On Software Complexity of Agent-Oriented Logic Programs: an Empirical Analysis

AMELIA BĂDICĂ, COSTIN BĂDICĂ and ION BULIGIU, University of Craiova
MIRJANA IVANOVIĆ, University of Novi Sad
MARIA GANZHA, Polish Academy of Sciences and Warsaw University of Technology
MARCIN PAPRZYCKI, Polish Academy of Sciences and Warsaw Management Academy

The aim of this paper is to briefly introduce some of our experiences with agent oriented programming, in particular combined with logic programming paradigm, in the context of software engineering, and with a focus on formal approaches for software development. The paper briefly reports some of the lessons learnt from carrying out few small research applications, employing agent-oriented logic programming languages including ECLiPSe-CLP and AgentSpeak(L) / Jason. The main contribution is to present complexity results of few small agent-oriented programming projects based on logical reasoning. Such approaches are traditionally the focus of artificial intelligence research, while our concern here is on the more pragmatic perspective of software complexity.

1. INTRODUCTION

Recently we have been involved in a number of research activities concerning the application of artificial intelligence programming techniques for the implementation of agent-based programs in different domains. We focused on combining agent-oriented and logic programming, following the knowledge representation and reasoning tradition of classical artificial intelligence. Nevertheless, this work targeted various application domains, including logistics, patrolling games, e-business agents, reinforcement learning, as well as modeling and simulation of dynamic systems.

The aim of this paper is to briefly introduce some of our experiences obtained by carrying out these tasks. We highlight the characteristics of agent oriented logic programming in the context of software engineering, and with a focus on formal approaches for software development. The paper briefly reports some of the lessons learnt from the practical application of agent-oriented and logic programming languages including ECLiPSe-CLP and AgentSpeak(L) / Jason. Our main intention is to promote agent-oriented programming based on logical reasoning that is traditionally the focus of artificial intelligence research, from the more pragmatic perspective of software engineering practices.

Agent-oriented and logic programming were traditionally promoted by the multi-agent systems and artificial intelligence communities. They are usually associated to artificial intelligence and intelligent agents academic curricula, as well as to research projects in artificial intelligence. Nevertheless, an increasing practical interest in these topics is also manifested inside the software engineering commu-

Authors address: Amelia Bădică, 13, A.I. Cuza Street, Craiova, 200585, Romania; e-mail: ameliabd@yahoo.com; Costin Bădică, Blvd. Decebal nr. 107, RO-200440, Craiova, Romania; e-mail: cbadica@software.ucv.ro; Ion Buligiu, 13, A.I. Cuza Street, Craiova, 200585, Romania; e-mail: buligiu.ion@gmail.com; Mirjana Ivanović, Trg Dositeja Obradovica 4, 21000 Novi Sad, Serbia; e-mail: mira@dmi.uns.ac.rs; Maria Ganzha, ul. Newelska 6, 01-447 Warsaw, Poland; e-mail: maria.ganzha@ibspan.waw.pl; Marcin Paprzycki, ul. Newelska 6, 01-447 Warsaw, Poland; e-mail: paprzyck@ibspan.waw.pl;

In: Z. Budimac (ed.): Proceedings of the SQAMIA 2018: 7th Workshop of Software Quality, Analysis, Monitoring, Improvement, and Applications, Novi Sad, Serbia, 27–30.8.2018. Also published online by CEUR Workshop Proceedings (http://ceur-ws.org, ISSN 1613-0073)

nity, that focuses on sound and robust software development methodologies, in particular showcasing formal approaches based on computational logic. Different flavors of logic programming, including for example constraint logic programming (CLP hereafter) or satisfiability checking (SAT) have been proposed for tackling various software engineering problems such as: testing, verification, symbolic execution, and quality assurance [Meudec 2001; Doyle and Meudec 2003; Anderson et al. 2005; Rojas and Gómez-Zamalloa 2013; Visser et al. 2014]. On the other hand, agent-oriented logic programming practices are by far less mature and under-represented from the software engineering point of view than other programming paradigms, like for example object-orientation.

The main contribution of this paper is to present complexity results of few small agent-oriented programming projects based on logical reasoning. Such approaches are traditionally the focus of artificial intelligence research, while our concern here is on the more pragmatic perspective of software complexity. We employ basic complexity measures specific to rule-based and logic programming, adapted to the agent-oriented context, and we provide estimations of these measures for the projects briefly reviewed in this paper. In our opinion these results could be of interest to both artificial intelligence and software engineering research communities.

Our paper is structured as follows. We start in Section 2 with a brief overview of agent-oriented logic programming. In Section 3 we introduce a number of small research projects that employ agent-oriented logic programming in different application areas, including logistics, reinforcement learning, as well as modeling and simulation of dynamic systems. For each application we provide summary complexity figures emphasizing various aspects, like size of the knowledge base, size of belief base, size of plan base, number of agents, as well as number of agent interactions. Section 4 concludes the paper and points to future works.

2. LOGIC-BASED AGENT-ORIENTED PROGRAMMING

Computational logic has a quite long tradition in the field of classical artificial intelligence that advocates the use of logic for knowledge representation and reasoning. This led to the development of a plethora of AI methods and languages which allow the capturing of the cognitive and problem solving capabilities of intelligent agents.

2.1 Logic Programming

Logic Programming is a programming paradigm comprising a collection of declarative programming languages that use logic as the underlying computational mechanism of intelligent agents [Levesque 2012]. The prototypical language of this class is Prolog, while state-of-the-art extensions refer to Answer Set Programming and Constraint Logic Programming.

A logic program is a set of facts and a set of rules. In the context of intelligent agents, a logic program is sometimes called knowledge base. Facts are atomic formulae composed of a predicate name and a number of arguments. Arguments can be constants, variables or compound terms. Facts can be used to capture the description of a situation, problem case or an agent's belief base.

Prolog rules are logical formulae equivalent to definite clauses. A definite clause is a disjunction of literals (direct or negated atomic formulas) such that exactly one literal is positive. Rules can be used to capture reasoning patterns of problem solving strategies of intelligent agents.

Intelligence is extracted, consulted and applied using queries addressed to the agent's knowledge base. A query may have an external origin from another agent or human, seeking for consultancy of the intelligent agent or internally, from the intelligent agent itself through introspection mechanisms, during the problem solving process.

CLP in general and ECLiPSe-CLP in particular distinguish between normal Prolog predicates as used by standard Prolog and constraints that are specific to CLP [Schimpf and Shen 2012]. This distinction is mandatory as constraints are handled by specialized constraint satisfaction algorithms that provide more efficient problem solving methods than the standard Prolog's backtracking search.

2.2 Agent Oriented Programming Using AgentSpeak(L) / Jason

AgentSpeak(L) is an abstract agent-oriented programming language introduced in [Rao 1996]. Jason is a Java-based implementation, as well as an extension of AgentSpeak(L) [Bordini et al. 2007].

AgentSpeak(L) follows the paradigm of practical reasoning, i.e. reasoning directed towards actions, and it provides an implementation of the Belief-Desire-Intention (BDI hereafter) architecture of software agents. According to this view an agent is a software module that (i) provides a software interface with the external world and (ii) contains three components: belief base, plan library and reasoning engine.

The agent's external world consists of the outer environment, as well as possibly other agents. The agent interface provides three elements: sensing interface, actuation interface and communication interface. The integration of agents with their external environment can be done using a Java API.

The belief base defines what an agent "knows" or "believes" about its environment at a certain time point. The BDI architecture does not impose a specific structuring of the belief base other than as a generic container of beliefs. By default AgentSpeak(L) and Jason are using a logical model of beliefs by structuring the belief base as a logic program composed of facts and rules. So, the agent can reason about its own beliefs using logic programming.

The plan library defines the agent's "know-how" structured as a set of behavioral elements called plans. A plan follows the general pattern of event-condition-action rules and it is composed of three elements: triggering event, context and body. Events are triggered by belief and goal updates. AgentSpeak(L) distinguishes between test goals and achievement goals. Test goals closely correspond to logic programming queries addressed to the agent belief base. Moreover, plan context matching with agent's belief base is also using logic programming queries.

The reasoning engine implements an update-select-act cycle that includes several activities: perception and communication, event handling and plan selection, as well as action selection and execution.

3. APPLICATIONS AND DISCUSSION

In this section we introduce a series of small applications having in common the use of logic and agent-oriented programming for problem solving in the following areas: optimization, learning, and simulation. These applications mainly serve research needs, but they can be also used in an educational setting centered on artificial intelligence and / or multi-agent systems curricula. We provide a brief description of each application, highlighting the role of agents and logic.

3.1 Declarative Approach to Freight Brokering

Freight brokering business aims to coordinate transportation arrangements of transport customers with transport resource providers. In this work we focused on the main function of this business to find available trucks and to define their feasible routes for serving customer transportation requests. We proposed a knowledge-based freight broker based on agents and constraint programming.

The brokering function is defined as a special type of vehicle routing with pickup and delivery problem. Our main achievement was the development of a declarative optimization model based on constraint logic programming [Bădică et al. 2018b]. This model was implemented and evaluated using the state-of-the-art ECLiPSe-CLP engine [Schimpf and Shen 2012].

We focus on the freight broker agent that mediates the indirect interaction between customer agents and freight transportation provider agents. This agent records the transport requests issued by customer agents, records provisions of transportation resources, including available trucks and/or vehicles, as well as their characteristics, and determines feasible and possibly optimal allocations of transport resources to transport requests.

The freight broker agent is a typical knowledge-based agent composed of knowledge base and inference engine. The knowledge base is structured into: facts, rules & queries, and constraints. The input description of the scheduling problem (vehicles characteristics, customer orders, transportation hops) is converted to an internal representation captured by a set of logical facts that share a predefined and self-explanatory schema. The constraints are general and they represent reusable knowledge for solving different problem instances. The rules and queries define additional predicates, including the generic search predicates, that are internally used by the agent to solve the given transportation scheduling problem. The inference engine is represented by a constraints solver engine.

Useful metrics of the freight broker agent refer to the size of the knowledge base estimated by the number of facts, number of rules, and number of constraints.

3.2 Reinforcement Learning

This research was conducted in the context of approaching reinforcement learning (RL hereafter), a typical task calling for intelligence of agents acting in an environment [Sutton and G. Barto 1998], within the framework of agent-oriented programming. Agents, single or in group, can use RL to learn how to choose their actions in an uncertain, dynamic and possibly unknown environment, in order to improve their long-term utility.

Our research was concerned with somehow trying to narrow the gap between agent research and the practical use of agent-oriented programming. In particular we approached classical RL methods based on temporal-difference learning (TDL hereafter) and Q-learning using BDI languages. Using our proposal, BDI agents programmed in Jason can be endowed with RL skills, possibly resulting in new forms of hybrid reasoning that can benefit from combining BDI reasoning and machine learning into a unique cognitive architecture [Bădică et al. 2015].

Our prototype system was designed to serve the purpose of assisting researchers in experimenting with BDI learning agents [Bădică et al. 2018a]. It contains RL agents situated in a two dimensional grid environment. Agents can take one step up, down, left or right in this grid. The result of their action is stochastic, i.e. it can produce a successful move to the desired direction, or an erroneous move by slightly deviating to left or right from the desired direction. The action's effect uncertainty captures the agents ignorance about their environment.

A basic abstraction for the development of our system was to decouple the agents from their environment. The simple sensing and acting interface provided by Jason platform proved very useful to effortlessly achieve this desiderate. Agents can use their sensorial capabilities to perceive their environment, while percepts can provide reward and state information, according to the RL paradigm. Moreover, agents can use their acting capabilities specific to practical reasoning in order to perform their actions, again complying with the RL paradigm. This model can be easily extended to multiple agents sharing their environment that can act either asynchronously (in isolation) through independent actions or synchronously (in coordination) through joint actions.

The specific RL method was encoded using event-condition-action plans of BDI agents. We devised BDI representations for the basic TDL approach in passive context, when the agent is only interested to accurately estimate its long-term utility, as well as for classic Q-learning and SARSA methods in active context, when the agent is interested to improve its long-term utility. Finally, taking into account the rigor of our approach, we have reasons to believe that this method can be adapted for agent-oriented programming languages following the BDI paradigm, thus supporting its generality [Bădică et al. 2011; Kravari and Bassiliades 2015].

Useful metrics for the RL agents are: size of the belief base in number of facts and number of rules, as well as size of plan base in number of plans and number of agent actions.

3.3 Modelling and Simulation of Dynamic Systems

In this section we consider three research applications having in common the modelling and simulation of continuous and discrete dynamic systems using BDI agents implemented in Jason in the areas of game theory, business processes and ecology.

3.3.1 *Patrolling Games.* This project was focused on developing a BDI-based modelling and simulation framework of patrolling games. This topic is part of the larger domain of game theory for security that is interested in the development of intelligent defence strategies for enhancing the trustworthiness of cyber-physical systems.

We introduce a formal model of two-person patrolling game as Bayesian normal form game [Leyton-Brown and Shoham 2008]. This game involves two agents (robber and guardian) and it is suitable for the scheduling of guardian patrols in surveillance applications. The model is mapped to agent-based simulation using Jason agent-oriented language [Bădică et al. 2017].

The application involves two BDI agents representing the robber and the guardian that iteratively play the patrolling game in a sequence of rounds. During each round the agents submit a joint action to the game environment that reacts by providing to each of them the corresponding utilities in the form of agent percepts[1]. In our experiments agents adopt mixed strategies that are fixed during a playing round [Leyton-Brown and Shoham 2008]. Sampling of mixed strategy spaces and agent decision making based on these strategies were implemented using logic programming.

We can observe that this application uses the same model as the RL application, by letting the agents to indirectly interact by jointly acting upon and separately perceiving their environment. The game logic is encapsulated by the environment that exposes a sense and act interface allowing the agents to play the game and to receive the corresponding utilities. This framework can be used to experiment with more complex agent strategies possibly combining results of game theory and RL.

Similarly to the RL application, useful metrics for the robber and guardian agents are: size of the belief base and size of plan base.

3.3.2 *Knowledge Based Business Agents.* This project was focused on modeling and enactment of business processes using state-of-the-art agent-oriented programming. In particular we proposed a method of capturing business process models expressed using role-activity diagrams [Ould 2005] by AgentSpeak(L) / Jason programs. This approach of combining sound methods of business process and agent-oriented modeling paves the way for the development of agent-based business organizations.

Roles chunk business processes into units of responsibility that structure the set of activities they are carrying out into single of multiple threads of work. Each role is mapped to a proactive Jason agent, while the state of each role is captured by the set of beliefs of its corresponding agent. State transitions are modeled by event-condition-action rules. Role coordination is achieved by letting involved agents share their beliefs about their current state by direct interaction via message exchange.

Our main achievement in this work was the generalization of the mapping of role activity diagrams to Jason by proposing a generic knowledge-based business agent architecture ($\mathcal{KB}^2\mathcal{A}^2$ in what follows) [Bădică et al. 2016]. $\mathcal{KB}^2\mathcal{A}^2$ proposes the configuration of each agent representing a role of a business process using the following components: i) a knowledge base defined as the set of facts capturing the operational knowledge of the role; ii) a set of template plans capturing the generic behavioral patterns of business agents.

[1]https://github.com/IntelligentDistributedSystems/Antoine-Conor

Some useful metrics of a $\mathcal{KB}^2\mathcal{A}^2$ compliant multi-agent system based on are: number of agents, size of the knowledge base as number of facts, and number of agent interactions. Note that the plan base is unique and shared by all the agents of the business organization.

3.3.3 *Modeling and Simulation of Graph-Based Predator-Prey Systems.* The goal of this research was to investigate the suitability of multi-agent systems based on BDI architecture for capturing the details of the simulation models of continuous dynamic systems, as encountered in science and engineering. Our achievement is the proposal of a framework based on BDI agents for the macroscopic modeling and simulation of continuous dynamic systems. Our framework was used for the modeling and simulation of an ecological system comprising a number of species that behave according to a given set of predator-prey relationships described by an acyclic directed graph [Bădică et al. 2018].

While traditionally, agent-based modeling has been mostly applied for micro-modeling and micro-simulation, this work can be seen as a contribution of employing BDI-based agent-oriented programming for the macro-modeling and macro-simulation of dynamic systems.

The idea was to break down the target system model into a collection of autonomous and loosely-coupled interacting components endowed with message-based interfaces and local intelligence. Each component has a type that precisely characterizes its behavior as a (possibly state-based) mathematical function, as well as a finite number of inputs and outputs.

Each component is mapped to a Jason agent that captures its state as a set of logical facts and its behavioral patterns as a set of plans. Agents are configured to include: the agent type, the set of parameters completely defining the mathematical function performed by the agent, the component state, and the agent acquaintance model that is necessary for the dissemination of simulation information.

This approach provides a natural mapping of a dynamic system model to a distributed computational system that can bring modularity, scalability, reusability, and flexibility to a network-based implementation. The building blocks of our solution are highly reusable. Their configuration is simply achieved by initializing accordingly the agents' belief bases.

Useful metrics include number of agents and number of agent types. Agents of a given type share their plan base, while agent interaction is achieved by message passing, like the $\mathcal{KB}^2\mathcal{A}^2$ architecture.

3.4 Discussion

The analysis of software complexity is an important problem in software engineering. Moreover, multi-agent systems are a special type of software developed using multi-agent platforms and languages. Therefore, their systematic analysis from the complexity and performance points of view attracted the interest of the computer science research community [Camacho and Aler 2005; Jordan and Collier 2012; Ivanović et al. 2016].

In this section we propose a brief comparative analysis of the software projects introduced in this paper, based on their quantitative and qualitative characteristics. In principle, we can reuse already existing analysis proposals and frameworks. Nevertheless, our endeavour is a bit special because our work is focused on logic based agent-oriented programming languages. The research literature contains also proposals for measuring software complexity of logic programs, in particular based on Prolog language [Moores 1998], as well as of rule-based expert systems [Chen and Suen 1994]. These works provided us some source of inspiration, as reasoning with agent beliefs in Jason is actually based on logic programming, while Jason plans resemble some-how production rules of expert systems.

In what follows we consider the five agent-based software projects presented in this paper from the following perspectives. The resulted summary figures are presented in Table I.

Table I. : Summary complexity figures for 5 logic-based agent-oriented applications

No.	Description	# Agents	Agent Interaction	Belief Base		# Plans	# Constraints	Notes
				# Facts	# Rules			
1	Freight broker agent	1	direct	$4+n+t+k^2$	17	-	$n(t+2)+\frac{mn(m+5)}{2}+(2+t)n-1$	$n =$ # orders $t =$ # trucks $k =$ # locations $m =$ # hops, $k \leq m \leq 2n$ [Bădică et al. 2018b]
2	RL agent	1	indirect	$14+n$	19	21	-	Many RL agents can run independently $n =$ # visited states
3	Patrolling game simulation	2	indirect	5	14	7	-	Evaluation is done only for guardian agent [Bădică et al. 2017]
4	$\mathcal{KB}^2\mathcal{A}^2$-based project management process	3	direct	22	11	18	-	Evaluation is done only for designer agent [Bădică et al. 2016]
5	Predator-prey continuous dynamic system	20	direct	6	0	6	-	Evaluation is done only for integrator agent [Bădică et al. 2018]

1. In the case of the freight brokering system introduced in Section 3.1, the analysis is exclusively focused on the freight broker agent.
2. In the case of RL agents introduced in Section 3.2, the analysis is focused on the implementation of classic RL strategies in Jason (so the Java-based implementation of the environment, as well as the other components of the system that use classical object-oriented programming are ignored).
3. In the case of the patrolling game simulation introduced in Section 3.3.1, we focus only on the guardian and robber agents, while the Java-based implementation of the environment, as well as the other components of the system are omitted.
4. In the case of $\mathcal{KB}^2\mathcal{A}^2$ framework introduced in Section 3.3.2, we focus on the sample project management process presented in [Bădică et al. 2016].
5. In the case of the BDI-based framework for the modeling and simulation of continuous dynamic systems from Section 3.3.3, we consider the predator-prey system introduced in [Bădică et al. 2018].

Table I reveals interesting aspects. In 2 projects (1 and 2) the number of facts in the belief base depends on the problem instance, actually increasing with the values of the input parameters of the problem. On the other hand, number of rules and plans are constant for each agent. Finally, for the constraint-based agent (project 1), the number of constraints also depends on the problem instance, actually increasing drastically with the values of the input parameters of the problem. Frankly speaking, constrains are actually represented as Prolog facts, so this observation is consistent with our finding referring to the number of facts of the belief base.

Moreover, while application 3 can be described as relatively simple, involving the modeling and simulation of the strategic interaction of two intelligent agents, applications 4 and 5 actually incorporate hidden sources of complexity that are not immediately noticeable from the quantitative figures presented in Table I. The sources are represented by the size of the the RAD process model, respectively by the size of the dynamic system under consideration. They directly impact the size of the agents' belief base and the number of agents (in project 4), as well as the number of agents (in project 5).

Finally, it is interesting to observe that with the modular approach employed by project 5, the complexity of the dynamic system under consideration is reflected into the complex interaction structure (acquaintance model) of the multi-agent system, while the internal model, as well as the belief base of each agent are kept as simple as possible.

This simple analysis provides some useful insights into the main sources of complexity of knowledge-based applications based on logical reasoning. We can observe that, in our simple projects, the number of logical facts (representing agents' belief bases or problem constraints), as well as the number of agents and their interactions can represent the main source of complexity. Both (beliefs and constraints) are actually derived from the domain conceptualization that in our opinion plays a crucial aspect in knowledge-based systems. This supports the conclusion that the knowledge engineer's experience, as well as his or her insight into the problem domain have a significant impact on the provided solution. Finally, the number of agents and their interactions clearly depend on the problem solving approach, also carrying out a significant impact on the provided solution.

4. CONCLUSIONS AND FUTURE WORKS

We briefly reviewed from a software engineering perspective, several research prototypes employing logic-based agent programming. We provided a comparative analysis of summary complexity figures of these projects. As future work, this analysis can be further deepened and consequently the conclusions strengthened, by considering more complexity figures, possibly applied to other research prototypes involving logic-based agent programming. This will involve at least two aspects that we plan to consider in our future work: i) expanding the repository of considered applications with more projects in the area of agent-based logic programming; one source could be the sample applications and examples available within Jason distribution; ii) developing a more systematic analysis approach, possibly enabled by a suitable tool support.

ACKNOWLEDGMENT

This paper is a part of the Serbia-Romania-Poland collaboration within multilateral agreement on "Agent systems and applications" and Romania-Poland collaboration within bilateral project "Semantic foundation of the Internet of Things".

REFERENCES

Bonnie Brinton Anderson, James V. Hansen, Paul Benjamin Lowry, and Scott L. Summers. 2005. Model checking for E-business control and assurance. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 35, 3 (2005), 445–450. DOI:http://dx.doi.org/10.1109/TSMCC.2004.843181

Amelia Bădică, Costin Bădică, Mirjana Ivanović, and Dana Dănciulescu. 2018. Multi-agent modelling and simulation of graph-based predator-prey dynamic systems: A BDI approach. *Expert Systems* (Jun 2018). DOI:http://dx.doi.org/10.1111/exsy.12263

Amelia Bădică, Costin Bădică, Mirjana Ivanović, and Dejan Mitrovic. 2015. An Approach of Temporal Difference Learning Using Agent-Oriented Programming. In *Proceedings of the 20th International Conference on Control Systems and Computer Science (CSCS'2015)*. IEEE, 735–742. DOI:http://dx.doi.org/10.1109/CSCS.2015.71

Amelia Bădică, Costin Bădică, Florin Leon, and Ionuţ Buligiu. 2016. Modeling and Enactment of Business Agents Using Jason. In *Proceedings of the 9th Hellenic Conference on Artificial Intelligence, SETN 16*. ACM, 376–380. DOI:http://dx.doi.org/10.1145/2903220.2903253

Costin Bădică, Amelia Bădică, Catalina Sitnikov, and Florin Leon. 2017. A Formal Model of Patrolling Game and its Agent-Based Simulation Using Jason. In *Proceedings of the 25th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP2017)*. IEEE, 376–380. DOI:http://dx.doi.org/10.1109/PDP.2017.95

Costin Bădică, Alex Becheru, and Samuel Felton. 2018a. Integration of Jason Reinforcement Learning Agents into an Interactive Application. In *Post-Proceedings of the 19th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, SYNASC 2017*. IEEE. in press.

Costin Bădică, Florin Leon, and Amelia Bădică. 2018b. Freight transportation broker agent based on constraint logic programming. *Evolving Systems* (Jun 2018). DOI:http://dx.doi.org/10.1007/s12530-018-9230-3

Rafael H. Bordini, Jomi Fred Hübner, and Michael Wooldridge. 2007. *Programming Multi-Agent Systems in AgentSpeak Using Jason*. John Wiley & Sons.

Costin Bădică, Zoran Budimac, Hans-Dieter Burkhard, and Mirjana Ivanović. 2011. Software agents: Languages, tools, platforms. *Computer Science and Information Systems* 8, 2 (2011), 255–298. DOI:http://dx.doi.org/10.2298/CSIS110214013B

David Camacho and Ricardo Aler. 2005. Software and Performance Measures for Evaluating Multi-Agent Frameworks. *Applied Artificial Intelligence* 19, 6 (2005), 645–657. DOI:http://dx.doi.org/10.1080/08839510590962050

Zhisong Chen and Ching Y. Suen. 1994. Measuring the complexity of rule-based expert systems. *Expert Systems with Applications* 7, 4 (1994), 467–481. DOI:http://dx.doi.org/10.1016/0957-4174(94)90072-8

J. Doyle and Christophe Meudec. 2003. IBIS: An Interactive Bytecode Inspection System, Using Symbolic Execution and Constraint Logic Programming. In *Proceedings of the 2nd International Conference on Principles and Practice of Programming in Java – PPPJ '03*. Computer Science Press, Inc., New York, NY, USA, 55–58. http://dl.acm.org/citation.cfm?id=957289.957307

Mirjana Ivanović, Marcin Paprzycki, Maria Ganzha, Costin Bădică, and Amelia Bădică. 2016. Software Metrics for Agent Technologies and Possible Educational Usage. In *Proceedings of the Fifth Workshop on Software Quality Analysis, Monitoring, Improvement, and Applications, SQAMIA 2016*, Zoran Budimac, Zoltán Horváth, and Tamás Kozsik (Eds.), Vol. 1677. CEUR Workshop Proceedings, 17–27. http://ceur-ws.org/Vol-1677/paper03.pdf

Howell R. Jordan and Rem Collier. 2012. Evaluating Agent-oriented Programs: Towards Multi-paradigm Metrics. In *Proceedings of the 8th International Conference on Programming Multi-Agent Systems – ProMAS'10*. Lecture Notes in Computer Science, Vol. 6599. Springer-Verlag, Berlin, Heidelberg, 63–78. DOI:http://dx.doi.org/10.1007/978-3-642-28939-2_4

Kalliopi Kravari and Nick Bassiliades. 2015. A Survey of Agent Platforms. *Journal of Artificial Societies and Social Simulation* 18, 1 (2015), 11. DOI:http://dx.doi.org/10.18564/jasss.2661

Hector J. Levesque. 2012. *Thinking as Computation: A First Course*. The MIT Press.

Kevin Leyton-Brown and Yoav Shoham. 2008. *Essentials of Game Theory: A Concise Multidisciplinary Introduction*. Morgan & Claypool. DOI:http://dx.doi.org/10.2200/S00108ED1V01Y200802AIM003

Christophe Meudec. 2001. ATGen: automatic test data generation using constraint logic programming and symbolic execution. *Software: Testing, Verification and Reliability* 11, 2 (2001), 81–96. DOI:http://dx.doi.org/10.1002/stvr.225

Trevor T Moores. 1998. Applying complexity measures to rule-based prolog programs. *Journal of Systems and Software* 44, 1 (1998), 45–52. DOI:http://dx.doi.org/10.1016/S0164-1212(98)10042-0

Martyn A. Ould. 2005. *Business Process Management: A Rigorous Approach*. Meghan Kiffer Pr.

Anand S. Rao. 1996. AgentSpeak(L): BDI agents speak out in a logical computable language. In *Agents Breaking Away: Proc.of the 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World, MAAMAW '96*, Walter Van de Velde and John W. Perram (Eds.). Lecture Notes in Computer Science, Vol. 1038. Springer, 42–55. DOI:http://dx.doi.org/10.1007/BFb0031845

José Miguel Rojas and Miguel Gómez-Zamalloa. 2013. A Framework for Guided Test Case Generation in Constraint Logic Programming. In *Logic-Based Program Synthesis and Transformation*, Elvira Albert (Ed.). Lecture Notes in Computer Science, Vol. 7844. Springer Berlin Heidelberg, Berlin, Heidelberg, 176–193.

Joachim. Schimpf and Kish Shen. 2012. ECLiPSe – from LP to CLP. *Theory and Practice of Logic Programming* 12, 1-2 (2012), 127–156. DOI:http://dx.doi.org/10.1017/S1471068411000469

Richard S. Sutton and Andrew G. Barto. 1998. *Reinforcement Learning: An Introduction*. The MIT Press, Cambridge, MA,.

Willem Visser, Nikolaj Bjørner, and Natarajan Shankar. 2014. Software Engineering and Automated Deduction. In *Proceedings of the on Future of Software Engineering – FOSE'2014*. ACM, New York, NY, USA, 155–166. DOI:http://dx.doi.org/10.1145/2593882.2593899