Formal Methods for multi-agent systems

Sergey Dedukh

University of Koblenz and Landau

1 Introduction and motivation

Deploying of autonomous agents becoming more and more important nowadays. Agents act in complex production environments, where failure of a single agent may cause serious losses. The challenge for multi-agent systems now is how to make sure that the agent will not behave unacceptable or undesirable? Formal methods had been used in computer science as a basis to solve correctness challenges. They represent agents as a high level abstractions in complex systems. Such representation can lead to simpler techniques for design and development.

There are two roles of formal methods in distributed artificial intelligence that are often referred to. Firstly, with respect to precise specifications they help in debugging specifications and in validation of system implementations. Abstracting from specific implementation leads to better understanding of the design of the system being developed. Secondly, in the long run formal methods help in developing a clearer understanding of problems and their solutions. [4]

This report in the first section will cover very briefly the theoretical background consisting of different types of logics and introduced operators. Later we will discuss formal methods for a single autonomous agent and the basic implementation of the interpreter. Next some concepts for multiple communicating agents will be introduced. And, finally, the report will conclude with a brief summary of the discussed topic.

2 Theoretical background

To formalize the concepts of multi agent systems different types of logics are used, such as propositional, modal, temporal and dynamic logics. I this section these logics, their properties and introduced operators will be briefly discussed. Describing the details about interpretations and models of each individual logic is not the purpose of this section and is left out for further reading.

Propositional logic is the simplest one and serves as a fundament for logics discussed further in this section. It is used for representing factual information and in our case is most suitable to model the agent's environment. Formulas in this logic language consist of atomic propositions (known facts about the world) and truth-functional connectives: $\wedge, \vee, \neg, \rightarrow$ which denote "and" "or" "not" and "implies" respectively. [2]

Modal logic extends propositional logic by introducing two different modes of truth: possibility and necessity. In the study of agents, it is used to give

meaning to concepts such as belief and knowledge. Syntactically modal operators in modal logic languages are defined as $\Diamond$ for possibility and $\Box$ for necessity. The semantics of modal logics is traditionally given in terms of sets of the so-called possible worlds. A world here can be interpreted as a possible state of affairs or sequence of states of affairs (history). Different worlds can be related via a binary accessibility relation, which tells us which worlds are are within the realm of possibility from the standpoint of a given world. In the sense of the accessibility relation a condition is assumed possible if it is true somewhere in the realm of possibility and it is assumed necessary if it is true everywhere in the realm of possibility. [3]

Dynamic logic is also can be referred to as modal logic of action. It adds differen atomic actions to the logic language. In our case atomic actions may be represented as actions that agents can perform directly. This makes dynamic logic very flexible and useful for distributed artificial intelligence systems. Necessity and possibility operators of dynamic logic are based upon the kinds of actions available. [1]

Temporal logic is the logic of time. There are several variations of this logic such as:

- Linear or Branching: single course of history or multiple courses of history.
- Discrete or Dense: discrete steps(like natural numbers) or always having intermediate steps (like real numbers).
- Moment-based or Period-based: atoms of time are points or intervals.

We will concentrate on discrete moment-based models with linear past, but consider both linear and branching futures.

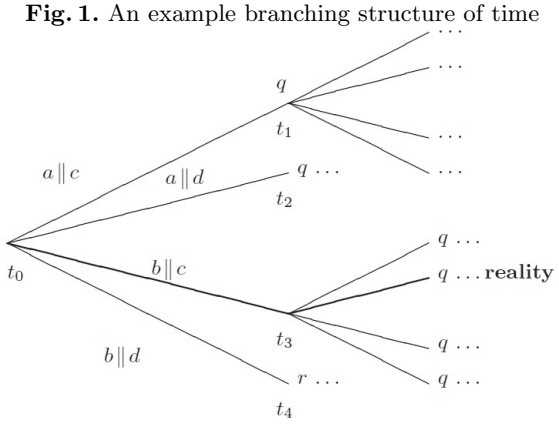
Linear temporal logic introduces several important operators. $p \cup q$ is true at a moment t on a path, if and only if q holds at a future moment on the given path and p holds on all moments between t and the selected occurrence of q. Fp means that p holds sometimes in the future on the given path. Gp means that p always holds in the future on the given path. Xp means that p holds in the next moment. Pq means that q held in a past moment. [4]

Branching temporal and action logic is built on top of both dynamic and linear temporal logics and captures the essential properties of actions and time that are of value in specifying agents. It also adds several specific branching-time operators. A denotes "in all paths at the present moment". The present moment here is the moment at which a given formula is evaluated. E denotes "in some path at the present moment". The reality operator R denotes "in the real path at the present moment". Figure 1 illustrates the example of branching time for two interacting agents.

3 Formal methods for a single agent

For modeling intelligent agents quite often used so-called BDI concept. BDI here stands for three cognitive specifications of agents: beliefs, desires, intensions. To model logic of these specifications we will need to introduce several modal

Fig. 1. An example branching structure of time



operators: Bel for beliefs, Des for desires, Int for intensions and K_h for know how. Considering these operators, for example, the mental state of an agent who desires to win the lottery and intends to buy a lottery ticket sometime, but does not believe that he will ever win can be represented by the following formula: $DesAFwin \wedge IntEFbuy \wedge \neg BelAFwin$. For simplification in future we will consider only those desires which are mutually consistent. Such desires are usually called goals.

It is important to note several important properties of intensions, which should be maintained by all agents[5]:

1. Satisfiability: $xIntp \rightarrow EFp$. This means that if p is intended by x, then it occurs eventually on some path. Intension following this condition is assumed satisfiable.
2. Temporal consistency: $(xIntp \wedge xIntq) \rightarrow xInt(Fp \wedge Fq)$. This requires that if an agent intends p and intends q, then it (implicitly) intends achieving them in some undetermined temporal order: p before q, q before p, or both simultaneously.
3. Persistence does not entail success: $EG((xIntp) \wedge \neg p)$ is satisfiable. This is quite intuitive: just because an agent persists with an intention does not mean that it will succeed.
4. Persist while succeeding. This constraint requires that agents desist from revising their intentions as long as they are able to proceed properly.

The introduced above concepts may be used in each of two roles of formal methods introduced earlier. There are two mostly used reasoning techniques to decide agent's actions: theorem proving and model checking. The first one is more complex in terms of calculations, when the second one is more practical, but it requires additional inputs, though it does not prove to be a problem in several cases.

Considering the practical implementation, the architecture of abstract BDI-interpreter can be described as follows. The inputs to the system are called

events, and are received via an event queue. Events can be external or internal for the system. Based on its current state and input events the system selects and executes options, corresponding to some plans. The interpreter continually performs the following: determines available options, deliberates to commit some options, updates its state and executes chosen atomic actions, after that it updates the event queue and eliminates the options which already achieved or no longer possible.

```
BDI–Interpreter
initialize_state();
do
    options := option–generator(event–queue, B, G, I);
    selected–options := deliberate(options, B, G, I);
    update–intentions(selected–options, I);
    execute(I);
    get–new–external–events();
    drop–successful–attitudes(B, G, I);
    drop–impossible–attitudes(B, G, I);
until quit.
```

As was mentioned above options are usually represented by plans. Plans consist of of the name or type, the body usually specified by a plan graph, invocation condition (triggering event), precondition specifying when it may be selected and add list with delete list, specifying which atomic propositions to be believed after successful plan execution. Intentions in this case may be represented as hierarchically related plans.

Getting back to the algorithm and assuming plans as options, the option generator may look like the following. Given a set of trigger events from the event queue, the option generator iterates through the plan library and returns those plans whose invocation condition matches the trigger event and whose preconditions are believed by the agent.

```
option–generator(trigger–events, B, G ,I)
options := {};
for trigger–event ∈ trigger–events do
    for plan ∈ plan–library do
        if matches(invocation(plan, trigger–event) then
            if provable(precondition(plan), B) then
                options := options ∪ plan;
return options.
```

Deliberation of options should conform with the execution time constraints, therefor under certain circumstances random choice might be appropriate. Sometimes lengthy deliberation becomes possible by introducing metalevel plans into plan library, which form intentions towards some particular plans.

```
deliberate(options)
if length(options) ≤ 1 then return options;
```

```
else metalevel-options :=
            option-generator(b-add(option-set(options)));
    selected-options := deliberate(metalevel-options);
    if null(selected-options) then
        return random-choice(options);
    else return selected-options.
```

4 Formal methods for multiple agents

Coordination is one of the core functionalities needed by multiagent systems. Especially when different agents autonomous and have different roles and possible actions.

One of the approaches developed by Singh [6] represents each agent as a small skeleton, which includes only the events or transitions made by the agent that are significant for coordination. The core of the architecture is the idea that agents should have limited knowledge about designs of other agents. This limited knowledge is called a significant events of the agent. Events can be of the four main types:

- flexible, which can be delayed or omitted,
- inevitable, which can be only delayed,
- immediate, which agent willing to perform immediately,
- triggerable, which the agent performs based on external events.

These events are organized into skeletons that characterize the coordination behavior of agents. The coordination service is independent of the exact skeletons or events used by agents in a multiagent system.

To specify coordinations a variant of linear-time temporal language with some restrictions is used. For that purpose two temporal operators are introduced: $\cdot$ - before operator, and $\odot$ - the operator of concatenation of two time traces, first of which is finite. Such special logic allows a variety of different relationships to be captured.

5 Conclusion

Formal methods provide a logic abstraction for multiagent systems. They help to find self-consistent models of agent's behavior. However relatively high complexity do not allow these methods to be implemented in real time systems. Therefore the role of formal methods nowadays is limited to debug, validate and design purposes.

Also some implementation details were discussed that might be useful for our project. Among them is the concept of complex plans and plan libraries for agent internal implementation. For multiagent systems concept of agent skeletons for coordination may also be used in designing agents' actions.

References

1. Dexter Kozen, J.T.: Logics of program. In Jan van Leeuwen, editor, Handbook of Theoretical Computer Science B, 789–840 (1990)
2. Enderton, H.B.: A Mathematical Introduction to Logic. Academic Press, San Diego (1972)
3. Kripke, S.A.: Semantical analysis of modal logic i: Normal modal propositional calculi. Zeitschrift für Mathematische Logik und Grundlagen der Mathematik 9, 67–96 (1963)
4. Munindar P. Singh, Anand S. Rao, M.P.G.: Formal methods in dai: Logic-based representation and reasoning. In: Multiagent Systems A Modern Approach to Distributed Artificial Intelligence, pp. 331–376. The MIT Press, Cambridge, Massachusetts (1999)
5. Singh, M.P.: A critical examination of the cohen-levesque theory of intentions. In Proceedings of the 10th European Conference on Artificial Intelligence pp. 364–368 (1992)
6. Singh, M.P.: A customizable coordination service for autonomous agents. In Proceedings of the 4th International Workshop on Agent Theories, Architectures and Languages (ATAL) (1997)