Liet. matem. rink, **46***, spec. nr.*, 2006, 232–237

Sequent calculus usage for BDI agent implementation

Adomas BIRŠTUNAS (VU)
e-mail: adomas.birstunas@mif.vu.lt

Abstract. BDI logic is widely used to describe agent based systems, since it can express a lot of different real world domains with three main operators: belief, desire and intention. There are lots of works where BDI logic is used as descriptive language, but authors do not talk about implementation issues ([5,2,4,8,1]). There is even known general agent implementation scheme, which do not describe how to use formal logic for such scheme implementation ([12]). The main aim of this paper is to show the bridge, which connects formal and practical parts of logic. We describe how sequent calculus can be used for implementing some parts of general scheme. In this paper, we show how flexible constraints can be implemented. Sequent calculus usage is illustrated with simple examples.

Keywords: agent implementation, BDI logic, sequent calculus.

1. Introduction

Humans always tried to make machines or systems work for their purposes. It is especially applicable in our days when systems and machines are used in most areas of our life. We want to make machines work independently as possible. It is the reason why, we can see a great interest on agent research in recent years. Agent can be defined as entity (machine, software system ...), which can change environment via its actions, can be influenced by environment, and has the following properties: autonomy, proactiveness, reactivity and social ability ([12]). The autonomy is the most important property when we want to make systems work independently.

BDI agent is an agent which is mostly described with three main concepts: beliefs (corresponds the information agent has about the world), desires (states of the world agent wants to be true in an ideal world), and intentions (states of the world agent tries to achieve). Agents from different domains can be described with these three concepts. These concepts are intuitively understandable, and, it is not surprise, that there are known a lot of BDI agent application ([12,5,9,2,4,8,1]). BDI logic is modal logic with three main modalities: BEL (for agents beliefs), $DESIRE$ (for desires), and $INTEND$ (for intentions).

The core thing in agent implementation is independent decision making. We have to construct an agent which selects an action to perform independently and considering to the current agent vision of the world state. Such a decision making assures the autonomy property of an agent, and it is the hardest part of agent implementation. Wooldridge suggested some general scheme for BDI agent implementation ([12]). General scheme describes functions used and these functions usage in a scheme. Unfortunately, implementation issues of these functions was left opened. In this paper, we

show how such functions implementation can be based on sequent calculus. Sequent calculus is well known, researched and is suitable for automation. General scheme do not force to use formal logic method at all, but if we do not use formal method, then BDI logic will be used only as descriptive language. In other words, agent, implemented without formal logic method usage, cannot be called real BDI agent.

In Section 2, we introduce BDI logic we use, and we present general scheme for BDI agent implementation. In Section 3, we shown how sequent calculus can be used for implementing some parts of BDI agent, and how flexible constraints for intentions can be implemented.

2. BDI logic and general scheme

In this section, we describe syntax of used BDI logic. In our paper, we deal with BDI logic formulas with quantified agents.

We define BDI formulas as follows: if p is propositional symbol, then p is formula; if ϕ, ψ are formulas, i is constant for agent, x is variable for agent, $M \in \{BEL, DESIRE, INTEND\}$, then $\neg\phi$, $\phi \vee \psi$, $M_i(\phi)$, $\forall x M_x(\phi)$, $\exists x M_x(\phi)$ are also formulas. We do not use $\wedge, \supset, \Leftrightarrow$ and formulas $\phi \wedge \psi, \phi \supset \psi, \phi \Leftrightarrow \psi$ are abbreviations of $\neg(\neg\phi \vee \neg\psi)$, $\neg\phi \vee \psi$, $(\phi \supset \psi) \wedge (\psi \supset \phi)$ respectively. Formulas of the shape $M_i(\phi)$ we call M modalized formulas, or just modalized formulas to denote formulas of all three types of M. If Γ_1, Γ_2 are sets of BDI logic formulas, then expression of the shape $\Gamma_1 \rightarrow \Gamma_2$ is a sequent of BDI logic.

We use fragment of the rich multi-agent BDI logic ($LORA$ – Logic of Rational Agents) introduced in [12]. We do not use mutual beliefs, and we use restricted quantifiers. We use quantifiers only for agents. Remark, that expression of the shape $\forall x BEL_x(\exists y(INTEND_y(\phi))$ is formula, but expression of the shape $\forall x \exists y(BEL_x(INTEND_y(\phi))$ is not. Logics with quantified agents are used then number of agents are not known in advance, or just for getting more compact expressions. Similar BDI logic fragment is used in [7], where sequent calculus $MBQ*$ and decision procedure are introduced. In [7], $DESIRE$ and $INTEND$ operators are omitted, since for these operators nothing special needed to get decidability. We use Kripke possible worlds semantics, which is fully described in [12].

Now we present general scheme for BDI agent implementation ([12]). Agents behaviour can be understood as a loop consisted of 3 steps: 1) getting information from the environment; 2) selecting an action for execution; 3) executing selected action (changing the environment). This scheme shows how agent of any type must work. In BDI model, agent (besides its actions) is mostly described using 3 main concepts: belief, desire, intention. Every belief (desire, intention) can be modeled as BEL ($DESIRE$, $INTEND$) modalized formula. Current agent state can be defined by sets of current beliefs, desires and intentions. Agent must take into account its beliefs, desires and intentions during action selection. Action selection is the most important part of the agent implementation, since it ensures the main property – autonomy. In [12], Wooldridge suggest to separate action selection into 3 functions (*options*, *filter*, *plan*) as it is shown in Fig. 1. In Fig. 1, p denotes all information obtained from the environment, B, D, I – sets of all agents beliefs, desires, intentions, and π denotes

```
1.   while true do
2.      get next percept p;
3.      B := brf(B, p);
4.      D := options(B, I);
5.      I := filter(B, D, I);
6.      π := plan(B, I);
7.      execute(π);
8.   end while
```

Fig. 1. General scheme for BDI agent implementation.

some agents plan. Every plan consists of ordered list of actions – $\pi = \alpha_1, \alpha_2, \ldots, \alpha_k$ (α_i is action). In other words, plan is some simple instructions for an agent.

As shown in general scheme, function *brf* is used to renew current agents beliefs according to information obtained from the environment. In function *options*, agent generates its desires according to new beliefs and last intentions. Of course, some intentions can contradict to chosen desires or to each other. Function *filter* is used to leave only such intentions, those do not contradict to agents current beliefs, desires and intentions. Finally, agent must select some plan to execute (function *plan*). In function *execute*, agent just performs all actions described in a selected plan π.

Only three functions deals with BDI logic formulas, so, only for these functions implementations sequent calculus can be used. We do not talk about function *options*, since, generally, agent generates the same desires every time. There are left functions *filter* and *plan*, where sequent calculus usage is rational. In the next section we discuss how these functions can be implemented.

For example, we model food gatherers. The environment is a grid-like world, where agents can move from one cell to a neighbouring cells if there is no other agent in that cell. Food can appear in every cell at random. Agents must gather food and bring it to the special depot cell. This example was taken from the first contest on Multi-Agent systems organized during CLIMA-VI[1] ([3]).

Considering this example, all agents have desire to test all cells and bring all food to depot, and these desires do not change.

3. Functions *plan* and *filter* implementation

In function *plan*, we have to select one plan according to current beliefs and intentions. In works [11,9], there are used invocation conditions for actions. If invocation condition is satisfied then action must be selected. In works [11,9], authors talk about invocation conditions without any context of general BDI agent implementation scheme, but such an approach can be easily adopted for function *plan* implementation.

For every predefined plan π_j, we set some condition formula F_j. We put all pairs (formula and plan) in the priority queue, as shown in Fig. 2.

For example, suppose, that agent has an intention to reach a depot, which is on the second cell on the right from current position, and agent has two plans to reach

[1] CLIMA-VI – VI International Workshop on Computational Logic in Multi-agent Systems.

1) $F_1 - \pi_1$;
...
k) $F_k - \pi_k$;

Fig. 2. Scheme for function *plan* implementation.

depot: $\pi' = \alpha_r, \alpha_r$, and $\pi'' = \alpha_d, \alpha_r, \alpha_r, \alpha_u$ (here α_r – 'go right', α_d – 'go down', α_u – 'go up'). Both plans, have instructions how to reach depot. It is wise to select plan π' only if agent has an intention to reach depot and there is no block on the left cell. Plan π'' can be selected if agent just have an intention to reach depot. Suppose, that $On(x, y) = t \Leftrightarrow$ agent is on the cell (x, y), $Block(x, y) = t \Rightarrow$ there is a block on the cell (x, y) (such as other agent). So, for plan π', we may set condition $INTEND_I(On(x_c + 2, y_c))\&\neg Block(x_c + 1, y_c)$, and for plan π'' condition – $INTEND_I(On(x_c + 2, y_c))$. Of course, it is rational to prefer plan π' to π'' , since it is shorter, and it is why, we set higher priority for plan π' then for plan π''.

Particular schemes are predefined for every agent. Function *plan* gets set of formulas for current beliefs (define them by Γ_1) ant set of formulas for current intentions (Γ_2). According to scheme in Fig. 2, agent checks whether condition formula F_1 is satisfied if current beliefs and intentions are true. If formula F_1 is satisfied, then function *plan* returns plan π_1, otherwise, agent checks whether formula F_2 is satisfied, and so on. At least one formula F_j must be satisfied. For such a formula check, sequent calculus can be used. If we want to check whether formula F_j is satisfied if current beliefs and intentions are true, we have to check whether the following sequent is derivable: $\Gamma_1, \Gamma_2 \rightarrow F_j$. If sequent is derivable, then condition formula F_j is satisfied and π_j is returned.

In function *filter*, we have to delete all inconsistent intentions from all possible. Like in function *plan*, we have predefined set of all possible plans, in function *filter* we have predefined set of all possible intentions (it is a set of $INTEND$ modalized formulas: $INTEND_I(\phi_1), \ldots, INTEND_I(\phi_l)$). For every intention $INTEND_I(\phi_j)$, we have to define condition G_j. If condition is satisfied, then intention is inconsistent with desires (Δ_1) or beliefs (Δ_2). So, we have to check every condition formula G_j and if condition formula G_j is satisfied, then we have to delete intention $INTEND_I(\phi_j)$. After all condition check, function *plan* returns all nondeleted intentions. Sequent calculus can be used to check whether condition G_j is satisfied. If sequent $\Delta_1, \Delta_2 \rightarrow G_j$ is derivable, then condition is satisfied and intention $INTEND_I(\phi_j)$ must be deleted.

In such a function *filter* implementation, we have predefined conditions and these conditions cannot change during agent life. We present new approach for function *filter* implementation. The main idea is to make condition as a special type of belief. We suggest to use such a BEL modalized formula for condition: $BEL_I(\beta \rightarrow \neg INTEND_I(\phi_j))$ (here β is a condition formula). It says, that agent beliefs, that if some condition β is satisfied, then it does not intend to achieve ϕ_j. We define all such beliefs for conditions by Δ_3. So, if sequent $\Delta_1, \Delta_2, \Delta_3 \rightarrow BEL(\neg INTEND_I(\phi_j)$ is derivable, then intention $INTEND_I(\phi_j)$ must be deleted.

It is function *filter* implementation with flexibility property, since agent can drop old and create new conditions for intentions during its life in function *brf*. It even allows

to obtain such beliefs for conditions from other agents during communication. One another advantage is that for every intention, we have to derive only one sequent to check whether it contradicts to all known conditions, no matter, how many conditions it must satisfy.

For example, for $INTEND_I(On(x_c + 2, y_c))$, agent can have condition: $BEL_I(\exists i\, BEL_i(Block(x_c + 2, j_c)) \rightarrow \neg INTEND_I(On(x_c + 2, y_c)))$ (agent believes, that if there are some agent, which believes that there is a block on the cell $(x_c + 2, y_c)$, then it does not intend to reach that cell). During agent life, there can be some new condition obtained. Suppose, that agent γ informs that it is not wise to go to the depot if there agent γ is in the depot (say, because it stays there for a long time). In function *brf*, our agent can generate belief for new condition for intention $INTEND_I(On(x_c + 2, y_c))$: $BEL_I(BEL_\gamma(On(x_c + 2, j_c)) \rightarrow \neg INTEND_I(On(x_c + 2, y_c)))$ (agent believes, that if agent γ believes that it is on the cell $(x_c + 2, j_c)$, then our agent does not intend to reach that cell).

4. Conclusion

In this paper, we show how BDI agent can be implemented with formal logic usage. We present general scheme for BDI multi-agent implementation, and suggest to use sequent calculus while implementing some parts of it. We show how flexible constraint conditions for intentions can be implemented. All implementation schemes are illustrated with examples.

References

1. M.M. Cheikhrouhou, BDI-oriented agents for network management, in: *GLOBECOM'99*, vol. 3 (1999), pp. 1964–1968.
2. S. Coffey, D. Gaertner, Using pheromones, broadcasting and negotiation for agent gathering tasks, in: *Pre-proc. CLIMA VI* (2005), pp. 267–273.
3. M. Dastani, J. Dix, The first contest on multi-agent systems based on computational logic, in: *Pre-proc. CLIMA VI* (2005), pp. 261–266.
4. C. Dixon, F. Gago, M. Fisher, W. Hoek, *Using Temporal Logics of Knowladge in the Formal Verification of Security Protocols*, Tech. r. ULCS-03-022.
5. N. Jennings, M. Wooldridge, Applications of intelligent agents, in: N. Jennings and M. Wooldridge (Eds.), *Agent Technology*: *Foundations, Applications, and Markets*, Springer (1998), pp. 3–28.
6. R.A. Kowalski, F. Sadri, From logic programming towards multi-agent systems, *Annals of Mathematics and Articial Intelligence*, **25**(3–4), 391–419 (1999).
7. R. Pliuskevicius, A. Pliuskeviciene, Decision procedure for a fragment of mutual belief logic with quantified agent variables, to appear in: *LNAI 3900*, *Proc. CLIMA VI* (2006).
8. O. Rana, M. Winikoff, L. Padgham, J. Harland, Applying conflict management strategies in BDI agents for resource management in computational grids, in: *Proc. of the Australasian Conference on Computer Science*, Melbourne, ACM Press (2002).
9. A.S. Rao, M. Georgeff, BDI agents: from theory to practice, in: *Proc. ICMAS-95* (1995), pp. 312–319.
10. A.S. Rao, AgentSpeak(L): BDI agents speak out in a logical computable language, in: *MAAMAW'96*, LNAI 1038, Springer (1996).
11. M. Wooldridge, N. Jennings, Intelligent agents: theory and practice, *The Knowledge Engineering Review*, **10**(2), 115–152 (1995).
12. M. Wooldridge, *Reasoning about Rational Agents*, The MIT Press (2000).

REZIUMĖ

A. Birštunas. Sekvencinio skaičiavimo panaudojimas BDI agentų realizavimui

Pateikta bendra BDI agento realizavimo schema. Parodyta kaip tam tikrų dalių realizavimui gali būti panaudotas sekvencinis skaičiavimas. Pasiūlyta, kaip galima realizuoti lankstesnę funkciją *filter*.