Verification and Behavioral Synthesis of Agent-Based Systems.

Fernando Asteasuain[1,2][0000−0002−5498−6878], Federico D'Angiolo[1][0000−0002−5500−6587], Manuel Dubinsky[1], and Pablo Daniel Gamboa[2]

[1] Universidad Nacional de Avellaneda
[2] Universidad Abierta Interamericana - Centro de Altos Estudios CAETI
{fasteasuain}@undav.edu.ar

Abstract. In this work we explore the FVS language as a formalism to express, validate and synthesize behavior in the agent-based systems' world. Recent work relates Behavioral Synthesis with agent-based systems, opening the possibility for formalisms in the formal verification area to make an impact in the artificial intelligence domain. In this work we analyze FVS as a potential candidate to make a contribution given its desirable characteristics such as flexibility, great expressive power and its ability to perform behavioral synthesis in Open Systems. A very well known case of study is analyzed: the Dining Cryptographers protocol, including one variation of the protocol. FVS was able to fully specify, validate and synthesize the behavior of the protocol.

Keywords: Agent-Based Systems · Behavioral Synthesis · Formal Verification

1 Introduction

The formal verification of agent-based systems [7, 11, 30, 31, 43, 9] is becoming a crucial activity since the application and use of these kind of appealing systems in several and important domains such as e-commerce, simulation or distributed collaborative systems [7, 18, 24, 38] among others is on rise in the last few years [19, 31, 26, 39].

Several formalisms have been applied to model how agents communicate and reason about the world and the environment in order to achieve their goals. Among them, we can name epistemic logics [40, 25, 1] , logic-based protocols [5, 7] and communicative social commitments [43, 6, 17]. In the same line, different model checkers like [13] and other similar techniques and tools have been applied to formally verify agent-based systems [7, 11, 30, 31, 43, 9].

Although the meaningful advances introduced by the mentioned techniques there are still issues to be addressed. One of them involves combining Open Systems [3, 28, 14, 33] and agent-based systems [43]. There is an natural challenge involved when dealing with Open Systems since actions beyond the control of the system must be considered, in contrast to systems known as closed where all the

events to occur are handled entirely by the system. Open Systems interact with an environment which generates events (non controllable by the system) which may impact in its behavior, which constitutes an affect known as Controllability. This is also referred as *uncertainty* in works like [43, 29, 22, 44].

Another relevant aspect is the expressiveness of the specification language used to denote the expected behavior of the system. In this sense, several authors claim the need for a more expressive specification language [21, 43, 2, 42, 31, 23, 20].

An interesting line of research has been pinpointed by [15, 41]. These works relate Behavioral Synthesis with *Artificial Intelligence* concepts as agent-based systems by treating behavioral synthesis as a planning problem. This relationship allows the possibility that improvements in one area may impact in the other one and vice versa. Behavioral Synthesis can be seen as an automated procedure to obtain a correct-by-construction reactive system from its temporal logic specification [28, 33]. In the case of reactive synthesis, an implementation is typically given as an automaton that accepts input from the environment (e.g., from sensors) and produces the system's output (e.g., on actuators).

Given this context in this work we explore FVS (Feather Weight Visual Scenarios) [2–4] as a formal specification language to model, specify and synthesize behavior of agent-based systems. FVS is a declarative language based on graphical scenarios and features a flexible and expressive notation with clear and solid language semantics. FVS can be used to denote, compose and synthesize behavior considering linear and branching type logics. That is, both LTL-like and CTL-like properties can be stated. In addition, FVS can be employed in the context of Open Systems taking into account non controllable actions. FVS expressive power is a distinguishable feature since it is more expressive than LTL [2] and also can synthesize properties that can not be expressed with Deterministic Büchi automata [3]. Based on the relationship between behavioral synthesis and agent-based systems [15, 41] and taking into consideration FVS's expressive power, flexibility and behavioral synthesis power in Open Systems FVS was explored in the agent-based systems' world. To validate our proposal we analyze a known case of study: The Dining Cryptographers protocol [12], which has been widely used in the literature for verifying agent-based systems [31, 23, 30, 34]. Besides fully modeling and verifying the protocol a controller for the system was also found. Also a variation of the protocol is analyzed. Given the obtained results we believe this work constitute an exploratory but solid first step for FVS in the Artificial Intelligence domain.

The rest of this paper is structured as follows. Section 2 presents the main features of the FVS language and explains how FVS specifications can be synthesized. Section 3 shows our approach in action by modeling and verifying the Dining Cryptographer protocol [12] plus obtaining a controller for the system. Sections 4 and 5 discuss some related and future work respectively while Section 6 exposes the conclusions of this paper.

2 Background

FVS is informally introduced in section 2.1 while FVS behavioral synthesis procedure is addressed in Section 2.2.

2.1 FVS: Feather Weight Visual Scenarios

In this section we will informally describe the standing features of FVS [2]. The reader is referred to [2] for a formal characterization of the language. FVS is a graphical language based on scenarios. Scenarios are partial order of events, consisting of points, which are labeled with a logic formula expressing the possible events occurring at that point, and arrows connecting them. An arrow between two points indicates precedence of the source with respect to the destination: for instance, in Fig. 1-a A-event precedes B-event. We use an abbreviation for a frequent sub-pattern: a certain point represents the next occurrence of an event after another. The abbreviation is a second (open) arrow near the destination point. For example, in Fig. 1-b the scenario captures the very next B-event following an A-event, and not any other B-event. Conversely, to represent the previous occurrence of a (source) event, there is a symmetrical notation: an open arrow near the source extreme. For example, in Fig. 1-c the scenario captures the immediate previous occurrence of a B-event from the occurrence of the A-event, and not any other B-event. Events labeling an arrow are interpreted as forbidden events between both points. In Fig. 1-d A-event precedes B-event such that C-event does not occur between them. FVS features aliasing between points. Scenario in Fig. 1-e indicates that a point labeled with A is also labeled with *A and B*. It is worth noticing that A-event is repeated on the labeling of the second point just because of FVS formal syntaxes [2] . Finally, two special points are introduced as delimiters to denote the beginning and the end of an execution. These are shown in Fig. 1-f.

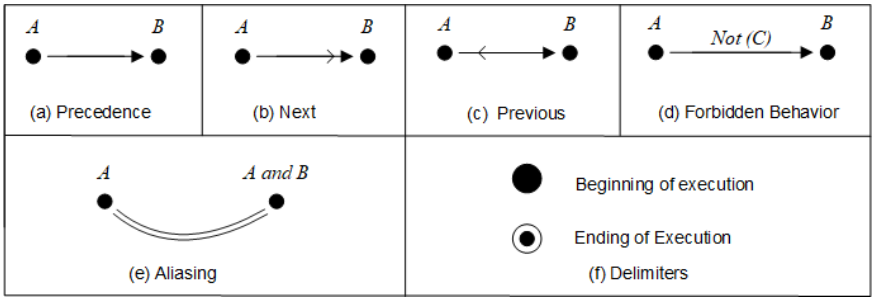

Fig. 1. FVS Basic Features

We now introduce the concept of **FVS rules**, a core concept in the language. Roughly speaking, a rule is divided into two parts: a scenario playing the role

of an antecedent and at least one scenario playing the role of a consequent. The intuition is that whenever a trace "matches" a given antecedent scenario, then it must also match at least one of the consequents. In other words, rules take the form of an implication: an antecedent scenario and one or more consequent scenarios. Graphically, the antecedent is shown in black, and consequents in grey. Since a rule can feature more than one consequent, elements which do not belong to the antecedent scenario are numbered to identify the consequent they belong to. Two examples are shown in Fig. 2 modeling the behavior of a client-server system. The rule in left margin of Fig. 2 establishes that every request received by a server must be answered, either accepting the request (consequent 1) or denying it (consequent 2). The rule at the right margin of Fig. 2 dictates that every granted request must be logged due to auditing requirements.

Fig. 2. FVS Rules examples

2.2 FVS Behavioral Synthesis

We now describe the basic functioning of the behavioral synthesis scheme in FVS. The reader is referred to [3, 4] for a complete characterization of the procedure. FVS specifications are used to obtain a controller using different tools depending on the type of the property. Using the tableau algorithm detailed in [2] FVS scenarios are translated into Büchi automata. If the automata represent one of the specification patterns (excludings those that can not be represented by Deterministic Büchi automata) then we obtain a controller using a technique [33] based on the specification patterns [16] and the GR(1) subset of LTL. If that is not the case, but the automata is a Deterministic Büchi automata we either employ the GOAL [45] or the Acacia+ tool [10]. If the automaton is non deterministic we obtain a controller using only the GOAL tool using an intermediate translation from Büchi automata to *Rabin* automata. For example, a controller for the *DDR2 Memory Interface* case of study explained in [4] is shown in Fig. 3. The controller monitors, amongst others, a number of properties that involve timing relationship between events that happen in data and data strobe signals (represented by the evensDQ and DQS respectively) and some specific thresholds for the memory interface [32, 4].

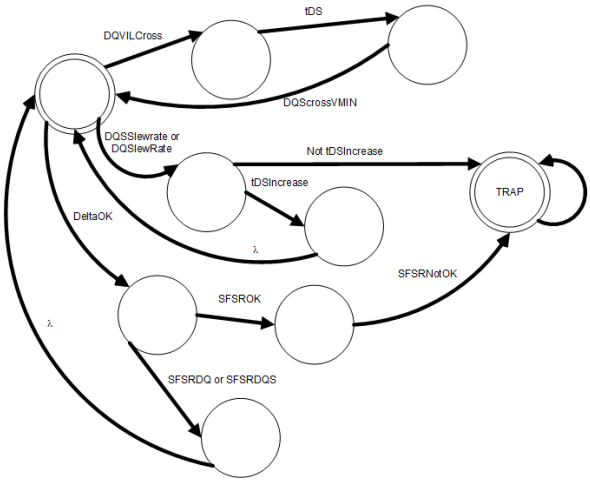

Fig. 3. A controller obtained upon FVS specifications

3 Case Study

In this section we show the FVS language in action modeling an appealing protocol (including also one variation) in the agent-based systems' domain. Furthermore, upon the FVS specification a controller for the system is obtained. In particular, we model, specify, verify and obtain a controller for the protocol of the dining cryptographers introduced by [12]. We followed some modeling conditions, restrictions and extensions based on the protocol specification given by [30, 23]. In few words, the main objective of this protocol is to allow the anonymous broadcasting of messages. The declarative specification for the protocol can be extracted from the text [12]: *Three cryptographers are sitting down to dinner at their favorite three-star restaurant. Their waiter informs them that arrangements have been made with the maitre dhotel for the bill to be paid anonymously. One of the cryptographers might be paying for dinner, or it might have been NSA (U.S. National Security Agency). The three cryptographers respect each others right to make an anonymous payment, but they wonder if NSA is paying. They resolve their uncertainty fairly by carrying out the following protocol: Each cryptographer flips an unbiased coin behind his menu, between him and the cryptographer on his right, so that only the two of them can see the outcome. Each cryptographer then states aloud whether the two coins he can seethe one he flipped and the one his left-hand neighbour flippedfell on the same side or on different sides. If one of the cryptographers is the payer, he states the opposite of what he sees. An odd number of differences uttered at the table indicates that a cryptographer is paying; an even number indicates that NSA is paying (assuming that dinner was paid for only once). Yet if a cryptographer is paying, neither of the other two learns anything from the utterances about which cryptographer it is.*

The following Section (Section 3.1) details how the protocol is modeled and verified in FVS. Section 3.2 shows how a controller for the system is obtained. Section 3.3 introduces a "cheating" version of the protocol. Finally, Section 3.4 presents some final observations for the case of study.

3.1 Dining Cryptographers' Specification in FVS

Following the strategy introduced in [30, 23] we introduce events representing cryptographers, what they know in each moment, what they express in each utterance, the flip of the coin, to know whether the number of utterances is odd or even and actions from the environment (selection of the payer and the result of coin tosses). We set a model with three cryptographers.

We first model some basic rules guiding the protocol behavior. For example, after the coin is flipped each cryptographer must declare what she see from its neighbours(*SayEqual* or *SayDifferent* events) before the coin is flipped again. This is addressed in the FVS rule in Fig. 4: between two consecutive events, cryptographers must announce what they see (either *SayEqual*, consequent #1 or *SayDifferent*, consequent #2).

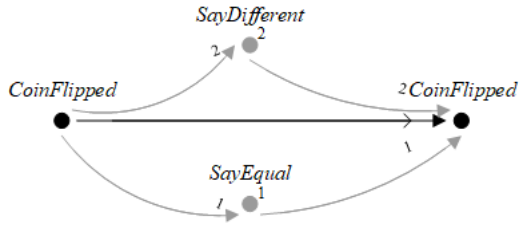

Fig. 4. Cryptographers must announce the flip of the coin result

The following rule in Fig. 5 states that if a cryptographer announces *SayEqual* or *SayDifferent* then the coin must had been flipped in the past. This rule emphasizes that no cryptographer speaks unless it is required.

Fig. 5. If a cryptographer speaks then a coin should have been flipped in the past

The following rules in Fig. 6 model the behavior of the artifact in charge of counting the number of utterances and establishing whether it is even or odd. The rule in the top says that the first occurrence of the *CoinFlipped* event indicates an odd number of occurrences. The following two rules shape the change of

state from even to odd and from odd to even given each next occurrence of the *CoinFlipped* event.

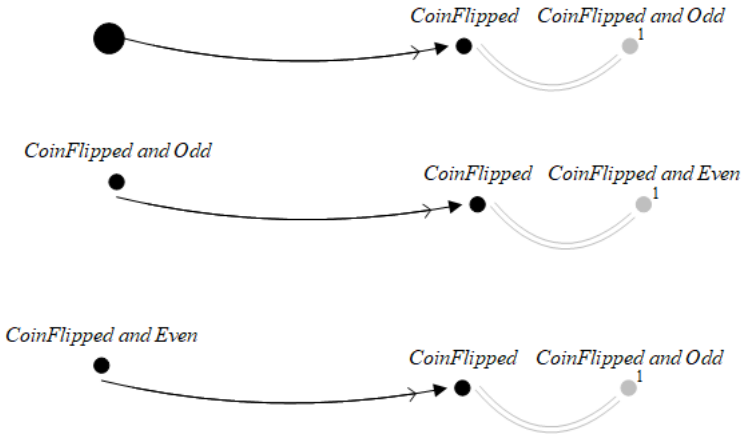

Fig. 6. Odd and even utterances' determination

We now model two properties that must be verified in the system considering a system with three cryptographers. The first one (*Property 1*) establishes that if the first cryptographer did not pay for the dinner and there is an odd number of utterances then she knows that someone of the remaining cryptographers paid for the dinner, but she does not know who the payer is. Finally, the second property (*Property 2*) says that if the number of utterances is even, the first cryptographers knows that nobody paid for the dinner. As it was previously mentioned we introduce events to represent whether a cryptographer paid or not, and also what does she knows at each moment. For example, the event *C2NotPaid* indicates that cryptographer number two did not pay the dinner. Similarly, an event like *C1KC2NotPaid* represents the fact that cryptographer number *1* knows that cryptographer number *2* did not pay. These two properties are addressed in Fig. 7. The first two rules of figure Fig. 7 capture the requirements denoted by *Property 1*. The one in the top says that given the required conditions the first cryptographer knows that one of the others cryptographers paid for the dinner. The rule beneath it adds that for the same triggering conditions the first cryptographer does not know *which* of the other two cryptographer actually paid. Finally, rule in the bottom of Fig. 7 shapes the behavior introduced by *Property 2*.

3.2 Dining Cryptographer's Validation and Controller Synthesis

Using the FVS specification for the system under consideration we were able to verify the properties shaping the behavior of the Dining Cryptographers protocol. In addition, FVS specifications can be translated into Büchi automata and

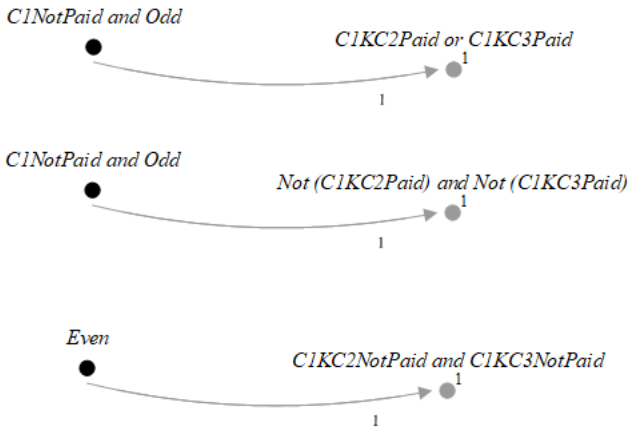

Fig. 7. Properties of the protocols that must be satisfied

then can be used as input to synthesis tools to obtain a controller for the system. Given this, we were able to obtain a controller for the protocol which monitors its behavior and responds as expected with the environment. A simplification of the controller is shown in Fig. 8, where two simple paths are shown: one where the *NSA* paid the dinner and other one where the payer is cryptographer number two. The complete controller for the protocol contains 103 states and 423 transitions.

3.3 Dining Cryptographer Protocol: a Variation

We now consider a different version of the protocol introduced in [23] named Cheating Dining Cryptographer. In this version a cryptographer is allowed to "cheat". That is, a cryptographer can say the opposite she is supposed to say. This version can be seen as modeling a faulty transmission. New events are introduced to reflect this new variation of the protocol. In particular, now a cryptographer can behave correctly or in a cheating mode. This is shown in Fig. 9.

We first introduce three rules to check if they are satisfied by the cheating version as denoted in [23]. The first one (*Property Cheating 1*) states that always when the number of differences is odd and the first cryptographer has not paid for dinner, then she knows the cryptographer who paid for dinner. As expected, this rule is not satisfied in the model since no of the cryptographers should have this information. The second one (*Property Cheating 2*) states that it is always true that if the first cryptographer has not paid for dinner then she knows that some other cryptographer pays. This also is also not satisfiable by the system since this behavior only happens if the number of utterances is odd, a condition which is missing in the elicitation of the property. Finally, a third property is introduced (*Property Cheating 3*) says that the last cryptographer knows that always when

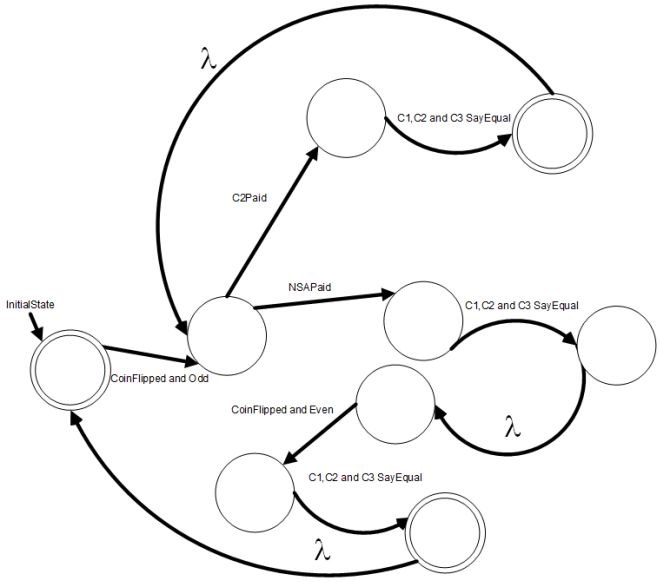

Fig. 8. A controller for the DC protocol

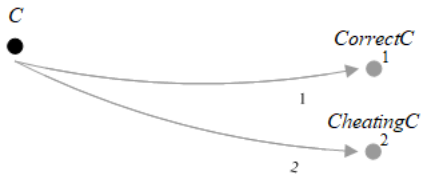

Fig. 9. Introducing cheating cryptographers

the first cryptographer behaves correctly and the number of differences is even, then any of the cryptographers is not a payer. This property is satisfied with a model including only one cheater [23]. These three rules for the cheating version are depicted in Fig. 10.

We now add an extra property, which corresponds with an instantiation of the *Response Chain pattern (with one stimuli and two responses) with After q Until r scope* [16]. In this case, the new property requires that cryptographer *C3* must be initialized as cheating or correct cryptographer only after cryptographer *C1* and *C2* are initialized. In other words, after the protocol is started an stimuli event *CoinFlipped* must be followed by responses *CorrectC1 or CheatingC1* or *CorrectC2 or CheatingC2* (cryptographers one and two are initialized either as cheating or correct cryptographers) before the initialization of cryptographer 3 occurs. FVS rule in Fig. 11 illustrates this behavior.

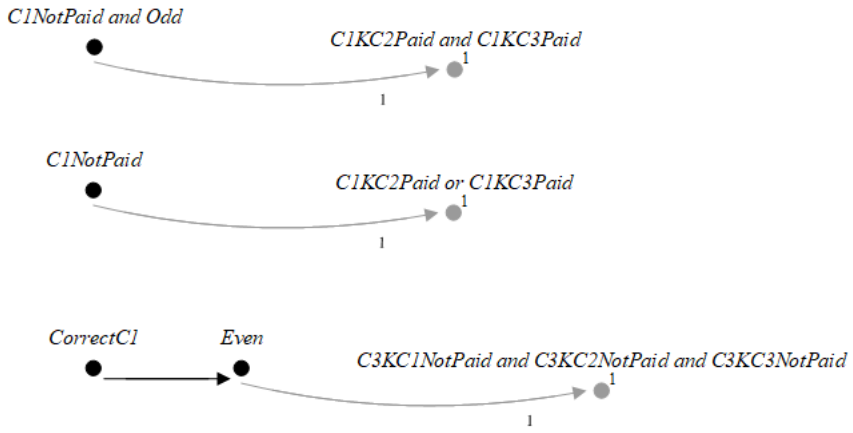

Fig. 10. Modeling the Cheating Variation's Rules

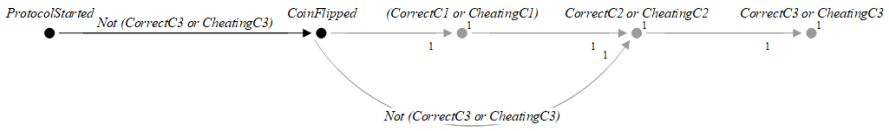

Fig. 11. An extra property for the Cheating Variation of the protocol

Finally, it is worth mentioning that we also obtained a controller for the cheating version of the protocol following the strategy described in Section 2.2. In this case, the controller includes 207 states and 671 transitions.

3.4 Some Observations

After completing this first exploratory step of FVS in agent-based systems we can point out that a complex protocol involving interesting agent communications was fully specified and verified using FVS as the specification language. What is more, a controller was found for both versions of the protocol exhibiting the flexibility and expressive power of the tool, including also open systems requirements. It is worth mentioning that the property depicted in Fig. 11 can not be represented by a Deterministic Büchi automata. Even in this conditions a controller could be found, a condition that is limitation for other synthesis approaches [33].

We are aware of some valid threats to the exposed results. First of all, more case of studies in the agent-systems world should be explored. Secondly, performance and time issues are key factors when dealing with behavioral synthesis and agent-based systems specification. In this work we only consider a model with three cryptographers. A reasonable line of future research would involve to considers more cryptographers to check scalability of our approach. Finally,

regarding expressive power, a more deeply study is needed to compare FVS expressive power to epistemic logics and other formalisms used in the literature like *CTLKD* (A CTL extension for agent-based systems) [8] or others like [36, 43, 42].

4 Related work

We share some research objectives with several approaches.

In [30, 23] agent-based systems are verified using the NuSMV model checker [13] combining logics like CTLK, CTLKD and ARCTL [7, 17, 9]. The Dining Cryptographers [12] protocol and the cheating versions is heavily analyzed in their work. In addition, in [31] the *MCMAS* model checker in introduced. Although our approach provides the possibility of obtaining a controller besides formal verification, a more extensive study to compare efficiency and expressive power with these approaches would be an undoubtedly enriching process for our line of research.

Some other approaches express agent based behavior through the use of social commitments formalisms [43, 17]. Social commitments proved to be a powerful representation for agent interactions. They, in fact, provide a social semantics that abstracts away from the agents internal states and offers social and observable meaning to agent messages exchange [43]. Social commitments are specified in a modal logical language called Probabilistic Computation Tree Logic of Commitments (PCTLC) [44]. It would be more than interesting trying to compare and analyze the impact of traditional probabilistic model checker [35] in this domain. This would involve an FVS extension to deal with probabilistic and social commitments constructors.

Efficient model checking algorithms for verifying agent-based systems are deeply analyzed in [43]. In order to improve FVS efficiency a thorough analysis of these algorithms and data structures must be done in the short future.

Work in [46] presents a novel technique employing symbolic modern checking in the context of the *Dynamic Epistemic Logic*. FVS does not feature the possibility of reasoning with symbolic model checkers. This remains as an interesting line to further expand our work. Similarly, work in [27] proposes an appealing logic combining Public Announcement Logic (PAL) [37] and Dynamic Epistemic Logic [46]. The flexibility and expressive power of these logics must be carefully considered to validate FVS contributions.

5 Future Work

We contemplate three main aspects to expand and validate the results of this work.

First, we would like to take into account crucial concepts as performance and efficiency and compare our approach against other known formalisms [30, 31, 43]. One way of addressing this issue would involve to include more cryptographers

in the case of study analyzed in Section 3 to check scalability since most of the algorithms involved in verifying and synthesising behavioral properties are based on the size of the automata which feed the tools.

In second place we would like to conduct an study to deeply analyze the expressive power of FVS compared to epistemic logics [40, 25, 1] and other agent-based systems logics like *CTLK*, *CTLKD* and *ARCTL* [7, 17, 9]. It would be interesting to explore if FVS needs to be extended to represent more sharply agent-based systems' behavior.

Lastly, we would like to perform more case of studies in the agent-based systems to continue investigating this line of research.

6 Conclusions

In this work we explore the FVS language as a formalism to express, validate and synthesize behavior in agent-based systems. Given FVS desirable features as flexility, expressive power and its ability to perform behavioral synthesis in Open Systems on one side, and the relationship between Behavioral Synthesis and agent-based systems on the other side, we analyzed FVS impact on this latter domain.

FVS was able to fully specified and verified a very well known protocol in the agent-based system field. In addition, a controller was obtained for the system. We believe the results are promising enough to continue and expand this line of research.

References

1. Arkoudas, K., Bringsjord, S.: Metareasoning for multi-agent epistemic logics. In: International Workshop on Computational Logic in Multi-Agent Systems. pp. 111–125. Springer (2004)
2. Asteasuain, F., Braberman, V.: Declaratively building behavior by means of scenario clauses. Requirements Engineering **22**(2), 239–274 (2017), doi:10.1007/s00766-015-0242-2
3. Asteasuain, F., Calonge, F., Dubinsky, M.: Exploring specification pattern based behavioral synthesis with scenario clauses. In: CACIC (2018)
4. Asteasuain, F., Calonge, F., Gamboa, P.: Behavioral synthesis with branching graphical scenarios. In: CONAIISI (2019)
5. Baldoni, M., Baroglio, C., Martelli, A., Patti, V., Schifanella, C.: Verifying protocol conformance for logic-based communicating agents. In: International Workshop on Computational Logic in Multi-Agent Systems. pp. 196–212. Springer (2004)
6. Bentahar, J., El-Menshawy, M., Qu, H., Dssouli, R.: Communicative commitments: Model checking and complexity analysis. Knowledge-Based Systems **35**, 21–34 (2012)
7. Bentahar, J., Meyer, J.J., Wan, W.: Model checking communicative agent-based systems. Knowledge-Based Systems **22**(3), 142–159 (2009)
8. Bentahar, J., Moulin, B., Meyer, J.J.C., Chaib-draa, B.: A logical model for commitment and argument network for agent communication. In: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 2. pp. 792–799. IEEE Computer Society (2004)

9. Biswas, P.K.: Towards an agent-oriented approach to conceptualization. Applied Soft Computing **8**(1), 127–139 (2008)
10. Bohy, A., Bruyère, V., Filiot, E., Jin, N., Raskin, J.F.: Acacia+, a tool for ltl synthesis. In: International Conference on Computer Aided Verification. pp. 652–657. Springer (2012)
11. Bordini, R.H., Fisher, M., Pardavila, C., Wooldridge, M.: Model checking agentspeak. In: Proceedings of the second international joint conference on Autonomous agents and multiagent systems. pp. 409–416 (2003)
12. Chaum, D.: The dining cryptographers problem: Unconditional sender and recipient untraceability. Journal of cryptology **1**(1), 65–75 (1988)
13. Cimatti, A., Clarke, E., Giunchiglia, F., Roveri, M.: Nusmv: A new symbolic model verifier. In: International conference on computer aided verification. pp. 495–499. Springer (1999)
14. DIppolito, N., Braberman, V., Piterman, N., Uchitel, S.: Synthesising non-anomalous event-based controllers for liveness goals. ACM Tran **22**(9) (2013)
15. D'Ippolito, N., Rodrıguez, N., Sardina, S.: Fully observable non-deterministic planning as assumption-based reactive synthesis. Journal of Artificial Intelligence Research **61**, 593–621 (2018)
16. Dwyer, M., Avrunin, M., Corbett, M.: Patterns in property specifications for finite-state verification. In: ICSE. pp. 411–420 (1999)
17. El Menshawy, M., Bentahar, J., El Kholy, W., Dssouli, R.: Reducing model checking commitments for agent communication to model checking arctl and gctl. Autonomous agents and multi-agent systems **27**(3), 375–418 (2013)
18. Endriss, U., Maudet, N., Sadri, F., Toni, F.: Protocol conformance for logic-based agents (2003)
19. Fagiolo, G., Guerini, M., Lamperti, F., Moneta, A., Roventini, A.: Validation of agent-based models in economics and finance. In: Computer Simulation Validation, pp. 763–787. Springer (2019)
20. Giordano, L., Martelli, A., Dupré, D.T.: Reasoning about actions with temporal answer sets. Theory and Practice of Logic Programming **13**(2), 201–225 (2013)
21. Giordano, L., Martelli, A., Schwind, C.: Specifying and verifying interaction protocols in a temporal action logic. Journal of Applied Logic **5**(2), 214–234 (2007)
22. Jonsson, B., Larsen, K.G.: Specification and refinement of probabilistic processes. In: [1991] Proceedings Sixth Annual IEEE Symposium on Logic in Computer Science. pp. 266–277. IEEE (1991)
23. Kacprzak, M., Lomuscio, A., Niewiadomski, A., Penczek, W., Raimondi, F., Szreter, M.: Comparing bdd and sat based techniques for model checking chaum's dining cryptographers protocol. Fundamenta Informaticae **72**(1-3), 215–234 (2006)
24. Kacprzak, M., Lomuscio, A., Penczek, W.: Verification of multiagent systems via unbounded model checking. In: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems, 2004. AAMAS 2004. pp. 638–645. IEEE (2004)
25. Kaneko, M.: Epistemic logics and their game theoretic applications: Introduction. Economic Theory **19**(1), 7–62 (2002)
26. Khan, F., Reyad, O.: Application of intelligent multi agent based systems for e-healthcare security. arXiv preprint arXiv:2004.01256 (2020)
27. Knight, S., Maubert, B., Schwarzentruber, F.: Reasoning about knowledge and messages in asynchronous multi-agent systems. Mathematical Structures in Computer Science **29**(1), 127–168 (2019)
28. Krka, I., Brun, Y., Edwards, G., Medvidovic, N.: Synthesizing partial component-level behavior models from system specifications. In: ESEC-FSE

29. Larrañaga, P., Moral, S.: Probabilistic graphical models in artificial intelligence. Applied soft computing **11**(2), 1511–1528 (2011)
30. Lomuscio, A., Pecheur, C., Raimondi, F.: Automatic verification of knowledge and time with nusmv. In: Proceedings of the Twentieth International Joint Conference on Artificial Intelligence. pp. 1384–1389. IJCAI/AAAI Press (2007)
31. Lomuscio, A., Qu, H., Raimondi, F.: Mcmas: an open-source model checker for the verification of multi-agent systems. International Journal on Software Tools for Technology Transfer **19**(1), 9–30 (2017)
32. Maler, O., Ničković, D.: Monitoring properties of analog and mixed-signal circuits. International Journal on Software Tools for Technology Transfer **15**(3), 247–268 (2013)
33. Maoz, S., Ringert, J.O.: Synthesizing a lego forklift controller in gr (1): A case study. arXiv preprint arXiv:1602.01172 (2016)
34. Van der Meyden, R., Suf, K.: Symbolic model checking the knowledge of the dining cryptographers. In: Proceedings. 17th IEEE Computer Security Foundations Workshop, 2004. pp. 280–291. IEEE (2004)
35. Pavese, E., Braberman, V., Uchitel, S.: Automated reliability estimation over partial systematic explorations. In: 2013 35th International Conference on Software Engineering (ICSE). pp. 602–611. IEEE (2013)
36. Penczek, W., Lomuscio, A.: Verifying epistemic properties of multi-agent systems via bounded model checking. Fundamenta Informaticae **55**(2), 167–185 (2003)
37. Plaza, J.: Logics of public communications. Synthese **158**(2), 165–179 (2007)
38. Raimondi, F., Lomuscio, A.: Automatic verification of multi-agent systems by model checking via ordered binary decision diagrams. Journal of Applied Logic **5**(2), 235–251 (2007)
39. Rathnakumar, A.J., Balakrishnan, S.: Design of multi-agent based systems for entrusted communication using jade. Taga Journal of Graphic Technology **14**, 766–774 (2018)
40. Ruspini, E.H.: Epistemic logics, probability, and the calculus of evidence. In: Proceedings of the 10th international joint conference on Artificial intelligence-Volume 2. pp. 924–931 (1987)
41. Sardina, S., D'Ippolito, N.: Towards fully observable non-deterministic planning as assumption-based automatic synthesis. In: Twenty-Fourth International Joint Conference on Artificial Intelligence (2015)
42. Singh, M.P.: A social semantics for agent communication languages. In: Issues in agent communication, pp. 31–45. Springer (2000)
43. Sultan, K., Bentahar, J., El-Menshawy, M.: Model checking probabilistic social commitments for intelligent agent communication. Applied Soft Computing **22**, 397–409 (2014)
44. Sultan, K., El Menshawy, M., Bentahar, J.: Reasoning about social commitments in the presence of uncertainty. In: 2013 IEEE 12th International Conference on Intelligent Software Methodologies, Tools and Techniques (SoMeT). pp. 29–35. IEEE (2013)
45. Tsay, Y.K., Chen, Y.F., Tsai, M.H., Wu, K.N., Chan, W.C.: Goal: A graphical tool for manipulating büchi automata and temporal formulae. In: TACAS. pp. 466–471. Springer (2007)
46. Van Benthem, J., Van Eijck, J., Gattinger, M., Su, K.: Symbolic model checking for dynamic epistemic logic. In: International Workshop on Logic, Rationality and Interaction. pp. 366–378. Springer (2015)