Proceedings of the
2nd Workshop on Artificial Intelligence and Formal Verification, Logics, Automata and Synthesis (OVERLAY),
September 25, 2020

A Fixed-point Model-checker for BDI Logics over Finite-state Worlds*

Salvatore La Torre[1] and Gennaro Parlato[2]

[1]University of Salerno, Italy
[2]University of Molise, Italy
[1]slatorre@unisa.it
[2]gennaro.parlato@unimol.it

Abstract

BDI agents are among the most widely studied models of rational agents. In this architecture, systems are seen as rational agents with certain mental attitudes such as *belief*, *desire*, and *intention*. In this paper, we consider the model-checking problem for CTL_{BDI}, the branching-time logic CTL augmented with the BDI modalities, over finite-state structures, and in particular, a proof-of-concept tool that is based on a translation to a formula in a fixed-point logic. We give a description of this tool in some details and also discuss some preliminary evaluations.

1 Introduction

BDI agents, i.e., rational agents having certain mental attitudes of *belief*, *desire*, and *intention*, are real-world system models that have been heavily investigated in the literature [11]. Their use in the design and implementation of safety-critical applications has also motivated the study and the development of tools for correctness verification (see [3, 4, 1]).

A common specification language that is used to express the specification for BDI agents is $\text{CTL}^*_{\text{BDI}}$. $\text{CTL}^*_{\text{BDI}}$is obtained by extending CTL^* [6] with the belief, desire, and intention modal operators. In a recent research [9, 10], the model-checking problem for $\text{CTL}^*_{\text{BDI}}$ (and its fragment CTL_{BDI}) over *finite-state* structures is shown to be PSPACE-complete. This logic is a semantic restriction of the possible-worlds semantics given by Rao and Georgeff [12, 13] where the worlds are modeled as Kripke structures and the BDI relations are captured by finite-state automata. In [10] is also given a fixed-point algorithm to decide the model-checking question. Further, for the logic CTL_{BDI}, the decision algorithm is formulated in a fixed-point calculus that can be directly translated to the input language of BDD-based model checker MUCKE [2], thus yielding a proof-of-concept tool to decide CTL_{BDI}.

In this paper, we report on the current status of our tool. In particular, after describing the overall architecture, we recall the decision algorithm and then by a case study we describe our encoding in the fixed-point calculus. We also synthesize a benchmark which is based on the given case study and can be varied in size such that it could be used to perform a scalability analysis of our approach. However, here we only report on some preliminary experiments where we use small values of the parameters. The reason is that the tool is not yet fully automatized and thus it would not be feasible to handle large input models. As future work, we plan to improve this by also integrating the translation with modules within the GETAFIX framework [7, 8] that parses models described as Boolean programs.

*This work was partially supported by GNCS 2020 and FARB-UNISA 2018-2019 grants.

$\Lambda(\sigma,\psi,u) := \Lambda^\sigma_{at} \vee \Lambda^\sigma_\neg \vee \Lambda^\sigma_\vee \vee \Lambda^\sigma_{\forall\bigcirc} \vee \Lambda^\sigma_{\exists\bigcirc} \vee \Lambda^\sigma_{\forall\mathcal{U}} \vee \Lambda^\sigma_{\exists\mathcal{U}} \vee \Lambda^\sigma_{\mathcal{B}} \vee \Lambda^\sigma_{\mathcal{D}} \vee \Lambda^\sigma_{\mathcal{I}}$ where:

1. $\Lambda^0_{at} := \texttt{Atomic}(\psi) \wedge \neg\texttt{Label}(\psi,u)$
2. $\Lambda^1_{at} := \texttt{Atomic}(\psi) \wedge \texttt{Label}(\psi,u)$
3. $\Lambda^\sigma_\neg := \texttt{Neg}(\psi) \wedge \exists\psi'.(\,\texttt{Ready}(\psi',u) \wedge \texttt{Sub}(\psi',\psi) \wedge \Lambda(1-\sigma,\psi',u)\,)$
4. $\Lambda^\sigma_\vee := \texttt{Or}(\psi)$
   $\wedge\exists\psi',\psi''.(\texttt{Ready}(\psi')\wedge\texttt{Sub}(\psi',\psi)\wedge\texttt{Ready}(\psi'')\wedge\texttt{Sub}(\psi'',\psi)\wedge(\Lambda(\sigma,\psi',u)\,o_\sigma\Lambda(\sigma,\psi'',u))\,)$,
   where o_σ is $\vee$ if $\sigma=1$ and $\wedge$ otherwise
5. $\Lambda^0_{\exists\bigcirc} := \texttt{Existential}(\psi) \wedge \texttt{Next}(\psi)$
   $\wedge\,\exists\psi'.\,(\,\texttt{Ready}(\psi') \wedge \texttt{Sub}(\psi',\psi) \wedge \forall v.\,(\,\neg\texttt{Succ}_T(v,u) \vee \Lambda(0,\psi',v)\,)\,)$
6. $\Lambda^1_{\exists\bigcirc} := \texttt{Existential}(\psi) \wedge \texttt{Next}(\psi)$
   $\wedge\,\exists\psi'.\,(\,\texttt{Ready}(\psi') \wedge \texttt{Sub}(\psi',\psi) \wedge \exists v.\,(\,\texttt{Succ}_T(v,u) \wedge \Lambda(1,\psi',v)\,)\,)$
7. $\Lambda^0_{\forall\bigcirc} := \texttt{Universal}(\psi) \wedge \texttt{Next}(\psi)$
   $\wedge\,\exists\psi'.\,(\,\texttt{Ready}(\psi') \wedge \texttt{Sub}(\psi',\psi) \wedge \exists v.\,(\,\texttt{Succ}_T(v,u) \wedge \Lambda(0,\psi',v)\,)\,)$
8. $\Lambda^1_{\forall\bigcirc} := \texttt{Universal}(\psi) \wedge \texttt{Next}(\psi)$
   $\wedge\,\exists\psi'.\,(\,\texttt{Ready}(\psi') \wedge \texttt{Sub}(\psi',\psi) \wedge \forall v.\,(\,\neg\texttt{Succ}_T(v,u) \vee \Lambda(1,\psi',v)\,)\,)$
9. $\Lambda^0_{\exists\mathcal{U}} := \texttt{Existential}(\psi) \wedge \texttt{Until}(\psi)$
   $\wedge\,\exists\psi',\psi''.\,(\,\texttt{Ready}(\psi') \wedge \texttt{Ready}(\psi'') \wedge \texttt{Sub}(\psi',\psi'',\psi) \wedge \Lambda(0,\psi'',u)$
   $\wedge\,(\,\Lambda(0,\psi',u) \vee \forall v.\,(\,\neg\texttt{Succ}_T(v,u) \vee \Lambda(0,\psi,v)\,)\;\;)\;\;)$
10. $\Lambda^1_{\exists\mathcal{U}} := \texttt{Existential}(\psi) \wedge \texttt{Until}(\psi)$
    $\wedge\,\exists\psi',\psi''.\,(\,\texttt{Ready}(\psi') \wedge \texttt{Ready}(\psi'') \wedge \texttt{Sub}(\psi',\psi'',\psi)$
    $\wedge\,(\,\Lambda(1,\psi'',u) \vee (\,\Lambda(1,\psi',u) \wedge \exists v.\,(\,\texttt{Succ}_T(v,u) \wedge \Lambda(1,\psi,v)\,)\;\;)\;\;)\;\;)$
11. $\Lambda^0_{\forall\mathcal{U}} := \texttt{Universal}(\psi) \wedge \texttt{Until}(\psi)$
    $\wedge\,\exists\psi',\psi''.\,(\,\texttt{Ready}(\psi') \wedge \texttt{Ready}(\psi'') \wedge \texttt{Sub}(\psi',\psi'',\psi) \wedge \Lambda(0,\psi'',u)$
    $\wedge\,(\,\Lambda(0,\psi',u) \vee \exists v.\,(\,\texttt{Succ}_T(v,u) \wedge \Lambda(0,\psi,v)\,)\;\;)\;\;)$
12. $\Lambda^1_{\forall\mathcal{U}} := \texttt{Universal}(\psi) \wedge \texttt{Until}(\psi)$
    $\wedge\,\exists\psi',\psi''.\,(\,\texttt{Ready}(\psi') \wedge \texttt{Ready}(\psi'') \wedge \texttt{Sub}(\psi',\psi'',\psi)$
    $\wedge\,(\,\Lambda(1,\psi'',u) \vee (\,\Lambda(1,\psi',u) \wedge \forall v.\,(\,\texttt{Succ}_T(v,u) \wedge \Lambda(1,\psi,v)\,)\;\;)\;\;)\;\;)$
13. $\Lambda^0_{\mathcal{B}} := \textsc{bel}(\psi) \wedge \exists\psi'.\,(\,\texttt{Ready}(\psi') \wedge \texttt{Sub}(\psi',\psi) \wedge \exists v.\,(\,\texttt{Succ}_{\mathcal{B}}(v,u) \wedge \Lambda(0,\psi',v)\,)\,)$
    (similarly for desire and intention formulas)
14. $\Lambda^1_{\mathcal{B}} := \textsc{bel}(\psi) \wedge \exists\psi'.\,(\,\texttt{Ready}(\psi') \wedge \texttt{Sub}(\psi',\psi) \wedge \forall v.(\,\neg\texttt{Succ}_{\mathcal{B}}(v,u) \vee \Lambda(1,\psi',v)\,)\,)$
    (similarly for desire and intention formulas)

Figure 1: Formal definition of the relation capturing the fulfillment of subformulas at the nodes of $\mathcal{G}_\mathcal{M}$.

2 Branching-time BDI logic over finite-state worlds

In this section, we briefly and informally recall the logic CTL$_{\text{BDI}}$ over finite-state worlds and the main known results. For the details we refer the reader to [10].

Syntax. CTL$_{\text{BDI}}$ formulas are inductively defined starting from atomic propositions by applying the logical connectives, the path quantifiers coupled with a temporal operator such as *next* ($\bigcirc$) and *until* ($\mathcal{U}$), and the *belief* (BEL), *desire* (DES), and *intention* (INT) operators.

Semantics. The meaning of the formulas from CTL$_{\text{BDI}}$ is defined according to a possible world semantics

where each possible world is not an instantaneous state but a transition system. All the worlds are synchronized over a shared branching-time structure whose time points (nodes) represent the instantaneous states. The meaning of the belief-desire-intention (BDI) operators is then given through accessibility relations that relate the possible worlds at each time point and thus can possibly vary over time, while the meaning of temporal operators is related to the temporal accessibility relation defined by the the branching-time structure.

A *finite-state structure* constrains a general CTL_{BDI} structure such that:

- the shared branching-time structure is a full k-ary tree for an integer $k > 0$ (i.e., an infinite tree where each node has exactly k children),
- each of the worlds is defined by a Kripke structure whose tree unrolling from its unique initial state is contained into the shared branching time structure (i.e., each transition is mapped to a child from 1 through k and from each state there is at most an outgoing transition corresponding to a given child i),
- each BDI relation is defined by a finite automaton that takes as input sequences over the alphabet $\{1, \ldots, k\}$, i.e., an automaton that maps paths of the tree-structure to the worlds modeling the BDI attitude.

For a structure $\mathcal{M}$, a world w and a CTL_{BDI} formula φ we denote that φ is fulfilled on $\mathcal{M}$ starting from w as $\mathcal{M}, w \models \varphi$.

Model-checking. The CTL_{BDI} model-checking problem over finite-state structures asks whether $\mathcal{M}, w \models \varphi$ holds for given finite-state structure $\mathcal{M}$, world w and CTL_{BDI} formula φ. In [10], we show that this problem is PSPACE-complete and give a fixed-point algorithm to decide it. This algorithm relies on the construction of a finite graph $\mathcal{G}_{\mathcal{M}}$ that captures the semantics of $\mathcal{M}$ as the cross product of its Kripke structures and automata synchronized over the sequences from alphabet $\{1, \ldots, k\}$. The graph has different kinds of edges depending on whether they capture the successor in a Kripke structure or $\mathcal{I}$ accessibility. Then, starting from the atomic propositions, the algorithm labels the nodes of $\mathcal{G}_{\mathcal{M}}$ similarly to the standard labeling algorithm to decide CTL model-checking (see [5]).

Figure 1 gives the relation that captures the labeling of $\mathcal{G}_{\mathcal{M}}$ nodes with the fulfilled sub-formulas. This definition uses predicates to denote the successors in the graph $\mathcal{G}_{\mathcal{M}}$, to relate formulas to sub-formulas, and to denote whether a sub-formula is an atomic proposition, the negation/disjunction of formulas, universally/existentially quantified, a next/until/belief/desire/intention formula. Additionally, $\texttt{Ready}(\psi, u)$ denotes the formula $\exists\sigma.\Lambda(\sigma, \psi, u)$.

In particular, the following theorem holds:

Theorem 1. *[10] Given a CTL_{BDI} formula φ, a finite-state structure $\mathcal{M}$ and a world w,*

$$\mathcal{M}, w \models \varphi \text{ iff } \Lambda(1, \varphi, u) \text{ holds true}$$

where u is the initial state of $\mathcal{G}_{\mathcal{M}}$ corresponding to w.

3 Proof-of-concept tool

In this section we give an overview of a prototype tool called BDI-checker for solving the model-checking problem of CTL_{BDI}. The underling search space engine uses Binary Decision Diagrams to implement the algorithm's operations as well as to represent the set of reachable states during the analysis. To illustrate our translation we consider a case study taken from [10]. We finally elaborate a benchmark starting from this case study to evaluate the tool.

Case study. We consider a simple scenario where a robot can essentially perform two tasks: getting a beer from the refrigerator and opening the door. The only uncertainties in the environment are the presence or not of a beer can in the refrigerator and of a person at the door house. We can model these as beliefs and thus have four different Kripke structures one for each of the possible beliefs.

In the beginning, all the beliefs are possible and thus all the worlds are belief-accessible. As soon as the robot realizes that no beer is in the refrigerator only the two worlds matching this belief become accessible. Also, if the doorbell rings, the robot changes its beliefs about the presence of a person at the

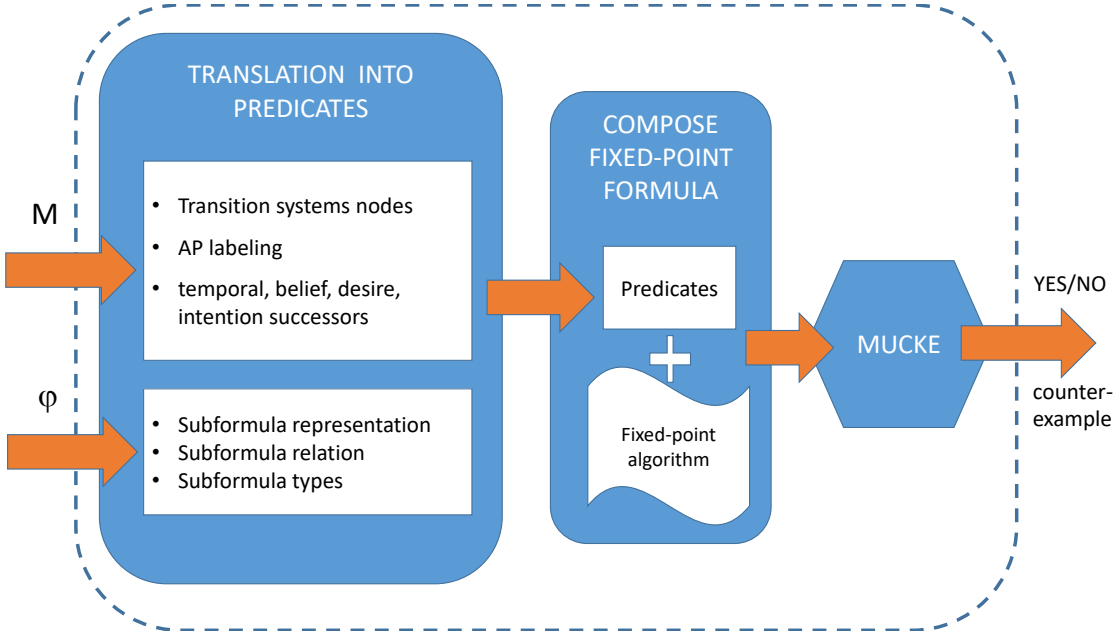

Figure 2: Tool architecture.

house door. After opening the door, the robot becomes again agnostic on whether there is a person at the door. After sensing that no beer is in the refrigerator instead its beliefs about the content of the refrigerator will not change forever (this might be changed by adding a further event that a delivery man brings some beer cans).

Robot desires and intentions can be modeled by adding more worlds "refining" the belief related worlds. To keep the description simple, in this version of the paper we omit a detailed discussion of such aspects.

We consider two CTL$_{\text{BDI}}$ formulas: $\varphi_1 = \forall\Box(\,\text{BEL}\,br \;\rightarrow\; \exists\Diamond\,\texttt{bb})$, i.e., "whenever the robot believes that a beer can is in the refrigerator, she can possibly bring it back", and $\varphi_2 = \forall\Box(\,\text{BEL}\,br \;\rightarrow\; \exists\Box\,\texttt{bb}\,)$, i.e., "whenever the robot believes that a beer can is in the refrigerator, she can always bring it back". The first one is fulfilled on our structure, while the second one is not.

Architecture. The tool architecture is shown in Figure 2. BDI-checker takes as input a finite-state structure M and a CTL$_{\text{BDI}}$ formula φ, and returns as result "YES", if the model M meets the specification φ, and a counter-example witnessing that M does not satisfy φ, otherwise.

The first internal module of BDI-checker, called TRANSLATION INTO PREDICATES, takes as input M and φ and transforms them into a series of definitions of Boolean predicates. In particular, we define the predicates used in the definition of formula Λ given in Figure 1: we encode the nodes of graph $\mathcal{G}_{\mathcal{M}}$ from the Kripke structures and the automata of $\mathcal{M}$, and capture the successor relations using predicates; and from φ we encode the subformulas used in the labeling algorithm and capture their syntactic type again by using predicates.

For our case study, we encode the nodes of the Kripke structures and the automata of M as integers, and thus the nodes of $\mathcal{G}_M$ as tuples of integers. The generated predicates use this encoding. As an example, the predicate encoding the labeling of $\mathcal{G}_M$ vertices with formulas is defined as follows:

```
bool Label(Formula f, GVertex u) (
       (u.w=0 & Lab1(f,u.s0) )
     | (u.w=1 & Lab2(f,u.s1) )
     | (u.w=2 & Lab3(f,u.s2) )
     | (u.w=3 & Lab4(f,u.s3) )
)
```

where `Lab1`, `Lab2`, `Lab3`, and `Lab4` are the predicates capturing the labeling of the four Kripke structures

of M, w is the u component denoting the current Kripke structure (i.e, the current world), and `s0`, `s1`, `s2`, and `s3` denote the u components corresponding to the states of the respective Kripke structures.

We then encode the seven subformulas of φ_1 as integers in the interval $[0, 6]$ (according to the syntax tree of φ_1 starting with 0 for φ_1 itself, and then proceeding top-down and left-to-right). With such an encoding, we define the predicates used in Figure 1. For example, we have the following definitions:

```
bool     Atomic( Formula f )   ( f=5 | f=6 )
bool  Universal( Formula f )   (    f=0    )
bool      Until( Formula f )   (   false   )
```

Once, all the predicates are defined, the original model-checking question, that is, determining whether $\mathcal{M}, w \models \varphi$, can be translated in the following clause:

```
( exists GVertex u.  Lambda(true,0,u) ),
```

which, in our case study, simply asks the backend solver to verify whether a vertex of $\mathcal{G}_M$ exists where the input formula φ_1 holds true. Thus, in the second module, called COMPOSE FIXED-POINT FORMULA, we just put together the predicates computed by the first module with the fixed-point algorithm given in Figure 1 and the above clause.

The last module just invokes the backend solver MUCKE on the built formula and returns its outcome.

Evaluation. To evaluate our prototype tool we have elaborated a benchmark by extending the given case study with the layout of the environment. A layout is given as a bi-dimensional grid where the refrigerator, the door, the lounge (the base position of the robot) and some obstacles are positioned. This benchmark is parameterized on the size of the grid, the number of instances and the positioning of the objects (refrigerator, door, lounge, and obstacles). This will allow us to evaluate the scalability of the approach by varying the parameters. However, to date, we have only exercised our tool on a simple scenario (2×2 grid) against the two different CTL_{BDI} formulas given above. The tool has performed quite well: it gives the right answer in a fraction of a second with BBD maximum size of 1134. We were unable to perform a systematic evaluation since most of the translations in our tool are still performed manually which prevents us to manage large benchmarks. As future work, we plan to complete the automatization of the modules of our tool and then perform a full evaluation over the described benchmark.

References

[1] M. Benerecetti, F. Giunchiglia, and L. Serafini. Model checking multiagent systems. *J. Log. Comput.*, 8(3):401–423, 1998.

[2] A. Biere. μcke - efficient μ-calculus model checking. In O. Grumberg, editor, *Computer Aided Verification, 9th International Conference, CAV '97, Haifa, Israel, June 22-25, 1997, Proceedings*, volume 1254 of *Lecture Notes in Computer Science*, pages 468–471. Springer, 1997.

[3] J. Blee, D. Billington, G. Governatori, and A. Sattar. Levels of modality for BDI logic. *J. Applied Logic*, 9(4):250–273, 2011.

[4] R. H. Bordini, M. Fisher, W. Visser, and M. J. Wooldridge. Verifying multi-agent programs by model checking. *Autonomous Agents and Multi-Agent Systems*, 12(2):239–256, 2006.

[5] E. A. Emerson. Temporal and modal logic. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*, pages 995–1072. The MIT Press/Elsevier, 1990.

[6] E. A. Emerson and E. M. Clarke. Using branching time temporal logic to synthesize synchronization skeletons. *Sci. Comput. Program.*, 2(3):241–266, 1982.

[7] S. La Torre, P. Madhusudan, and G. Parlato. Analyzing recursive programs using a fixed-point calculus. In M. Hind and A. Diwan, editors, *Proceedings of the 2009 ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2009, Dublin, Ireland, June 15-21, 2009*, pages 211–222. ACM, 2009.

[8] S. La Torre, P. Madhusudan, and G. Parlato. Model-checking parameterized concurrent programs using linear interfaces. In T. Touili, B. Cook, and P. B. Jackson, editors, *Computer Aided Verification, 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. Proceedings*, volume 6174 of *Lecture Notes in Computer Science*, pages 629–644. Springer, 2010.

[9] S. La Torre and G. Parlato. Model checking bdi logics over finite-state worlds. In *Proceedings of the 1st Workshop on Artificial Intelligence and Formal Verification, Logic, Automata, and Synthesis co-located with the 18th International Conference of the Italian Association for Artificial Intelligence (AI*IA 2019), Rende (CS), Italy, November 19-20, 2019*, pages 11–16, 2019.

[10] S. La Torre and G. Parlato. On the model-checking of branching-time temporal logic with BDI modalities. In A. E. F. Seghrouchni, G. Sukthankar, B. An, and N. Yorke-Smith, editors, *Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems, AAMAS '20, Auckland, New Zealand, May 9-13, 2020*, pages 681–689. International Foundation for Autonomous Agents and Multiagent Systems, 2020.

[11] J.-J. C. Meyer, J. Broersen, and A. Herzig. Bdi logics. In *Handbook of Epistemic Logic*, pages 453–498. College Publications, 2015.

[12] A. S. Rao and M. P. Georgeff. Modeling rational agents within a bdi-architecture. In J. F. Allen, R. Fikes, and E. Sandewall, editors, *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning (KR'91). Cambridge, MA, USA, April 22-25, 1991.*, pages 473–484. Morgan Kaufmann, 1991.

[13] A. S. Rao and M. P. Georgeff. A model-theoretic approach to the verification of situated reasoning systems. In R. Bajcsy, editor, *Proceedings of the 13th International Joint Conference on Artificial Intelligence. Chambéry, France, August 28 - September 3, 1993*, pages 318–324. Morgan Kaufmann, 1993.