Toward a Requirements-Driven Neuro-Symbolic Approach to Developing Trustworthy AI Agent Systems

Bita Banihashemi[1,*], Yves Lespérance[2] and Sotirios Liaskos[2]

[1]IGDORE, Gothenburg, Sweden
[2]York University, Toronto, Canada

Abstract

Agentic AI systems that use large language models (LLMs) to implement advanced applications are becoming popular. However, such systems are known to be prone to hallucinations, opaque in the way they make decisions, and not offering any safety guarantees. This introduces the need for principled ways to analyze and design such systems, such that their behavior is verifiable against requirements specifications, and compliant and explainable according to predefined behavioral and decision-making rules and strategies. We propose a requirements-driven neuro-symbolic approach to systematically developing such systems. Our approach uses agent-based goal models to represent the system requirements and allowable strategies for achieving them. Goal models are then translated into programs in the IndiGolog logic-based agent programming language to orchestrate agent behavior in compliance with the requirements. LLMs are integrated as components for pursuing specified tasks and rendering formalizable results, amenable to symbolic verification and explanation generation. In this paper, we introduce our approach and examine how it can be applied to the design of a meeting scheduling application.

Keywords

Agents, requirements engineering, goal models, situation calculus, IndiGolog

1. Introduction

Agentic AI systems that are based on large language models (LLMs) are becoming a popular framework for developing advanced applications. These systems make autonomous decisions to achieve the system's objectives often using external tools to retrieve information and perform actions. But such systems are often prone to hallucinations, offer no safety guarantees, are unable to explain their decisions, and are, hence, untrustworthy. It is, furthermore, challenging to specify the requirements for such systems, and to ensure that the requirements are met at run-time.

Goal models are popular in requirements engineering [1], due to their ability to capture and analyze alternative solutions through which human or computational agents can achieve business objectives. As such, these models are a natural framework for designing agentic solutions, in a way that allows precise specification and analysis of their desired properties.

We propose a requirements-driven neuro-symbolic approach to developing AI agent systems. Our approach uses goal models specified in an extension of the iStar 2.0 language [2, 3], called *i+* [4], to represent the requirements for the system and alternative ways of achieving them. Such goal models can be used to analyze behavior and make decisions at design-time. They can also be used by agents to analyze alternatives and make decisions at runtime. Within *i+* models, human and symbolic agents, delegate to neural (LLM) agents specified sub-goals or tasks, and the latter are expected to deliver output that meets a required specification.

Our *i+* models are then translated into programs in the IndiGolog agent programming framework [5, 6]. IndiGolog is a language where one can write high-level programs that specify agent behavior.

Joint Workshop on Statistics and Knowledge Integration for Logic, Learning, Ethical Decisions, and LLMs (SKILLED-LLMs 2026), co-located with FLoC 2026, Lisbon, Portugal
*Corresponding author.
bita.banihashemi@igdore.org (B. Banihashemi); lesperan@eecs.yorku.ca (Y. Lespérance); liaskos@yorku.ca (S. Liaskos)
https://www.researchgate.net/profile/Bita-Banihashemi (B. Banihashemi); https://www.eecs.yorku.ca/~lesperan (Y. Lespérance); https://www.yorku.ca/liaskos/ (S. Liaskos)
0000-0002-7147-484X (B. Banihashemi); 0000-0003-1625-0226 (Y. Lespérance); 0000-0001-5625-5297 (S. Liaskos)

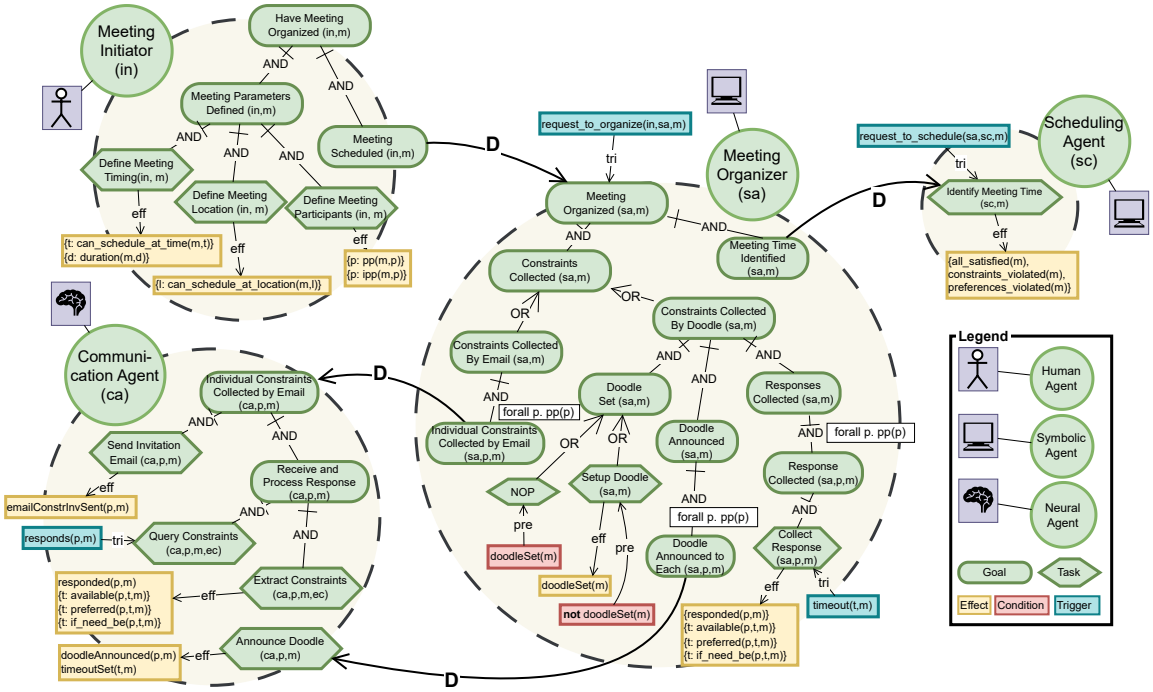

Figure 1: Partial Goal Model for the Meeting Scheduling Problem

Such programs use atomic actions and test conditions that are specified by an action theory in the situation calculus [7, 8], a predicate logic framework for reasoning about action. Programs can be nondeterministic and leave choices to the executing agent. The IndiGolog interpreter uses the program's action theory to reason about possible executions of the program and search for executions that satisfy certain properties.

In our approach, we translate the symbolic agents of the goal models into IndiGolog programs that specify the possible agent behaviors. Delegations to LLMs are treated via specifications of interfaces between the IndiGolog implementation of the symbolic agents and the LLM endpoint, whereby, through appropriately designed prompts, the LLM receives a description of the delegated task and returns a formal output that can be verified against the goal specifications and domain constraints. In this way, rather than improvising their own opaque strategies, LLMs are part of the execution of a symbolically pre-specified agent orchestration, allowing formal analysis of the behavior of the overall system, including checking compliance with safety properties or the generation of explanations [4].

In this paper, we introduce our approach and examine how it can be applied to the design of a meeting scheduling application. We start by describing how goal models can be used to specify the requirements of multi-agent systems in Section 2. We show how LLMs are incorporated in the analysis in Section 3. In Section 4 we sketch how goal models can be translated into IndiGolog specifications. In Section 5 we show how LLMs can be integrated in the model and how the IndiGolog interpreter can allow both simulation of the solution and act as the backbone of the end system. We conclude in Section 6 with our agenda for future work on the topic.

2. Goal Models

Goal models have been found to be an effective tool for identifying and analyzing requirements within a given domain [1, 2, 3, 9, 10]. By identifying the agents relevant to the problem at hand and their high-level goals, analysts can use such models to show how these high-level goals are recursively decomposed into alternative lower level sub-goals that can be delegated to other agents and/or further decomposed into eventually actionable tasks. A specified subset of these tasks constitutes the requirements of the

system-to-be, i.e., functions that a computational agent is supposed to perform. The process ensures that each resulting requirement is justified (i.e., it is connected to someone's goal) and that the set of requirements is complete (i.e., no requirements needed for fulfilling goals are left out). Moreover, through further formalizing goals, tasks, and domain properties, using, e.g., temporal logic [9, 11], we pave the way for automatically testing whether the observed system behaviors satisfy the goal model.

An example of a goal model can be viewed in Figure 1. The model is based on *i+* [4] a recently proposed extension of iStar 2.0 [2], a prominent agent-based goal modeling language; *i+* is the latest of a family of such extensions that have been developed for various analysis purposes [12, 13, 14, 15, 16, 17, 18]. The figure offers a partial goal-oriented analysis of the meeting scheduling problem, which has been used as an exemplar in requirements engineering for many years [19] allowing different methods and formalisms to be compared. The figure represents *actors* (the circular elements – actors are the iStar constructs for agents) involved in the scheduling process and the goals (ovals) each of them has. The *Meeting Initiator* (top left) is the actor that has the top level need to *Have [a] Meeting Organized*. The goal is *AND-refined* into two other subgoals *Meeting Parameters Defined* and *Meeting Scheduled* – both goals need to be satisfied for the parent goal to be considered satisfied. The former subgoal is decomposed into *tasks* which describe specific activities to be performed. The latter goal is delegated to the *Meeting Organizer*, via the iStar's *dependency* construct, meaning that the latter actor is now responsible for achieving it. The different strategies for achieving it are encoded in the underlying decomposition. *OR-refinements*, specifically, (e.g., *Constraints Collected by Email* vs. *Constraints Collected by Doodle*) allow variability in the ways by which goals are achieved [10]. The *Meeting Organizer* delegates its low-level communication tasks to a *Communication Agent* and the subgoal of identifying the meeting time (i.e., finding the optimal time given the collected constraints) to a *Scheduling Agent*. Early diagrams can be agnostic on whether actors are human or computational, and, if the latter, of what specific type (e.g., symbolic or neural); these designations can be made at a later stage as discussed below.

In addition to goals and tasks the diagram contains elements and relationships that add details to the multi-agent design. These include *effect elements*, signifying that the containing predicate lists turn true upon the performance of an associated task, *pre-condition elements*, specifying conditions that must be true for tasks to be performed, *trigger conditions* that cause the agent to attempt a goal or task, goal and task *parameters*, as well as *conditional* refinements (e.g., *Constraints Collected by Email*). Through these extensions translation of the model to formal specifications of the Golog family [5, 6, 20, 21] is possible paving the way for simulation analyses and automated generation of running prototypes.

3. LLMs and Requirements Specification

The meeting scheduling domain that is partially modeled in Figure 1 is suitable for LLM-intensive agent-based solutions. LLM techniques have the potential to improve usability, especially in handling natural language (NL) input/output. For instance, the initiator could provide the meeting request and associated constraints and preferences in NL, as could the potential participants. Explanations for proposed time slots and conflicts could also be provided in NL. LLMs could also generate commonsense constraints and preferences to complement those given explicitly by users. LLMs could even be tasked with finding solutions satisfying the constraints and preferences as best as possible.

From an analysis and design perspective, while LLMs offer such novel capabilities, they do so without transparency and validity guarantees. On the one hand, when asked to make a decision or judgment, it is unknown why they make the decision they make – and whether explanations they render or the chain-of-thought [22] trace that they produce are faithful to the decision-making process they actually followed is an open question [23]. On the other hand, they are known to hallucinate, i.e., provide invalid output that, moreover, they may confidently present as valid. From an analysis standpoint, the question is how we can use the advantages of LLMs in processing complex human language and performing commonsense reasoning, while ensuring that their behavior is at all times compliant with the goal, task, and domain property specifications identified during analysis.

We propose to approach the analysis of agentic systems by viewing LLMs as constituent agents

in neuro-symbolic multi-agent designs, where their role is precisely prescribed during analysis as recipients of goals delegated from other agents (human or symbolic), with precisely specified desired and undesired outcomes. Hence, while operational strategies for the multi-agent system (with all their variability) are specified symbolically (e.g., which actor assumes what goals, how goals are decomposed, in what order they are attempted, what pre-conditions must they satisfy, etc.), execution of tasks requiring human language and decisions based on common sense is delegated to neural agents. The latter, however, render the output in a formal or formalizable manner, so as to allow monitoring and verification with respect to the overall multi-agent design.

In the meeting scheduling example, the naive LLM-centered option is to assign to the LLM agent the overall responsibility to arrange the scheduling of the meeting based on parameters specified in natural language. Then, the LLM is responsible for improvising a plan, perhaps with the assistance of prompt elements, and executing it via contacting participants, receiving and analyzing responses, and making an "intuitive" decision on scheduling time and date. While modern LLMs can perhaps accomplish such complex tasks, both the problem of transparency and explainability (e.g., why this and not that strategy to contact participant X) and the broad problem of safety (e.g., important participant ignored), mentioned above, emerge.

The neuro-symbolic approach we propose, instead, assumes a study of the multi-agent setting, as captured by *i+*, and decisions of what type each agent can be. In Figure 1, the meeting organizer is a symbolic agent operating predictably under the specified decomposition model, and delegating appropriate tasks to other agents, such as extracting formal constraints from emails to a *Communication Agent* which is neural, and deciding the meeting time to a *Scheduling Agent*, which is symbolic (e.g, a constraint solver). These designations are the result of a deliberate late-stage analysis that discriminates symbolic agents which need to follow well-defined strategies for fulfilling goals – hence, explainable, verifiable, etc. – from neural agents which can engage in improvised and creative strategies. While the *i+* model explicates the intentional structure of the former, it is restricted in describing the interface of the latter in the form of pre-conditions and formal output format. This interface description, however, becomes part of the specification of the entire multi-agent design.

With such a model describing neuro-symbolic multi-agent designs, various forms of automated analysis are possible including simulation and generation of running prototypes. This can be achieved via translation of the goal models into IndiGolog, which we discuss next.

4. Translating Goal Models to IndiGolog

IndiGolog [5, 6] is the latest in a family of high-level agent programming languages based on the situation calculus. The original language, Golog [20], allowed complex actions that include conditionals, loops, and nondeterministic choice to be specified and reasoned about. This was later extended to ConGolog [21], which supports concurrent programming. IndiGolog further extends the latter by supporting online execution that interleaves sensing, planning, and execution. Shapiro et al. [24] show how ConGolog specifications of multi-agent systems can be verified. Casciani et al. [25] show how business process models in BPMN can be translated into IndiGolog and analyzed.

The operational semantics of *i+* are defined by a set of translation rules that map *i+* constructs into their corresponding Prolog-based implementations of Golog constructs [4]. Here, we adapt this approach to translate *i+* into IndiGolog, to take advantage of its support for concurrent processes and online sensing. In the translation, every actor is mapped to an object in the Golog domain. Each goal (e.g., *Constraints Collected (sa,m)*) is translated into (a) a satisfaction fluent (*sat_constraintsCollected(sa,m)*) and (b) a procedure name (*constraintsCollected(sa,m)*). Each task in the goal model, such as *Send Invitation Email (sa,p,m)* is translated into (a) a performance fluent (*perf_sendInvitationEmail(sa,p,m)*), and (b) a primitive action (*sendInvitationEmail(sa,p,m)*).

In our example, the agent has incomplete information in the initial state. For instance, the availability and preferred timings of meeting participants are not known initially, and will become available once each participant has provided this information by replying to the invitation email sent or filling up the

Doodle. To accommodate agents that acquire knowledge during executions, we need to consider the agent's *online executions* where it must make decisions based on what it knows, and its knowledge may be updated as it executes the program. In Listing 1, we show part of the result of translating the *i+* model in Figure 1 into IndiGolog, focusing on the meeting organizer agent SA and how it may obtain constraints from participants through email. The sending of emails and processing of the responses are delegated to the communication agent CA, which is assumed to use an LLM to extract the constraints from the email text. CA is implemented as a server in IndiGolog: when a goal is delegated to it, it handles it in a separate thread. We model the agents' knowledge acquisition using two exogenous actions: *responds*(P, M, EC) models the reception of the response email EC from participant P for meeting M and *extractConstraints*$(CA, P, M, EC, LA, LP, LI)$ represents the communication agent extracting lists of available, preferred, and if-need-be time slots (LA, LP, and LI respectively) from the email using an LLM. The effects of these exogenous actions are specified in IndiGolog by declarations *causes_true*(*Action*, *Fluent*, *Condition*), meaning that *Action* causes *Fluent* to become true when *condition* holds.

```
% in SA agent
proc(meetingOrganized(SA,M), % main goal/procedure - AND decomposed into sequence
    [constraintsCollected(SA,M), meetingTimeIdentified(SA,M)]
).

proc(constraintsCollected(SA,M), % OR decomposed
    ndet(constraintsCollectedByEmail(SA,M), constraintsCollectedByDoodle(SA,M))
).

proc(constraintsCollectedByEmail(SA,M), % conditional AND decomposed
  [while(some(p, and(potentialParticipant(p,M),
                     neg(sat_individualConstraintsCollectedByEmail(SA,p,M)))),
     pick(p,[
       ?(and(potentialParticipant(p,M),neg(sat_individualConstraintsCollectedByEmail(SA,p,M)))),
       delegate(SA,ca, individualConstraintsCollectedByEmail(SA,p,M),
           individualConstraintsCollectedByEmail(ca,p,M))])),
   ?(neg(some([p,ec,la,lp,li],and(potentialParticipant(p,M),   % wait until all have responded
            and(neg(responded(p,M,ec)),neg(extractedConstraints(ca,p,M,ec,la,lp,li)))))))]
).
...
% in CA agent
proc(communicationAgentMain(CA),
  iconc(ndet(pi([p,m], [acquireDelegatedindividualConstraintsCollectedByEmail(CA,p,m),
                        individualConstraintsCollectedByEmail(CA,p,m)]),
             pi([p,m], [acquireDelegatedAnnounceDoodle(CA,p,m),
                        announceDoodle(CA,p,m)])
             )
        )).

proc(individualConstraintsCollectedByEmail(CA,P,M), % AND decomposed into sequence
    [sendInvitationEmail(CA,P,M), receiveAndProcessResponse(CA,P,M)]
).

proc(receiveAndProcessResponse(CA,P,M), % AND decomposed into sequence
    [?(some(ec,responded(P,M,ec))),
     pick(ec, [?(responded(P,M,ec)),
                  queryConstraints(CA,P,M,ec),
               ?(extractedConstraints(CA,P,M,ec))])
    ]).
...
% associated declarations
exog_action(responds(P,M,EC)).
```

```
exog_action(extractConstraints(CA,P,M,EC,LA,LP,LI)).
causes_true(responds(P,M,EC), responded(P,M,EC), true).
causes_true(extractConstraints(CA,P,M,EC,LA,LP,LI),extractedConstraints(CA,P,M,EC), true).
causes_true(extractConstraints(CA,P,M,EC,LA,LP,LI),available(P,T,M), member(T,LA)).
causes_true(extractConstraints(CA,P,M,EC,LA,LP,LI),preferred(P,T,M), member(T,LP)).
causes_true(extractConstraints(CA,P,M,EC,LA,LP,LI),ifNeedBe(P,T,M), member(T,LI)).
```

Listing 1: Partial Translation of Goal Model in Fig. 1 into IndiGolog.

5. Simulation and Analysis of Agent Behaviors

The formalized models are useful for simulating the system-to-be in order to support the process of discovering and specifying requirements, analyzing system behavior, verifying system properties, and generating explanations. They can also be used to develop a system prototype that uses IndiGolog to implement the high-level behavior of the agents involved and their interaction. The implemented agents can be interfaced to application software, LLMs, and symbolic solvers to support their functionalities. For this, we use a SWI Prolog-based implementation of IndiGolog[1] which readily supports the implementation of individual agents as well as multi-agent systems. Agents can be interfaced with their environment through an environment manager.

The IndiGolog interpreter supports the generation of partial or complete execution traces of the agent programs. The system's initial situation as well as the exogenous actions that the system responds to can be controlled. The interpreter can be used to search for executions that falsify a state formula representing a safety property. For example, we may want to check that the system never tries to identify a meeting time unless it has collected constraints from all important participants. It can also be used to check for the existence of an execution that achieves some goal condition under some assumptions. In related work [25], IndiGolog has been used to formally specify business process models and support their analysis. Because the entire system is formally specified in the situation calculus, verification techniques based on model checking and SMT can also be exploited. De Giacomo et al. [26] discuss how reactive system synthesis techniques can be applied to Golog specifications.

6. Conclusion and Future Work

In this paper, we investigated using an *i+* goal model for both specification of the requirements of a neuro-symbolic agentic AI system, as well as facilitating analysis of agent behavior and decision making at both design-time and runtime. These goal models are translated into programs in the IndiGolog agent programming language to orchestrate agent behavior in compliance with the requirements. We focused on a meeting scheduling application that includes both LLM-based agents and symbolic agents. Simulation is based on generating possible action histories and asking questions based on them.

Integration of LLM techniques in symbolic AI agent systems raises many open questions. For instance, are LLMs sufficiently accurate in their interpretation of the input and in their recommendations? What prompt engineering methods work best under which conditions? Is it better to get users to perform quality checks on LLM interpretations or not, and if so, under what conditions? How do we manage the trade-off between quality and user interaction? And more generally, how can one combine LLM methods and symbolic solvers to optimally satisfy the system's goals? We aim to explore these open questions in future research.

Declaration on Generative AI

The author(s) have not employed any Generative AI tools.

[1]The IndiGolog implementation we use is available at: https://github.com/ai-krml-uoft/indigolog/.

References

[1] J. Horkoff, F. B. Aydemir, E. Cardoso, T. Li, A. Maté, E. Paja, M. Salnitri, L. Piras, J. Mylopoulos, P. Giorgini, Goal-oriented requirements engineering: an extended systematic mapping study, Requirements Engineering 24 (2019) 133–160.

[2] F. Dalpiaz, X. Franch, J. Horkoff, iStar 2.0 language guide, CoRR abs/1605.07767 (2016).

[3] E. S. K. Yu, J. Mylopoulos, Understanding "why" in software process modelling, analysis, and design, in: Proceedings of 16th International Conference on Software Engineering ICSE 94, IEEE Computer Society / ACM Press, 1994, pp. 159–168.

[4] S. Liaskos, J. Mylopoulos, A. Borgida, S. M. Khan, Modeling and reasoning about explanation requirements using goal models, in: Proceedings of the 43rd International Conference on Conceptual Modeling (ER 2024), Lecture Notes in Computer Science, Springer, 2024, pp. 215–234. doi:https://doi.org/10.1007/978-3-031-75872-0_12.

[5] G. De Giacomo, H. J. Levesque, An incremental interpreter for high-level programs with sensing, in: H. J. Levesque, F. Pirri (Eds.), Logical Foundations for Cognitive Agents: Contributions in Honor of Ray Reiter, Springer, Berlin, 1999, pp. 86–102.

[6] G. De Giacomo, Y. Lespérance, H. J. Levesque, S. Sardiña, IndiGolog: A high-level programming language for embedded reasoning agents, in: Multi-Agent Programming, Languages, Tools and Applications, Springer, 2009, pp. 31–72.

[7] J. McCarthy, P. J. Hayes, Some philosophical problems from the standpoint of artificial intelligence, Machine Intelligence 4 (1969) 463–502.

[8] R. Reiter, Knowledge in Action. Logical Foundations for Specifying and Implementing Dynamical Systems, The MIT Press, 2001.

[9] A. Dardenne, A. van Lamsweerde, S. Fickas, Goal-directed requirements acquisition, Science of Computer Programming 20 (1993) 3–50. URL: https://www.sciencedirect.com/science/article/pii/016764239390021G. doi:10.1016/0167-6423(93)90021-G.

[10] S. Liaskos, A. Lapouchnian, Y. Yu, E. Yu, J. Mylopoulos, On goal-based variability acquisition and analysis, in: Proceedings of the 14th IEEE International Requirements Engineering Conference (RE'06), 2006, pp. 79–88. doi:10.1109/RE.2006.45.

[11] A. Fuxman, L. Liu, J. Mylopoulos, M. Pistore, M. Roveri, P. Traverso, Specifying and analyzing early requirements in Tropos, Requirements Engineering 9 (2004) 132–150. URL: https://doi.org/10.1007/s00766-004-0191-7. doi:10.1007/s00766-004-0191-7.

[12] A. Lapouchnian, Y. Lespérance, Using the ConGolog and CASL Formal Agent Specification Languages for the Analysis, Verification, and Simulation of i* Models, in: A. T. Borgida, V. K. Chaudhri, P. Giorgini, E. S. Yu (Eds.), Conceptual Modeling: Foundations and Applications, volume 5600, Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, pp. 483–503. URL: http://link.springer.com/10.1007/978-3-642-02463-4_25. doi:10.1007/978-3-642-02463-4_25, series Title: Lecture Notes in Computer Science.

[13] S. Liaskos, S. A. McIlraith, J. Mylopoulos, Towards Augmenting Requirements Models with Preferences, in: Proceedings of the 24th IEEE/ACM International Conference on Automated Software Engineering (ASE'09), 2009, pp. 565–569. doi:10.1109/ASE.2009.91.

[14] S. Liaskos, S. McIlraith, S. Sohrabi, J. Mylopoulos, Representing and reasoning about preferences in requirements engineering, Requirements Engineering Journal (REJ) 16 (2011) 227–249. doi:https://doi.org/10.1007/s00766-011-0129-9.

[15] S. Liaskos, S. M. Khan, M. Soutchanski, J. Mylopoulos, Modeling and reasoning with decision-theoretic goals, in: W. Ng, V. C. Storey, J. C. Trujillo (Eds.), Proceedings of the 32th International Conference on Conceptual Modeling (ER'13), Springer, Berlin, Heidelberg, 2013, pp. 19–32. doi:doi.org/10.1007/978-3-642-41924-9_3.

[16] S. Liaskos, S. M. Khan, J. Mylopoulos, Modeling and reasoning about uncertainty in goal models: a decision-theoretic approach, Software & Systems Modeling 21 (2022) 1–24. doi:https://doi.org/10.1007/s10270-021-00968-w.

[17] S. Liaskos, N. Dang, S. M. Khan, J. Mylopoulos, R. Golipour, A. Radjou, Designing and generating

reinforcement learning training simulators using goal models, Data & Knowledge Engineering 164 (2026) 102603. URL: https://linkinghub.elsevier.com/retrieve/pii/S0169023X26000509. doi:10.1016/j.datak.2026.102603.

[18] S. Liaskos, S. M. Khan, J. Mylopoulos, R. Golipour, Model-Driven Design and Generation of Training Simulators for Reinforcement Learning, in: W. Maass, H. Han, H. Yasar, N. J. Multari (Eds.), Proceedings of the 43rd International Conference on Conceptual Modeling, (ER 2024), volume 15238 of *Lecture Notes in Computer Science*, Springer, Pittsburgh, Pennsylvania, 2024, pp. 170–191. URL: https://doi.org/10.1007/978-3-031-75872-0_10. doi:10.1007/978-3-031-75872-0_10.

[19] M. S. Feather, S. Fickas, A. Finkelstein, A. van Lamsweerde, Requirements and specification exemplars, Automated Software Engineering 4 (1997) 419–438.

[20] H. J. Levesque, R. Reiter, Y. Lespérance, F. Lin, R. B. Scherl, GOLOG: A logic programming language for dynamic domains, The Journal of Logic Programming 31 (1997) 59–83.

[21] G. De Giacomo, Y. Lespérance, H. J. Levesque, ConGolog, a concurrent programming language based on the situation calculus, Artificial Intelligence 121 (2000) 109–169.

[22] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. H. Chi, Q. V. Le, D. Zhou, Chain-of-thought prompting elicits reasoning in large language models, in: S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, A. Oh (Eds.), Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022, 2022. URL: http://papers.nips.cc/paper_files/paper/2022/hash/9d5609613524ecf4f15af0f7b31abca4-Abstract-Conference.html.

[23] D. Ganguli, A. Askell, N. Schiefer, T. I. Liao, K. Lukosiute, A. Chen, A. Goldie, A. Mirhoseini, C. Olsson, D. Hernandez, D. Drain, D. Li, E. Tran-Johnson, E. Perez, J. Kernion, J. Kerr, J. Mueller, J. Landau, K. Ndousse, K. Nguyen, L. Lovitt, M. Sellitto, N. Elhage, N. Mercado, N. DasSarma, O. Rausch, R. Lasenby, R. Larson, S. Ringer, S. Kundu, S. Kadavath, S. Johnston, S. Kravec, S. E. Showk, T. Lanham, T. Telleen-Lawton, T. Henighan, T. Hume, Y. Bai, Z. Hatfield-Dodds, B. Mann, D. Amodei, N. Joseph, S. McCandlish, T. Brown, C. Olah, J. Clark, S. R. Bowman, J. Kaplan, The capacity for moral self-correction in large language models, CoRR abs/2302.07459 (2023). URL: https://doi.org/10.48550/arXiv.2302.07459. doi:10.48550/ARXIV.2302.07459. arXiv:2302.07459.

[24] S. Shapiro, Y. Lespérance, H. J. Levesque, The cognitive agents specification language and verification environment for multiagent systems, in: Proceedings of the first international joint conference on Autonomous agents and multiagent systems (AAMAS'02), ACM, 2002, pp. 19–26. doi:https://doi.org/10.1145/544741.544746.

[25] A. Casciani, S. Agostinelli, Y. Lespérance, A. Marrella, S. Sardiña, Formal semantics for knowledge representation and automated reasoning in BPMN process models, Information Systems 140 (2026) 102718.

[26] G. De Giacomo, Y. Lespérance, M. Mancanelli, G. Parretti, Reactive synthesis for Golog specifications in the propositional situation calculus, in: Proceedings of the 23rd International Conference on Principles of Knowledge Representation and Reasoning (KR 2026), 2026, p. 11 pages.