Rafael H. Bordini, Louise A. Dennis and Yves Lespérance (eds)
Pre-proceedings of the 7th International Workshop on Engineering Multi-Agent Systems (EMAS 2019)
13th-14th May 2019, Montreal, Canada
cgi.csc.liv.ac.uk/~lad/emas2019/accepted/EMAS2019_paper_18.pdf

SAT for Epistemic Logic using Belief Bases

Fabián Romero[1] and Emiliano Lorini[1]

IRIT, Toulouse France

Abstract. In [1] a new epistemic logic LDA of explicit and implicit beliefs was introduced, and in [2] we presented a tableau-based satisfability checking procedure as well as a dynamic extension for LDA. Based on such procedure, we created a portable software implementation that works for the family of multi-agent epistemic logics, as well as for the proposed dynamic extension. This software implementation runs as a library for the most common operative systems, also runs in popular IoT and robot hardware, as well as cloud environments and in server-less configurations.

1 Introduction

We believe that semantics based on explicit representation of agents' epistemic states expressed as knowledge or belief bases, are a more natural paradigm for the description of intelligent systems such as robotic and conversational agents than the Kripkean semantics commonly used for epistemic logics [4]. We implemented the tableau-based satisfability procedure for LDA given in [2] in order to have a tool to experiment with such semantics and explore its use.

2 Syntax and semantics

The language $\mathcal{L}_{\mathsf{LDA}}$ is constructed in the following way. Assume a countably infinite set of atomic propositions $Atm = \{p, q, \ldots\}$ and a finite set of agents $Agt = \{1, \ldots, n\}$.

The language $\mathcal{L}_0$, is the language of explicit beliefs defined by the grammar:

$$\alpha ::= \gamma \mid \triangle_i \alpha$$

Where γ is the grammar of classical propositional logic. The multi-modal operator $\triangle_i \alpha$ is read as "α is a formula on agent's i belief base". The language of implicit beliefs $\mathcal{L}_{\mathsf{LDA}}$ is defined by the grammar:

$$\phi ::= \alpha \mid \Box_i \phi \mid \Diamond_i \phi$$

Where $\alpha \in \mathcal{L}_0$ And the $\Box_i \phi$ modality can be read as "agent i can deduce ϕ from its belief base" and the modal dual $\Diamond_i \phi$ as "ϕ is consistent with agent i belief base".

The dynamic extension of LDA we present in our companion paper, allow us to describe actions of agents under observability conditions, this perceptive context, where the dynamic actions take place is defined by the following grammar $\mathcal{L}_{\mathsf{OBS}}$:

$$\omega ::= \mathsf{see}_{i,j} \mid \mathsf{see}_i\omega$$

The expression $\mathsf{see}_{i,j}$ Can be read as "agent i sees what agent j does". And $\mathsf{see}_i\omega$ represents the fact that "agent i sees that ω".

The language $\mathcal{L}_{\mathsf{DLDA}}$ is defined by the following grammar:

$$\chi ::= \neg\chi \mid \chi_1 \wedge \chi_2 \mid [(p, \tau, i, \Omega)]\chi \mid \phi$$

Where p is a proposition, $i \in Agt$, ϕ ranges over the language $\mathcal{L}_{\mathsf{LDA}}$,τ ranges over $\{+, -\}$ and Ω is a finite set of formulas of $\mathcal{L}_{\mathsf{OBS}}$.

The action $+p$ consists in setting the value of the atomic variable p to true, whereas the action $-p$ consists in setting the value of the atomic variable p to false. The formula $[(p, \tau, i, \Omega)]\phi$ has to be read "ϕ holds after the performance of the action τp by agent i in the perceptive context Ω".

2.1 Input syntax

The syntax used for the library is the following. Operations and precedence order for unparenthesized expressions in $\mathcal{L}_{\mathsf{LDA}}$ are:

$$\begin{aligned}
false &:= false, F, \bot \\
true &:= true, T, \top \\
box\ operator\ agent\ j &:= [j], \Box j \\
diamond\ operator\ agent\ j &:= < j >, \Diamond j \\
triangle\ operator\ agent\ j &:= \{j\}, \triangle j \\
negation &:= -, \sim, \neg \\
conjunction &:= \&, \wedge, /\backslash, \widehat{} \\
disjunction &:= \vee, \backslash/, | \\
implication &:= - >, \rightarrow \\
double\ implication &:= < - >, \leftrightarrow \\
conjunction &:= \ ;
\end{aligned}$$

Propositions are strings of lowercase letters of length greater than zero, followed by zero or more digits, agents are strings of digits of length greater than zero.

We represent $\mathsf{see}_{i,j}$ in $\mathcal{L}_{\mathsf{OBS}}$ as "$i < j$" with infix right associative operator "$<$" as . We use ";" to separate observations in a perceptive context, and for the dynamic operator introduced as: $[(p, \tau, i, \Omega)]$ we will use $i + p$ or $i - p$ to represent the Boolean value of the variable p for the agent i. And "[(", ")]" will be used to open and close the definition of the operator. For example, if $\Omega = \{\mathsf{see}_{i,i}, \mathsf{see}_{j,i}, \mathsf{see}_i\mathsf{see}_{j,i}\}$. Then the $\mathcal{L}_{\mathsf{DLDA}}$ operator $[(p, +, i, \Omega)]$ is written as:

```
[(i+p;i<i;j<i;i<j<i)]
```

For readability, we allow comments starting from a character '#' to the end of the line, and all contiguous white space characters including new lines are interpreted as a single space.

3 Example

The example consists in a simple scenario of human-robot interaction from the famous Sally-Anne false belief's task from the psychological literature on Theory of Mind [5]. As discussed on our companion paper [2].

We assume that $Agt = \{1, 2\}$ where 1 denotes a human and 2 denotes a robot. The human and the robot are standing in front of each other on the opposite sides of a table. The robot has a black ball, a grey ball, and two boxes in front of him. Initially, the human has not previous knowledge of the setting, and the robot, doesn't has any knowledge about the human's knowledge. Then, the robot puts the black ball inside the box no.2, while it is aware that the human is watching his actions. And the robot believes that the human believes that if a ball is in a given box, then that ball is not in the other box. From this setting, the robot should be able to deduce that the human believes that the black ball is not in box no.1.

```
# Observe the semicolon ';' means conjunction (with the least precendence)
# it is convenient as we usually create belief bases on conjunctive form.
#Hypotesis 1
-{1}b1 & -{1}b2;       # Human doesn't explicitly believe either ball
-{1}g1 & -{1}g2;       # is in either box
-{2}{1}b1 & -{2}{1}b2; # Robot doesn't explicitly believe the human believe
-{2}{1}g1 & -{2}{1}g2; # if either ball is in either box
#Hypotesis 2
{2}({1}b1->{1}-b2;      # Robot explicitly believes that
    {1}b2->{1}-b1;      # if human believes any ball is in either box
    {1}g1->{1}-g2;      # then it also believes that such ball is not
    {1}g1->{1}-g2);     # in the other box (here enumerated the 4 options)

# As usual with SAT, we use the negation of the formula we want to
# test, because, if the negation is unsatisfiable, the formula holds

# The observation context is both observing each other
# and simultaneosly aware of this fact and of themselves
-[( 2+b2; 1<1; 2<2; 1<2; 2<1; 1<2<1; 2<1<2 )](   # We set b2 true for the robot
    ({2}b2) & ({1}b2) & ({2}{1}b2);# All aware that black ball is in box 2
    [2]{1}-b1                      # Robot can conclude that human belives ...
)                                  # ... that the black ball is not in box 1
```

Which of course, after evaluating the translation, returns that is unsatisfiable.
The tool is available for testing at https://tableau.irit.fr.

4 Implementation

4.1 Software, Architecture and algorithms

We created a tool in the $F\sharp$ programming language (an open source, cross platform ML language for the Common Language Infrastructure (CLI)), that follows closely the paper as reference implementation, with the following speed improvements.

There are two separated API methods, one for the reduction of the dynamic extension, and the second for the evaluation of the satisfability given by the tableau procedure.

For the reduction of the dynamic extension, we implement the exact rewriting as specified in the paper, with no further optimization.

For the propositional case, we added a modern yet simple DPLL SAT solver, we focused more in having a clean and solid functional architecture for this rather than adding all possible heuristics, it is slower (2x-50x) than other modern SAT solver (We benchmarked against Z3 [3]), and also is much simpler (the current implementation of the SAT solver has less than 1k lines of code). However, is written in $F\sharp$, so it is exactly as portable as the library itself, which simplifies enormously the development/testing and integration as compared as using a $C++$ library which is the language most modern SAT solvers are implemented. This solver is used to discard processes, but the solution when available, is given by the tableau itself. So this is only used to help speed up execution, and it can be disabled when calling the library.

We use a reactive asynchronous execution workflow that allows us to aggressively benefit from hardware parallelism when available.

We create a process tree which is the contraction of the tableau tree on the root node and all nodes created by applying a transitional rule. Each process runs a "SAT solver" for the propositional interpretation of the set of variables, and spawns one process for each transitional rule that would apply to the contracted tableau node. If the "SAT solver" is not satisfiable or any of the children sends a message saying it is unsatisfiable, it kills all remaining children and returns with the same message to its father. In other case, when all transitional children return a satisfactory configuration, it returns itself with the appropriate message to its father.

As we use immutable data structures, we can use shared memory between processes, in a safe and fast manner.

It is written entirely for the .net core platform, which runs in an array of architectures and operative systems, that include RaspberyPi, Linux, MacOs, Windows and the Windows 10 IoT which is rapidly increasing the array of hosts.

A trade off for the current version, is that we use a full in-memory approach. So, it runs well with models having few thousands of "modal" tableau nodes and few million propositional variables among them, but fails in much larger models, which we consider is acceptable for the kind of environments/problems the tool is designed for.

References

1. Lorini, E. 2018a. In praise of belief bases: Doing epistemic logic without possible worlds. In Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, 19151922. AAAI Press
2. Lorini, E., Romero, F. 2019a. Decision Procedures for Epistemic Logic Exploiting Belief Bases. Conference 2019 AAMAS.
3. De Moura, Leonardo and Bjrner, Nikolaj. 2008. Z3: an efficient SMT solver. In: Tools and Algorithms for the Construction and Analysis of Systems, pp227-340
4. Fagin, R.; Halpern, J.; Moses, Y.; and Vardi, M. 1995. Reasoning about Knowledge. Cambridge: MIT Press.
5. S. Baron-Cohen AND A. M. Leslie AND U. Frith. 1985. Does the autistic child have a "theory of mind"?. Ignition 21 pp36-46