Indian Journal of Science and Technology, Vol 8(16), DOI: 10.17485/ijst/2015/v8i16/55160, July 2015
ISSN (Print) : 0974-6846
ISSN (Online) : 0974-5645

Executable Semantics for the Formal Specification and Verification of E-agents

Awais Qasim*, Syed Asad Raza Kazmi and Ilyas Fakhir

Department of Computer Science, Government College University, Lahore - 5400, Pakistan; awais@gcu.edu.pk*

Abstract

Background: Software agents are expected to work autonomously and deal with unfamiliar situations astutely. Achieving cent percent test case coverage for these agents has always been a problem due to limited resources. Also a high degree of dependability is expected from autonomous software agents. Formal verification of these systems can be done by specifying the possible actions of the agents to increase the confidence in the correctness of such systems. **Methods:** We have used social approach of agent communication where actions of the agents are described as social obligation between the participants. Formal specification of e-agents was done using μ-calculus as it is more expressive than Linear Temporal Logic (LTL). An algorithm has been given for syntactic conversion of μ-calculus formulas to TAPA model checker environment. **Results:** Using converstion algorithm syntactically modified μ-calculus formulas will be executed on TAPA model checker to verify the system being modelled for the specified properties. **Application:** The study will help to provide future directions for formal modelling of e-agents for their correct functioning.

Keywords: Agent Communication, Model Checking, μ-Calculus, μ TAPA Model Checker

1. Introduction

The term 'formal methods' is utilized to allude to any exercises that depend on mathematical representations of software which includes but not limited to formal software specification, transformational development, formal verification of programs. A high degree of dependability is expected from intelligent software agents. The main purpose of these agents is to interact with the dynamic environment and its not possible to build an agent with all the world knowledge composed into it. Being intelligent agents they are expected to respond intelligently to unfamiliar situations. For such systems executing a predefined representative test cases of the system cannot provide a high degree of reliability. Even if we do achieve a high degree of test case coverage, the chances of system failure are still high due to test scenarios that were never executed. When receiving the external stimuli, these agents try to perceive it according to their knowledge and any failure in this process might cause the agent to fail. In this paper an executable μ-calculus based formal technique has been proposed to specify the action of communicating agents. The expressiveness of the μ-calculus allows formalization of the communicative actions in verifiable form. Different types of agent communication languages have been proposed for communication between autonomous agents but we will be using the agent communication language proposed by Singh[1]. In this approach instead of mental attitudes of the agent like intentions, the focus is on social commitments of agents in the system. This is justifiable because in a multi agent system the agent's actions are affected by its social commitments. When perceiving the system as a whole in these open multi agent systems, the inner states of certain agents might not be perceptible. In these social semantics the actions of the agents are modeled by its commitments to other agents. Using these semantics the system is evolved by the dynamic interaction of the agents where agent has to respect its commitments. Another advantage of the social approach is that we do not need to provide a comprehensive specification of all

*Author for correspondence

possible action sequences as in a behavioral approach. Here we have a categorical specification of basic actions to model the agent communication.

In the past developing executable semantics for e-agent's communication languages has been examined in detail. In [2] verifiable semantics based on computational models have been proposed. In [3] a framework for communication among e-agents was presented. In particular they used different languages for the specification of communication among e-agents. Giordano has used Dynamic LTL Logic for specifying and verifying agent communication properties[4]. In DLTL we express the programs as regular expressions and specify the communicative actions as precondition laws, causal laws and constraints. The use of model checking for verification of e-agents has been discussed in [5] using Belief-Desire-Intention (BDI) attitudes. They used a new logic which was the composition of formal temporal evolution and formal BDI attitudes. A process algebra based approach for e-agent's communication was discussed in [6]. They proposed new semantics called ACPL which models the basics of agent communication. They used the information processing aspects of Concurrent Constraint Programming (CCP) with a generalisation of the synchronous handshaking communication mechanism of Communicating Sequential Processes (CSP). However the use of μ-calculus for formal specification of e-agents is a novel approach.The μ-calculus being more expressive than LTL[7] is computationally more suitable for modeling and verification of the proposed system. The μ-calculus is an augmentation of Computation Tree Logic (CTL) with least fixpoint (μ) and greatest fixpoint (ν) operators[8]. The addition of minimum and maximum fixed points incorporating recursion to modal logic provides a great increase in expressive power.

In this paper communication between agents is exhibited in the form of interaction protocols. These interaction protocols can be viewed as commitments between agents in a communication system. The overall state of communicating system comprising of these agents is evolved by the dynamic execution of these commitments[1]. Preconditions for the execution of actions are specified in the form of precondition laws. Execution of certain actions may result in the establishment of new commitments between participating agents. We have specified the action of agents involved in interaction protocol in the form of μ-calculus formulas. For verification that every agent in the system satisfies the preconditions and is able to execute its actions can be formally represented as a validity problem in μ-calculus logic. The validity problem can then be solved by the help of μ-calculus based model checker.

The rest of this paper is divided as follows. In section 2 some preliminaries for notations and semantics of μ-calculus are given. Section 3 describes the formal specification of the actions of e-agents along with the preconditions and commitments. In section 4 verification of the e-agents is done through a TAPA model checker alongwith an algorithm for syntactic conversion of μ-calculus based properties to TAPA environment. Section 5 concludes the paper.

2. Preliminaries

A few terminologies, mathematical and computational models have been described in this section that will be used in the rest of the discussion for the specification of the problem under analysis.

2.1 Modal μ-Calculus

The μ-calculus is a type of temporal logic used in diverse areas of computer science for the specification and formal verification of systems. The requirements of hardware and software systems can be specified in it and then it can be verified whether these systems satisfies those requirements or not. The process of verification can be done by using different model checkers available for μ-calculus[9]. The μ-calculus is an extension of CTL with a least fixpoint operator (μ) and greatest fixpoint operator (ν) which makes it possible to give characterizations of correctness properties. Most modal and temporal logics of interest can be defined in terms of μ-calculus. We will use the Hennessy Milner Logic syntax to specify the formulas of μ-calculus. HML is an archaic modal logic of action[10]. Formulas of the μ-calculus F_μ can be be generated by following rules given below. Let Act denote the set of all possible actions and $a \in Act$

$$F_\mu ::= false \mid true \mid F_\mu \wedge F_\mu \mid F_\mu \wedge F_\mu \mid [a]F_\mu \langle a \rangle F_\mu$$

Interpretation of $\langle a \rangle F_\mu$ is that there is a choice to take $'a'$ action and reach a state in which F_μ is satisfied. $[a]F_\mu$ denotes that no matter how we take $'a'$ action we will end up in a state where F_μ holds. To formally specify that a state s of a transition system satisfy a required property of interest we can express it as $s \vDash \langle a \rangle F_\mu$ iff $\{\exists t.s \rightarrow t \wedge t = F_\mu\}$. Atomic propositions can be used in the

logic to find out the set of states that satisfies those atomic propositions. The operators ⟨ ⟩ and [ ] can be used with a set of actions.

Let A = a_a,...$a_n \subseteq$ Act with n ≥ 1 and ϕ denotes no action then $\langle A \rangle F_\mu$ represents $\langle a_1 \rangle F_\mu \vee \ldots \vee \langle a_n \rangle F_\mu$ and $\langle \phi \rangle F_\mu$ = false and $[A]F_\mu$ represents $[a_1]F_\mu \wedge \ldots \wedge [a_n]F_\mu$ and $[\phi]F_\mu$ = true.

2.2 Semantics of μ-Calculus

Hennessy Milner Logic does not allow the specification of properties of infinite depth. This limits the expressive power of μ-calculus and even the invariant and reachability properties cannot be verified. For example the properties

Invariant $(F_\mu) = F_\mu \wedge [Act]\ F_\mu \wedge [Act][Act]\ F_\mu \wedge [Act][Act][Act]\ F_\mu \wedge \ldots$

Reachability $(F_\mu) = F_\mu \vee \langle Act \rangle\ F_\mu \vee \langle Act \rangle \langle Act \rangle\ F_\mu \vee \langle Act \rangle \langle Act \rangle \langle Act \rangle\ F_\mu \vee \ldots$

are not expressible in HML. The use of least and greatest fixpoint operators is the basis of μ-calculus. If S is the set of states representing the complete state space of the system then we can provide semantics of a state-based modal logic. Using these semantics we can map a μ-calculus formulae F_μ to the set of states where that formula is satisfied. This mapping of formula F_μ to the set of states is given by $[\![F_\mu]\!]$. The mapping provides us the states that satisfy the *formula.* $[\![F_\mu]\!] \subseteq S$ *is defined inductively as*

$$[\![true]\!] = S$$
$$[\![false]\!] = \phi$$
$$[\![F_\mu \wedge G_\mu]\!] = [\![F_\mu]\!] \cap [\![G_\mu]\!]$$
$$[\![F_\mu \vee G_\mu]\!] = [\![F_\mu]\!] \cup [\![G_\mu]\!]$$
$$[\![\langle a \rangle F_\mu]\!] = \langle .a. \rangle [\![F_\mu]\!]$$

$$where \langle .a. \rangle T = \{p \in S \mid \exists p'.p \xrightarrow{a} p' \ and \ p' \in T\}$$
$$[\![[a]F]\!] = [.a.][\![F]\!]$$
$$where\ [.a.]T = \{p \in S \mid \forall p'.p \xrightarrow{a} p' \Rightarrow p' \in T\}$$

To specify formulas in a recursive mode we can add a free variable X in our logic having interpretations over F_μ S like

$$X \overset{def}{=} F_\mu \wedge [Act]X(Invariant)$$
$$X \overset{def}{=} F_\mu \vee [Act]X(Reachability)$$

We can now write HML with one recursively defined variable as

$$F_\mu ::= X \mid false \mid true \mid F_\mu \wedge F_\mu \mid F_\mu \vee F_\mu \mid [a]F_\mu \mid \langle a \rangle F_\mu$$

where X is a distinguished variable with a definition $X \overset{\min}{=} F_X$ or $X \overset{\max}{=} F_X$ and a ∈ Act such that F_X is a formula of the logic that can contain X. Alternatively

$$F_\mu ::= X \mid false \mid true \mid F_\mu \wedge F_\mu \mid F_\mu \vee F_\mu \mid [a]F_\mu \mid \langle a \rangle F_\mu \mid \mu X.F_\mu \mid \nu X.F_\mu$$

These formulation will be used in order to specify the actions of the communicating agents.

3. Action Specifications of Communicating Agents

Every agent of the system under consideration has its own set of actions. Every action has a corresponding local affect on the agent and global affect on the system when that action is executed. Global state represents complete state of the system, which is composed of individual states of all the agents in the system. Let $Agents_i = Agent_1$, $Agent_2$...$Agent_n$ be the set of agents representing agent space for the system under consideration. For action specification of the agents we will adopt the synchronous model of communication. This implies that every communication action will generically be of the form *action (*$agent_1$*,* $agent_2$*).* It means that $agent_1$ will intentionally execute an action with $agent_2$ e.g. *send (sender, receiver, message)* means that sender will perform an action send on the receiver with message as argument. If the receiver needs to reply back to the sender then he can use the same pattern. However the execution of actions will be done synchronously in a deterministic order. Depending on the nature of the action an argument may or may not be required. Local states of both the sender and the receiver are updated sequentially based on their action specification.

The semantics of communicative actions will be specified by a protocol that describes the effects of each action on the social state of the system. Execution of certain actions may give rise to new obligations between the interacting agents. Preconditions laws will be used to specify conditions that need to be true before some actions can be executed. For simplicity we will only use preconditions for the sender and assume that actions are always executable for the receiver. Normally execution of an action will affect both the state of the sender and the

state of the receiver but for action like acknowledgement this will not be true. All agents can only see the effects of actions in which they participate. This means that for two agents the history of all communications is known to both of them and so they have the same local view of the system state. For more than two interacting agents this will not be true.

We will use the example of NetBill protocol for specifying the properties of two interacting agents. NetBill protocol was proposed for buying and selling goods on the internet[11]. In the protocol initially a customer requests a quote for some desired goods and subsequently the merchant replies back with the quote. On acceptance of the quote by the customer the merchant delivers the goods in an encrypted form. The customer cannot use the goods until he has made the payment. After the customer pays the agreed amount, the merchant sends the receipt to the customer with which he can successfully decrypt and use the goods. In this protocol the communication can be initialized in any sequence like the merchant may send the quote without customer request and the customer may send confirmation without prior conversation on the price of the goods. For some actions we will need to impose restriction like merchant will never send the receipt until the customer has made the payment. For such actions we will used the precondition laws. We will use two agents for our example *mr* representing the merchant and *ct* representing the customer.

Atomic propositions for our example will be {*Send Quote, Send Goods, Send Request, Send Receipt, Send Payment, Send Accept*}. In our example for communicative actions {*Send Quote, Send Goods, Send Receipt*}, merchant will be the sender and customer will be the receiver. For communicative actions {*Send Request, Send Accept, Send Payment*}, customer will be the sender and merchant will be the receiver. The customer and merchant agents will execute actions synchronously. For actions that will create new obligations between agents we will represent them as commitments. Commitments can be basic of the form $C(ag1, ag2, action)$, which means that agent *ag1* is obligated to agent *ag2* to execute the action. Additionally there can be conditional commitments of the form $CC(ag1, ag2, cond, action)$, which means that agent *ag1* is obligated to agent *ag2* to execute action only if condition *cond* is satisfied. There is a vast majority of properties that can be verified for customer and merchant according to the protocol but we will restrict ourselves to the basic properties for simplicity. Action laws for the merchant and customer are specified as follow.

3.1 Semantics of Commitments

As discussed above that execution of certain actions may create new obligations between the interacting agents which we will express as commitments. They represent static properties of the agents and cannot be verified on a labeled transition system until the dynamic behavior of the agents is observed. Some of the commitments are specified below.

- When the action *Send Quote* will be executed by the merchant then he is committed to execute the action *Send Goods* if customer has executed the action *Send Accept* showing his confirmation. Also merchant is committed to execute the action *Send Receipt* if the customer has executed the action *Send Payment*

$$[Send\ Quote]_{mr}\ true \rightarrow CC(mr, ct, Send\ Accept, Deliver\ Goods) \wedge CC(mr, ct, Send\ Payment, Send\ Receipt)$$

- On execution of action *Send Goods* by the merchant, the merchant is committed to send receipt to the customer after he has made payment.

$$[Send\ Goods]_{mr}\ true \rightarrow \wedge\ CC(mr, ct, Send\ Payment, Send\ Receipt)$$

3.2 Semantics of Preconditions

As discussed above we will only use preconditions for the sender and for receiver we take the assumption that all actions are always executable. For merchant there is only one precondition law

- The action *Send Receipt* cannot be executed by the merchant until the customer has paid. All other actions are always executable by the merchant.

$$[Send\ Payment]\ false \rightarrow [Send\ Receipt]_{mr}\ false$$

The only condition on the execution of actions by the customer is the following precondition law:

- It means that the customer may send a payment for the goods only if he has received the goods (all other actions are always executable for the customer).

$$[Send\ Goods]\ false \rightarrow [Send\ Receipt]_{mr}\ false$$

4. Verification of Communicating Agents

We have used the TAPA model checker[12] to verify the properties of *merchant* and *customer* agents. The required properties had to be modified syntactically for their execution on the model checker. The syntactic changes did not affected the semantics of the actions of agent. To automate the process of this conversion we have devised an algorithm. All the μ-calculus formulas to be converted are input to the algorithm in a single text file. The input file should contain a single formula per line. In Data section we declare variable to be used in the algorithm. The variable *fixpoint* will be used to keep track of whether the formula we are going to convert is least/greatest fix point or not. *formula* will contain the single μ-calculus formula which is going to be converted. *conformula* represents converted forumla which can be input to the TAPA model checker. *Ptr* will serve as the file pointer for the input file. *EOF* represents the end of file for the input file. The function *readLine* will read a single μ-calculus formula at a time. The function *append* will concatenate the second argument to the first argument. The function *readToken* reads single token at a time separated by space. The function *lastToken* will return the last token of any text input to it as argument. The function *initiate* creates a new string with text input to it as argument. the function *movePointer* will move the file pointer to the next formula. The function *writeToFile* will write the converted formula to the output file one at a time.

Table 1 and Table 2 shows the actual μ-calculus formulas for the *merchant* and *customer* agents respectively along with their corresponding modified versions used to run on the model checker. For each property the model checker verifies whether the agent satisfies it or not by outputting a boolean result. The technique can be extended to verify the properties of interest for any number of agents in a system. After execution of *Algorithm 1* on the specified properties of merchant and customer we get syntactically modified executable semantics. Table 3 and Table 4 represents these executable semantics.

Table 1. Actions of the merchant

Property Semantics	Description
$\neg\langle Send\ Quote\rangle_{mr}$ true or $\langle Send\ Quote\rangle_{mr}$ false	Merchant cannot send quote at the moment to the customer
$\langle Send\ Quote\rangle_{mr}$ true $\wedge\neg\langle Send\ Goods\rangle_{mr}$ false	Merchant can send quote to customer but cannot deliver the goods.
$\langle Send\ Quote\rangle_{mr}$ true	When there is a request for a quote the merchant sends the quote and the request is finished.
$\langle Send\ Receipt\rangle_{mr}$ true	When the merchant sends the receipt then the receipt is delivered to the customer.
$X \overset{\min}{=} [SendQuote]false \vee \langle Act\rangle X$	It is possible to reach a state where merchant cannot send the quote.
$X \overset{\min}{=} [SendReceipt]false \vee \langle Act\rangle X$	It is possible to reach a state where merchant cannot send the receipt.
$X \overset{\min}{=} [SendGoods]false \vee \langle Act\rangle X$	It is possible to reach a state where merchant cannot send the goods.
$X \overset{\max}{=} \langle PA_{mr}\rangle true \wedge [Act]X$	Let PA_{mr}={*send Quote, send Receipt, send Goods*} then in every reachable state the merchant can send quote or send receipt or send goods.
$X \overset{\max}{=} [Act]false \wedge X$	The merchant is deadlocked.

Table 2. Possible actions of the customer

Property Semantics	Description
$\langle Send\ Request \rangle_{ct}$ true	The customer can send the request.
$\langle Send\ Accept \rangle_{ct}$ true	The customer can send the confirmation.
$\langle Send\ Payment \rangle_{ct}$ true	The customer can send the payment.
$[Send\ Accept]_{ct}$ ($\langle Send\ Goods \rangle$ *true* $\wedge$ $[Act \backslash \{Deliver\ Goods\}]$ *false*)	On acceptance of quote by the customer the merchant always delivers the goods.
$[Send\ Payment]_{ct}$ ($\langle Send\ Receipt \rangle$ *true* $\wedge$ $[Act \backslash \{Send\ Receipt\}]$ *false*)	If the customer has sent the payment then the merchant always send the receipt.
$\langle Request\ Quote \rangle_{ct}$ *true* $\wedge$ $\langle Send\ Accept \rangle_{ct}$ *false*	The customer can request quote for some desired goods but cannot send the accept message.
$X \overset{\min}{=} [RequestQuote]\,false \vee \langle Act \rangle X$	It is possible to reach a state where customer cannot request the quote.
$X \overset{\min}{=} [SendAccept]\,false \vee \langle Act \rangle X$	It is possible to reach a state where customer cannot send the confirmation.
$X \overset{\min}{=} [SendPayment]\,false \vee \langle Act \rangle X$	It is possible to reach a state where customer cannot send the payment.
$X \overset{\max}{=} \langle PA_{ct} \rangle true \wedge [Act]X$	In every reachable state the customer can send request or send confirmation or send payment. Let PA_{ct} = {*send Request, send Accept, send Payment*} then
$X \overset{\max}{=} [Act]\,false \wedge X$	The customer is deadlocked.

Table 3. μ-calculus properties for the customer and their corresponding modified formulas for TAPA model checker

Property Name	Modified Formulas for Model Checker
C1	property C1 : ⟨*Send Request?*⟩ true end
C2	property C2 : ⟨*Send Accept?*⟩ true end
C3	property C3 : ⟨*Send Payment?*⟩ true end
C4	property C4 : [*Send Accept?*] [*Deliver Goods?*] true end
C5	property C5 : [*Send Payment?*] [*Send Receipt?*] true end
C6	property C6 : ⟨*Request Quote?*⟩ true & ![*Send Accept?*] false end
C7	property C7 : minZ: ([*Request Quote?*] false | <*> Z) end
C8	property C8 : minZ: ([*Send Accept?*] false | <*> Z) end
C9	property C9 : minZ: ([*Send Payment?*] false | <*> Z) end
C10	property C10 : maxZ: (⟨*Send Request?*⟩ true | ⟨*Send Accept?*⟩ true | ⟨*Send Payment?*⟩true & [*]Z) end
C11	property C11 : maxZ: ([*] false & Z) end

Table 4. μ-calculus properties for the merchant and their corresponding modified formulas for TAPA model checker

Property Name	Modified Formulas for Model Checker
M1	*property M1:* !⟨*Send Quote?*⟩ *trueend*
M2	*property M2:* ⟨*Send Quote?*⟩ *true* &! [*Send Goods?*] *falseend*
M3	*property M3:* ⟨*Send Quote?*⟩ *trueend*
M4	*property M4:* ⟨*Send Receipt?*⟩ *trueend*
M5	*property M5: minZ.* ([*Send Quote?*] *false* | ⟨∗⟩ *Z*) *end*
M6	*property M6: minZ.* ([*Send Receipt?*] *false* | ⟨∗⟩ *Z*) *end*
M7	*property M7: minZ.* ([*Send Goods?*] *false* | ⟨∗⟩ *Z*) *end*
M8	*property M8: maxZ.* (⟨*Send Quote?*⟩ *true* | ⟨*Send Receipt?*⟩ true |⟨*Send Goods?*⟩ true & [∗] *Z*) *end*
M9	*property M9: maxZ.* ([∗] *false* & *Z*) *end*

5. Conclusion

In this paper we have shown how actions of communicating agents can be specified in modal μ-calculus. The problem of developing a verifiable semantics for agent communication has been discussed by many in the past[2]. Multi agent systems are seen as a key enabling technology for the future e-commerce products. To ensure a high degree of reliability of these systems we need an executable semantics for them. This is evident by the difficulty to achieve good test case coverage for a system of reasonable size. We have used a mentalistic approach for the specification and verification of communicating agents. Netbill protocol was used to specify the properties of two interacting agents for buying and selling of goods on the internet. The major advantage of our approach is that the semantics are in executable form and the process of verification of properties can be done on any model checker supporting the modal μ-calculus logic. We used TAPA model checker to verify the properties of the customer and merchant agents. The technique can be used to specify and verify the properties of any number of agents. The current work limits the commitment to two participating agents. In a multi agent system we may be need to specify interaction between more than two agents simultaneously e.g a merchant might be communicating with two or more customers and if the two customers already have some commitments between them than the interaction protocol needs to be modified to accommodate the interaction between three or more agents con currently.

```
Data: fixpoint = no, formula, conformula, ptr
ptr = openFile(inputFile)
while ptr != EOF do
   formula = readLine(ptr)
   conformula = append (conformula, property propertyname)
   while readToken(formula) != \n do
      if readToken(formula)= ¬ then
         conformula = append(conformula,!)
      else if readToken(formula)= ∧ then
         conformula = append(con formula,&)
      else if read Token(formula)= ∨ then
         conformula = append(con formula,|)
      else if readToken(formula)= ⟩ then
         if lasttoken(conformula)= ∗ then
            conformula = append(conformula, ⟩)
         else
            conformula = append(conformula,? ⟩)
         end if
      else if readToken(formula)= ] then
      if lastToken(conformula)= ∗ then
         conformula = append(conformula,])
      else
         conformula = append(conformula,?])
      end if
      else if readToken(formula)= =min then
         conformula = initiate(minZ:()
         fixpoint=yes
```

```
    else if readToken(formula)= max= then
        conformula = initiate(maxZ:()
        fixpoint=yes
    else if readToken(formula)= Act then
        conformula = append(conformula,*)
    else if readToken(formula)= X then
        conformula = append(conformula,Z)
    else
        conformula = append(conformula,readToken
        (formula))
    end if
    movePointer(ptr)
  end
    if fixpoint=yes then
        conformula = append(conformula,))
    else
        conformula = append(conformula,end)
    end if
        writeToFile(outputFile,conformula)
        fixpoint = no
end
```

Algorithm 1: Syntactic conversion of μ-calculus formulas to TAPA

6. References

1. Singh MP. A social semantics for agent communication languages. Issues in agent communication. Springer Berlin Heidelberg; 2000. p. 31–45.
2. Wooldridge M. Semantic issues in the verification of agent communication languages. Autonomous agents and multi-agent systems. 2000; 3(1):9–31.
3. Guerin F. Specifying agent communication languages [Doctoral Thesis]. University of London; 2002.
4. Giordano L, Martelli A, Schwind C. Specifying and verifying systems of communicatingagents in a temporal action logic. Advances in Artificial Intelligence. Springer Berlin Heidelberg; 2003. p. 262–74.
5. Benerecetti M, Giunchiglia F, Serafini L. Model checking multiagent systems. Journal of Logic and Computation. 1998; 8(3):401–23.
6. van Eijk RM, de Boer FS. van der HW, Meyer J-JC. Process algebra for agent communication: A general semantic approach. Communication in Multiagent Systems. Springer; 2003. p. 113–28.
7. Wang BY. Mu-calculus model checking in maude. Electronic Notes in Theoretical Computer Science. 2005; 117:135–52.
8. Emerson EA. Model checking and the mu-calculus. DIMACS series in discrete mathematics. 1997; 31:185–214.
9. Bradfield J, Stirling C. Modal mu-calculi. Handbook of modal logic. 2007; 3:721–56.
10. Hennessy M, Milner R. On observing nondeterminism and concurrency. ICALP, LNCS. 1980; 85:299–309.
11. Yolum P, Singh M. Flexible protocol specification and execution: Applying event calculusplanning using commitments. AAMAS'02 Bologna, Italy. 2002. p. 527–34.
12. Calzolai F, De NR, Loreti M, Tiezzi F. Tapas: A tool for the analysis of process algebras. Transactions on Petri Nets and Other Models of Concurrency I. Springer Berlin Heidelberg; 2008. p. 54–70.