ is a transition in R and $s'' \xrightarrow{a'} s_1''$ is a transition in R' for some a, a' such that $a'' = a$ or $a'' = a'$.

Lemma 2.

The merging $\cup$ is monotonic with respect to $\preceq$.

Proof. Let $M = (S, R, I, O, S_0)$, $M_1 = (S_1, R_1, I_1, O_1, S_{1,0})$, $M' = (S', R', I', O', S_0')$, $M_1' = (S_1', R_1', I_1', O_1', S_{1,0}')$ be four Moore machines. Assume that $M \preceq M_1$ and $M' \preceq M_1'$. Let $E \subseteq S \times S_1$ and $E' \subseteq S' \times S_1'$ be the corresponding simulation relations. We prove that $M \cup M' \preceq M_1 \cup M_1'$.

We say that $(s'', s_1'') \in E''$ iff $(s'', s_1'') \in E$ or $(s'', s_1'') \in E'$. We show that E'' has the required properties. It is clear from the definition that given state $s_0 \in S_0 \cup S_0'$, there exists $s_{0,1} \in S_{0,1} \cup S_{1,0}'$ such that $(s_0, s_{0,1}) \in E \cup E'$.

Assume that $(s, s_1) \in E \cup E'$.

(1) By assumption, we have that $h(s) = h(s_1)$.

(2) Let $s \xrightarrow{a''} t$ be a transition in $M \cup M'$. This means that there exists transition $s \xrightarrow{a} t$ in M or transition $s \xrightarrow{a'} t$ in M' such that $a'' = a$ or $a'' = a'$. By definition there exists $t_1 \in S_1 \cup S_1'$

such that $s_1 \xrightarrow{a} t_1$ or $s_1 \xrightarrow{a'} t_1$, where $(t, t_1) \in E$ or $(t, t_1) \in E'$. Therefore, $s_1 \xrightarrow{a''} t_1$ and $(t, t_1) \in E''$. The proof is thus complete. □

The notion of a representative give us a way to construct a simulation invariant. Given a CSNQ-grammar G, we associate with each symbol A of the grammar *a representative process* $rep(A)$. Let us adopt the definition of a monotonicity property for a set of representative processes of CSNQ-grammar:

- for every terminal and quasi-terminal A: $h(rep(A)) \succeq h(A)$, and
- for every rule $A \longrightarrow B \parallel C$: $h(rep(A)) \succeq h(h(rep(B)) \parallel h(rep(C)))$.

We extend the proof of the following theorem on context-free network grammar from [3] to CSNQ-grammars:

Theorem 2.

Let G be a monotonic grammar and suppose we can find representatives for the symbols of G that satisfy the monotonicity property. Let A be a symbol of the grammar G, and let a be an LTS derived from A using the rules of the grammar G. Then, $h(rep(A)) \succeq a$.

Proof. We prove that $h(rep(A)) \succeq h(a)$. Since $h(a) \succeq a$, the result follows by transitivity. Let $A \Rightarrow^k a$, i.e., A derives a in k steps. Induction on k.

($k = 0$) Proved in [3].

($k = 1$) In the case A, B are quasi-terminals in a rule $A, B \longrightarrow t(A) \cup t(B)$ and $a = t(A) \cup t(B)$. The result follows from the monotonicity property and Lemma 1.

($k \geq 1$) Proved in [3]. □

Verification method is exactly the same as in [3]. Assume that we are given monotonic grammar G and $\forall CTL$ formula φ with atomic formulas $D_1, ..., D_k$. To check that every LTS derived by the grammar G satisfies φ we perform the following steps:

1. For every symbol A in G choose representative process $rep(A)$ and construct the abstract LTS $h(rep(A))$ with respect to the formulas $D_1, ..., D_k$.
2. Check that the set of representatives satisfies the monotonicity property. Theorem 2 implies that for every a derived by the grammar G, $h(rep(S)) \succeq a$.
3. Perform model checking on $h(rep(S))$ with specification φ. By Theorem 1, if $h(rep(S)) \models \varphi$, then for all LTSs M derived by the grammar G, $M \models \varphi$.

For finding monotonic representatives we could use an algorithm from [3] setting $\{t(A)\}$ as an initial representative association set of every quasi-terminal A.

4. Verification of Multiagent Ambiguity Resolution

A detailed description of the multiagent algorithm for ambiguity resolution in ontology population is given in [5]. In this paper we sketch a communication structure without considering agents' actions on message processing.

Let a set of *information agents* be given. Some of agents are *in a conflict* corresponding to some ambiguity. An *agent-master* constructs a conflict-free set of information agents taking into account integration of conflict agents in the system. This integration is evaluated by computing weights and conflict weights of the agents. A conflict is resolved by removing a weak agent from the system. The agent-master performs the main protocol of constructing the conflict-free set, while the information agents perform protocols of computing their weights.

Every information agent is connected with the master by two-way channel. Information agents are linked with others by labeled connections of two types corresponding their conflict reaction: removing (*rem*-type) and updating (*upd*-type). Every labeled connection is acyclic. Processing of every conflict reaction induced by specified connection is considered to be certain base process. An information agent can be union of such processes. This fact specifies a form of a grammar generating family of our multiagent systems for various number of agents connected in various ways. Agents are connected by two-way channels corresponding to these labeled connections. This structure of multiagent network is generated by the following context-sensitive grammar with quasi-terminals $G = (T, Qt, t, N, P, S)$. Let a set of connections be $C = \{c_1, ..., c_n\}$ and c_i^k be a connection having conflict type $k \in \{rem, del\}$.

- terminals $T = \{master\} \cup \bigcup_{i=1}^{n} \{root_i, inter_i, leaf_i\} \cup vrtxs^i$, and $vrtxs^i = \{vrtx \mid vrtx = inter_j$ or $vrtx = leaf_j, j \in [1..n]\}$ and $|vrtxs^i| = i$,
- quasi-terminals $Qt = \bigcup_{i=1}^{n} \{INTER_i, LEAF_i\}$,
- associate mapping $t : Qt \mapsto T$ is defined by $t(INTER_i) = inter_i$, and $t(LEAF_i) = leaf_i$ for every $i \in [1..n]$,
- nonterminals $N = \{S\} \bigcup_{i=1}^{n} \{ROOT_i, SUB_i\}$;
- set of production rules P for every $i \in [1..n]$:
  1. $S \longrightarrow master \parallel_m ROOT_1 \parallel_m \ldots \parallel_m ROOT_n$
  2. $ROOT_i \longrightarrow (ROOT_i \parallel_{c_i^k} SUB_i) \bigvee (root_i \parallel_{c_i^k} SUB_i)$
  3. $SUB_i \longrightarrow (SUB_i \parallel_{c_i^k} SUB_i) \bigvee (INTER_i \parallel_{c_i^k} SUB_i) \bigvee$
     $(SUB_i \parallel_{c_i^k} LEAF_i) \bigvee (INTER_i \parallel_{c_i^k} LEAF_i) \bigvee$
     $(inter_i \parallel_{c_i^k} SUB_i) \bigvee (SUB_i \parallel_{c_i^k} leaf_i) \bigvee$

$$(inter_i \parallel_{c_i^k} LEAF_i) \bigvee (INTER_i \parallel_{c_i^k} leaf_i) \bigvee (inter_i \parallel_{c_i^k} leaf_i)$$

4. $\{V_1, ..., V_m\} \longrightarrow t(V_1) \cup ... \cup t(V_m) = vrtx^m$, where for every $j \in [1..m]$ $V_j \in \{INTER_i, LEAF_i\}$, and if $V_j = INTER_i$ then for every $l \in [1..m]$ holds $V_l \neq LEAF_i$ $(i \in [1..n])$.

Parallel composition of agent-processes is synchronous. Protocols for computing weights and conflict weights are highly parallel. Hence it is very important to prove that they terminate and are synchronized properly. Satisfiability of these properties is necessary for correctness of weight computing. Launch of these computing could be modeled by sending tokens.

Every base process is defined by the following state variables:

- $Name:$ *int* is a name of the process;
- *Channel*: set of $\{name:$ *int*; $c\_type:$ *bool*; $dir:$ *bool*; $agn:$ *int*; $rmvd:$ *bool*$\}$, where *name* is a label of a connection, *c_type* is its type, *dir* is a direction: a child $(dir = 0)$ or a parent $(dir = 1)$ named *agn*, and *rmvd* is an absence status;
- $Rmvd:$ *bool* is an absence status;
- *Active* : *bool* is an activity status;
- $WasActive:$ *bool* is a previous activity status.

In synchronous composition of base processes with different names the corresponding channels of the same name must connect. In merging of processes with the same *Name* sets of channels and sets of *Channel* join. Processes with different names cannot be merged and processes with the same *Name* cannot be composed in parallel. Values of above variables define states of a base process. Its input and output channels correspond to names, types and directions of *Channel*. Transitions are defined by sending and receiving tokens through the channels. The initial state is $(Channel, 0, 0, 0)$, where *Channel* is a nonempty set of channels with $Channel.rmvd = 0$, and a number of channels with $dir = 1$ does not exceed 1 and a number of channels with $dir = 0$ can be equal to 0.

We would like to verify the following properties expressed by $\forall CTL$. For the protocol of parallel weight computing: $\mathbf{AF}(\{wasActive\}^* \wedge \mathbf{AXAF}\{\neg Active\}^*)$ (every agent was active, and then all computation will be terminated). For the protocol of conflict weight computing: $\mathbf{AF}\{\neg Active\}^*$ (all computation will be terminated); $\mathbf{AG}\{Not2Rmvd\}^*$ (Channels and agents cannot be removed twice). For every atomic formula we construct a finite deterministic automaton. They are a base for abstract functions for states of our systems. Then we should construct a set of consistent representatives for symbols of our grammar. This technique is not

present here.

5. Conclusion

In the paper we present the verification method for families of distributed systems specified by a context-sensitive grammar with quasi-terminals. This method can be used for verification of the multi-agent system of ambiguity resolution in ontology population. Properties of the system are expressed by $\forall CTL$-formulas.

In the near future we plan to implement the suggested method using model checking tool SPIN and give formal proofs of correctness of the ambiguity resolution algorithm. But some properties concerning agent interaction cannot be expressed easily in this framework. This fact is a reason for trying other more expressive formalisms for properties. Other research direction is to extend the method for other types of context-sensitive grammars.

Список литературы

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. Clarke E.M., Grumberg O., Peled D. Model Checking. MIT Press, 1999.
3. Clarke E.M., Grumberg O., Jha S. Verifying Parameterized Networks // In: ACM Transactions on Programming Languages and Systems, Vol. 19, No. 5, September 1997. Pages 726-750.
4. Fagin R., Halpern J.Y., Moses Y., Vardi M.Y. Reasoning about Knowledge. MIT Press, 1995.
5. Garanina N., Sidorova E. An Approach to Ambiguity Resolution for Ontology Population // Proc. of the 24th International Workshop on CS&P. Rzeszow, Poland, Sep. 28-30, 2015. – University of Rzeszow, 2015, Vol. 1, pp 134-145.
6. Garanina N. O., Sidorova E. A. Ontology Population as Algebraic Information System Processing Based on Multi-agent Natural Language Text Analysis Algorithms//Programming and Computer Software, 2015, V. 41, n.3, pp. 140–148.
7. De Gennaro M.C., Jadbabaie, A. Decentralized Control of Connectivity for Multi-Agent Systems // In: Proc. of 45th IEEE Conference on Decision and Control, pp. 3628 - 3633.
8. Huhns M. N., Stephens L. M. Multiagent Systems and Societies of Agents // In: Multiagent Systems, MIT Press, 1999 pp. 79–120.
9. Wooldridge, M. An Introduction to Multiagent Systems. Willey&Sons Ltd, 2002.