

Model Checking for PRS-Like Agents

Wayne Wobcke¹, Marc Chee¹, and Krystian Ji^{1,2}

¹ School of Computer Science and Engineering,
University of New South Wales, Sydney NSW 2052, Australia
{wobcke, marcchee, krystianj}@cse.unsw.edu.au

² National ICT Australia, University of New South Wales,
Sydney NSW 2052, Australia
krystian.ji@nicta.com.au

Abstract. The key problem in applying verification techniques such as model checking to agent architectures is to show how to map systematically from an agent program to a model structure that not only includes the possible behaviours of the agent in its environment, but which also captures appropriate mental notions, such as belief, desire and intention, that may be used by the designer to reason about the agent. In this paper, we present an algorithm providing a mapping from agent programs under a simplified PRS-type agent architecture to a reachability graph structure extended to include representations of beliefs, goals and intentions, and illustrate the translation with a simple “waypoint following” agent. We conclude with a brief discussion of the differences between the internal (operational) notion of intention used in the architecture and the formal (external) notion of intention used in the modelling.

1 Introduction

The issues of specification and verification of agent programs are of critical importance in developing reliable software agents. This paper is concerned with applying model checking techniques to agents built using PRS-type agent architectures. The PRS-style architecture is useful for a class of applications requiring the agent to achieve a balance between reactivity to changes in the environment, and deliberation to exhibit goal-directed behaviour. This balance, which is partly under the programmer’s control and partly pre-defined by the architecture, accounts for much of the complexity of designing and reasoning about agent programs for this architecture. Because of the inherent intricacy of this computational model, methodologies and tools for building agents of this type are an important step towards making this paradigm more widely adopted.

The PRS-style architecture operates with notions of belief, desire and intention, and is accordingly described as a “BDI architecture”. However, using logical versions of idealized concepts of belief, desire and intention to model this architecture is inadequate for validating agent programs, because these idealized concepts do not match their specific meanings as used in the architecture. Rather, the “PRS-like” agent uses a plan library to store pre-defined plans for achieving goals, explicit beliefs to represent information about the environment that guides the selection of actions both

in response to events and to refine plans, and various strategies for action execution to realize commitments to the chosen plans. Specification and verification of PRS-type agent programs requires a formal model of *these* mental notions, and then for these attitudes to be derived, in specific cases, in a principled way, from the execution models of agent programs implemented within the architecture. So, while Rao and Georgeff [13] presented a model checking algorithm for their BDI logic, they did not address the central question of generating these models from agent programs in the first place.

Recently, there have been some efforts to apply model checking to classes of systems that include some characteristics of agency. In the work of Bordini *et al.* [1], AgentSpeak(L) agent programs, Rao [10], are coded in PROMELA for verification using the SPIN model checker; however, the meaning of the belief, goal and intention modalities is not captured semantically, but instead specific beliefs, goals and intentions of the agent are encoded as extra state information in SPIN models. Under this approach, there is thus no way to answer the question of whether these mental notions are modelled correctly in the semantics, nor even to determine all but the simplest logical properties of these notions so as to check whether they satisfy intuitively correct properties. On the other hand, in the work of van der Meyden and Shilov [15], Penczek and Lomuscio [8] and Raimondi and Lomuscio [9], model checking techniques are applied to a class of distributed programs in a principled way, but these models involve only the knowledge modality (and this understood in the way specific to Fagin *et al.* [4]), so the agents do not have the complexity of PRS-type agents (do not have beliefs, desires and intentions).

Our aim in this paper is to develop a technique for applying model checking for PRS-type agents. The key problem is to show how any agent program executing within the architecture can be mapped onto a semantic structure that can be generated computationally and that includes not only the possible behaviours of the agent in a potentially highly nondeterministic environment, but faithfully captures the mental states (beliefs, desires and intentions) of the agent at any time in its execution. Inevitably, in contrast to the logical formalisms developed in previous work, e.g. Cohen and Levesque [3], Rao and Georgeff [11, 14], Wooldridge [19], these mappings must incorporate architecture-specific aspects both in the definition of the possible execution paths of an agent program and in how mental notions are realized in agents implemented in the architecture.

This paper is organized as follows. We first briefly summarize the PRS-like agent architecture. Then the main part of the paper consists of a description of how the execution structures of an agent program can be represented in a reachability graph. We give an algorithm for computing a reachability graph for any PRS-like agent program from a given initial configuration, then an example of this construction for a simple “waypoint following” agent. The example shows how the behaviour of the agent in changing its intentions in response to events in the environment is modelled.

This work is part of a larger project aimed towards developing sound logical foundations and computational tools for understanding and reasoning about the family of PRS-like agent architectures. This project includes development of a precise operational semantics that synthesizes a number of properties common to various PRS-like architectures, Wobcke [16], and the development of formal approaches to the semantics of the

mental notions employed in this type of architecture, Wobcke [17], extended to belief update and the modelling of success and failure through action attempts, Wobcke [18]. To make this paper self-contained, many of the formal details must be omitted: the reader is referred to this earlier work for such details.

2 PRS-Like Agent Architectures

The class of agent architectures studied in this paper are the *PRS-like* architectures as defined in Wobcke [16], which is supposed to cover PRS (Procedural Reasoning System), Georgeff and Lansky [5], and variants such as UM-PRS, C-PRS, AgentSpeak(L), dMARS, JAM and JACK Intelligent AgentsTM. The agent's computation cycle can be conveniently described with reference to the simplified interpreter shown in Figure 1, adapted from Rao and Georgeff [12]. In this abstract interpreter, the system state consists of sets of beliefs B and intentions I . Each element of I is a partially executed hierarchical plan. Each cycle of this simplified interpreter runs as follows. The process begins with the external event (if there is one) triggering instances of pre-existing plans from the plan library (the belief set is used to test the context conditions of these plans): one selected plan is then added to the intention structure. The plans in the set of intentions that are applicable in the current state (i.e. the precondition of the next action in the plan body is believed to hold) constitute the options available to the agent in that state. The agent selects one such plan for execution, involving, if the next action in this plan is of the form *achieve* γ , selecting an instance of a plan from the plan library whose postcondition logically implies γ and whose precondition and context are believed, and updating the set of intentions accordingly. The agent then attempts the next action in the chosen plan, updating the intention structure appropriately in the case of a test action. After making a new observation and revising beliefs, the set of current intentions is further updated, first by removing those plans that are believed to have successfully completed (whose postcondition is believed), then by dropping those believed to be im-

```

Abstract BDI Interpreter:
  initialize-state(B, I);
  do
    get-external-event(e);
    new-options := trigger-plans(e, B);
    selected-option := select-option(new-options);
    update-intentions(selected-option, I);
    selected-intention := select-intention(I);
    execute(selected-intention);
    update-intention(selected-intention, I);
    get-observation(o);
    update-beliefs(o, B);
    drop-successful-plans(B, I);
    drop-impossible-plans(B, I)
  until quit

```

Fig. 1. Abstract BDI Interpreter

possible to complete (whose termination condition is believed). For the very first cycle, we can assume the agent has no plans and some arbitrary set of beliefs.

The internal states of the agent are pairs $\langle B, I \rangle$ where B is the set of beliefs and I is the set of intentions (partially executed hierarchical plans). We assume a finite propositional language \mathcal{L} for representing the beliefs of the agent, and the agent’s belief set B at any time is assumed to be a consistent set of literals of \mathcal{L} . The conditions of each plan in the plan library are also formulae of \mathcal{L} , and the body of each plan is a program. The language of programs consists of a set of atomic programs, including special actions *achieve* γ (where γ is a formula of \mathcal{L}) and an “empty” program Λ , and conditional and iterative statements **if** α **then** π **else** ψ and **while** α **do** π (where α is a formula of \mathcal{L} and π and ψ are programs). Note that the tests in these statements are tests on the agent’s beliefs, not on the state of the environment.

The agent’s selection mechanisms are constrained as follows:

- Given a set of options triggered by an event (whose precondition and context are believed), the function *select-option* returns a randomly selected element of maximal priority within that set;
- Given a set of intentions, the function *select-intention* returns a randomly selected element of maximal priority in the subset of this set of plans which are applicable (whose preconditions are believed) in the current state; moreover, if a plan is chosen to break a tie, as long as its execution is believed successful, it continues to be chosen on future cycles until it terminates.

The functions for belief and intention update are as follows. For simplicity, it is assumed that the observations on each cycle correspond to a consistent set of literals L . Then the belief revision function can be defined as the function that removes the complement \bar{l} of each literal l in L from the belief set B (if it is contained in B) and then adds each (now consistent) literal l of L to B . The only subtlety with the intention update function is in defining which plans may be dropped as achieved or infeasible. We take it that on failure, only whole plans can be dropped, and on success, only whole plans or the initial actions of any plan can be dropped (if an action of the form *achieve* γ is dropped, the *achieve* action and all its subplans are removed from the intention structure). Dropping a subplan leaves the *achieve* subgoal that resulted in the plan’s selection as the next step to be executed in that hierarchical plan.

3 Reachability Graphs for PRS-Like Agents

In this section, we show how the execution of a PRS-like agent is modelled using reachability graphs. A reachability graph is a structure representing a class of program executions that also has the flavour of a mathematical model, in that formulae (including modal and temporal formulae) can be evaluated at nodes in the graph. Reachability graphs are used as the representation underlying the SPIN model checker, Holzmann [7], and are particularly appropriate for PRS-like agents because computing the entire state space is impractical. A reachability graph essentially represents a simulation of multiple executions of a program, with nodes in the graph corresponding to execution states, and transitions in the graph corresponding to atomic action execution. These executions all

start with a given initial configuration, so only a portion of the entire state space (those states “reachable” from the starting configuration) are explored. Once a reachability graph is constructed, evaluating the truth of formulae is relatively straightforward, e.g. SPIN includes an algorithm for evaluating Linear Temporal Logic formulae with respect to the execution paths of distributed reactive systems. However, applying the reachability graph method for PRS-like agents is more complicated, because not only do program executions need to be modelled, but the beliefs, goals and intentions of the agent need to be represented in the graph structure. Moreover, the execution of an agent program depends on the mental states of the agent: first, because the evaluation of tests in conditional and iterative statements depends on the agent’s beliefs, and second because the intentions of the agent influence which actions the agent will eventually perform. Thus there is a close interaction between the agent’s program execution and its mental states.

The nodes of the reachability graph, the *execution states*, must be related systematically to the internal (operational) states of the PRS-like interpreter, yet cannot be identical to those states (as in the work of Bordini *et al.* [1]) because this would leave the important intentional notions unformalized. On the other hand, execution states cannot be identical to environment states, because the reachability graph construction algorithm relies on termination through eventually revisiting execution states, so all the information relevant to the execution state must be used when identifying execution states. Our approach is to distinguish between the information explicitly associated with a node in the reachability graph and the information used for identifying execution states, which is also represented in the graph but not as information explicitly encoded in states. The explicit information in an execution state is just the state of the environment together with any event that occurs at that state.

A single reachability graph may contain multiple computation trees, corresponding to different starting states – these are “possible worlds” the agent could be inhabiting (*actual worlds*). A reachability graph may also contain “possible worlds” corresponding to the beliefs of the agent in states in actual worlds (*belief-alternative worlds*). As in the work of Raimondi and Lomuscio [9], the beliefs of the agent are represented using a serial, transitive, Euclidean accessibility relation b between execution states (this means that the agent’s beliefs can be no more fine-grained than the model of the environment). Intentions are modelled in a more complex manner, following our analysis of intention as actions the agent will eventually successfully perform in all actual worlds considered possible by the agent. Some reduction of this nature is necessary if model checking techniques, which construct only execution paths, are to capture a notion of intention. Intentions in reachability graphs are represented using a reflexive, symmetric, transitive accessibility relation a and a boolean i : for any execution state, the a -related states (actual alternatives) are those corresponding to execution states indistinguishable by the agent given its current set of intentions (accordingly, those states in alternative actual worlds in which its intentions must be capable of being fulfilled), and i defines which execution states in a world result from successful action attempts. Execution states in reachability graphs are identified only if they have the same environment state and event, the same set of execution states related under b and a , and the same value of i . This embodies a Markov assumption in that the co-evolution of the agent/environment must be identical for execution states identified using this definition.

Our action semantics is inspired by the computation tree semantics for Propositional Dynamic Logic, Harel [6]. The basic idea is to define a relation \mathcal{R}_π for each program π which enables the execution of each program to be modelled as the set of all its execution structures (actual worlds with associated belief-alternative worlds), the arcs in each tree corresponding to individual performances of atomic actions together with the effect of making an observation (it is further assumed that the agent’s observations are all accurate and at least include whether the postcondition of the action holds). It is assumed that each atomic action π is defined as a binary relation R_π on states of the environment (with $R_\pi \subseteq R_{\text{attempt } \pi}$), and that the occurrence of events in states of the environment is independent of history and dependent only on the chosen action. To model belief revision, special actions *observe* α are used, where α is a conjunction of literals in the language of beliefs \mathcal{L} . The relation $R_{\text{observe } \alpha}$ is defined to respect the operational definition of belief revision (and so is deterministic). Transitions in belief-alternative worlds correspond only to the execution of actions of the form *observe* α .

Our logical language makes use of a formula $do(\pi)$ which is true at an execution state iff the program π is the action about to be executed by the agent at that state. A state in a reachability graph satisfies a formula $do(\pi)$ if the subgraph emanating from that state is isomorphic to an execution structure in \mathcal{R}_π (i.e. are the ways of executing π in that state), taking into account the belief alternatives of the states. The semantics of action must distinguish between doing an action and attempting to do an action, as the PRS-like agent cannot guarantee success. For this, the language of programs includes special actions of the form *attempt* π ; typically an execution state satisfies $do(\text{attempt } \pi)$ while what the agent intends is $do(\pi)$ (a successful attempt to do π).

In our work on the formal semantics of PRS-like agents, we developed a logical language of beliefs and intentions called ADL (Agent Dynamic Logic) for reasoning about BDI agents, based on both Rao and Georgeff’s BDI-CTL [11], which extends CTL (Computation Tree Logic) with modal operators for modelling beliefs, desires (goals) and intentions, and PDL (Propositional Dynamic Logic), which includes modal operators corresponding to program terms. The following definition gives the satisfaction conditions of ADL formulae in reachability graphs. For any reachability graph G , let $\mathcal{I}(G)$ be the subgraph of G that contains only those transitions of G corresponding to successful action executions and their associated observe actions (\forall quantifies over all paths emanating from a state, \diamond is “eventually” with respect to a given path).

Definition 1. *An execution state σ in a reachability graph satisfies a BDI formula as follows.*

$$\begin{aligned} \sigma \models_G B\alpha & \quad \text{if } \sigma' \models_G \alpha \text{ whenever } b(\sigma, \sigma') \\ \sigma \models_G I\pi & \quad \text{if } \sigma' \models_{\mathcal{I}(G)} \forall \diamond do(\pi) \text{ whenever } a(\sigma, \sigma') \\ \sigma \models_G G\gamma & \quad \text{if } \sigma \models_G I(\text{achieve } \gamma) \end{aligned}$$

4 Reachability Graph Construction

In this section, we provide an algorithm for the generation of a reachability graph for an arbitrary PRS-like agent program. The algorithm effectively carries out a breadth-first search of the execution states, beginning with a given set of initial execution states (a -related alternatives), and their sets of epistemic alternatives (b -related alternatives). The

basic idea of the algorithm is that, at each execution state in the graph, for each possible action the agent could attempt at that state and for each possible outcome of executing that attempt (including a possible new event), the search explores a new execution state and its belief-alternative states. The algorithm keeps track of execution states already visited, and terminates when no new execution states are discovered (which is not guaranteed in general, since there can be infinitely many steps in iterative programs and expansions of hierarchical plans).

A high-level description of the general reachability graph construction algorithm is shown in Figure 2. The a -related alternatives of a state are called “intention alternatives” in this description of the algorithm.

```

Reachability Graph Construction:
create initial (intention alternative) states and their belief alternatives
do
  for each new state in the graph
    for each action the agent can attempt in that state
      create a copy of the state and its belief alternatives to represent this choice
      for each possible outcome and observation for the action attempt
        create a new state and its belief alternatives for this outcome
        mark the new state a success state iff it satisfies the action's postcondition
        if an event can occur in this new state
          for each maximal priority applicable plan triggered by this event
            create a new state and its belief alternatives for this outcome
              with this event and the resulting new set of intentions
        set two states to be intention alternatives if they are both successors of intention alternatives
          and are both indexed by the same set of intentions
until no new states are generated

```

Fig. 2. High-Level Reachability Graph Construction Algorithm

The detailed reachability graph construction algorithm is shown in the appendix. As in the high-level description, the algorithm simulates all possible execution paths of the BDI interpreter in a breadth-first search, keeping track of repeated execution states. The variables Σ_i in the algorithm are used to record the new execution states generated at each level of the search, and the algorithm terminates if there are no new states discovered. The algorithm takes two parameters: S , the set of possible initial environment states, which become the initial states in the alternative actual worlds of the agent, i.e. these states are initially all a -related alternatives of one another, and B , the initial set of beliefs of the agent, which generate a set of b -related alternative states for each initial environment state. The execution states in the algorithm are denoted $\sigma_{\{s,e,B,I\}}$ for those in actual worlds and $\sigma_{\{s,s',e,B,I\}}$ for those in belief alternatives, i.e. execution states are indexed by s (the state of the environment), s' (the possible state of the environment in a belief alternative of s), e (any event occurring at the state, ϵ if there is no event), B (the agent's set of beliefs) and I (the agent's set of intentions). Two execution states are identified only if all of these are the same.

As the reachability graph construction algorithm proceeds, a number of relations are computed that represent the structure of the graph. The relation R defines the transitions between execution states, while the accessibility relations b and a represent the belief alternatives and the actual (intention) alternatives. The b and a relations of the R -successors of an execution state σ are defined in terms of the successors of the states

b and a -related to σ . The boolean i is true iff an execution state is the result of a successful action attempt (which is determined by whether the state of the environment satisfies the action’s postcondition).

The algorithm makes use of two selection functions from the PRS-like agent interpreter: `options` and `intentions` return the set of plans and the set of intentions that could be chosen by the interpreter’s functions `select-option` (the maximal priority applicable plans triggered by an event) and `select-intention` (the maximal priority applicable plans possible to be executed). But whereas the agent will choose randomly one of the outputs of these functions in any particular execution, the reachability graph includes execution paths for all of them (intuitively, the agent program must work for all possible outputs of these functions). The algorithm also makes use of the functions `update-intention`, `drop` and `revise`. The function `update-intention` computes the continuations of the program (a test statement is replaced by one reflecting the outcome of the test, an action *achieve* γ is replaced by the continuation of one of its expansions). The functions `drop` and `revise` implement the agent’s intention and belief revision functions: `drop` removes from the intentions any successfully completed action or subplan (one whose postcondition is believed) or any infeasible subplan (one whose termination condition is believed); `revise` implements the agent’s belief revision function. Note that all these functions are independent of the particular agent program.

5 Example

In this section, we illustrate the reachability graph construction algorithm through an example “waypoint following” agent. The waypoint agent has a simple task: it must visit four waypoints, numbered 1–4, in order, whilst not running out of fuel. There is fuel at locations 1 and 3 though this knowledge is not explicit in the agent’s beliefs. Rather, the agent is constructed to have refuelling plans, as shown in Figure 3 for the fuel at location 1 (the agent also has two plans similar to $Refuel_1$ and $Refuel_2$ for the fuel at location 3). The agent is capable of carrying out the actions $visit_i$ and $refuel$; the refueling plans are used in response to a *warn* event in order to direct the agent to a fuel depot where a refuel action is attempted.

The agent always has knowledge of its position, represented as beliefs at_i and $\neg at_j$ ($j \neq i$), and each $visit_i$ action includes a correct observation of the agent’s position. The agent initially has no belief about the fuel level, but after a *refuel* action, correctly observes the state of the fuel tank, represented as a belief $full (= \neg empty)$ or $empty$.

An issue that must be addressed in applying model checking to any agent system is the specification of the action semantics, including the transition relations \mathcal{R}_π for each atomic action and the specification of when events may occur in the environment. For now, we specify this informally for the waypoint agent. For simplicity, the $visit_i$ and $refuel$ actions always succeed, except that on occasion (here only at location 3) a *warn* event occurs. A *warn* event can therefore occur even after a refueling action (the *refuel* action may not provide enough fuel to offset the warning).

The reachability graph construction algorithm must be supplied an initial configuration representing the agent’s beliefs and the possible initial states of the environment: in this example, the agent starts at location 1 (and believes this) and has a full fuel tank

Main
 priority: 0
 pre: *true*
 trigger: *start*
 context: *true*
 body: *visit₁*; *visit₂*; *visit₃*; *visit₄*
 post: $v_1 \wedge v_2 \wedge v_3 \wedge v_4$

<i>Refuel₁</i>	<i>Refuel₂</i>
priority: 1	priority: 1
pre: <i>true</i>	pre: <i>true</i>
trigger: <i>warn</i>	trigger: <i>warn</i>
context: <i>at₁</i>	context: $\neg at_1$
body: <i>refuel</i>	body: <i>visit₁</i> ; <i>refuel</i>
post: <i>full</i>	post: <i>full</i>

Fig. 3. Waypoint Agent Plans

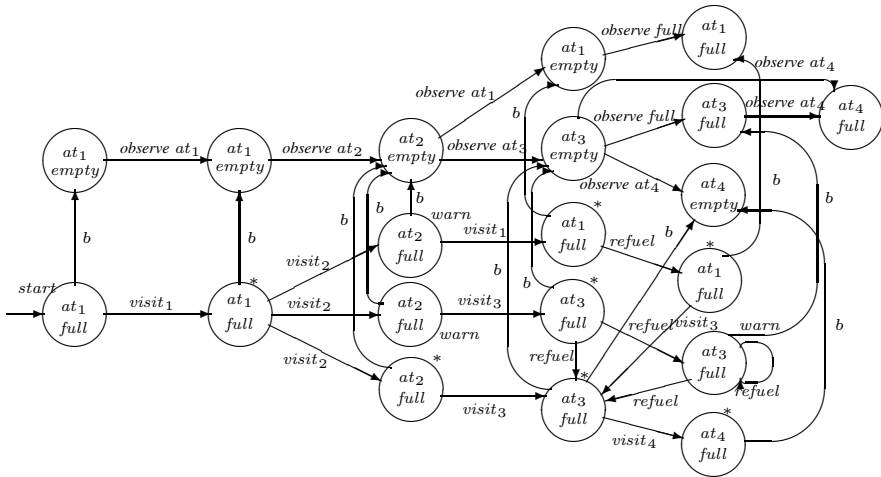


Fig. 4. Waypoint Agent Reachability Graph

(though does not believe this). Thus the initial execution state in the reachability graph has two *b*-related alternatives (one where the tank is *full* and one where it is *empty*), but just one *a*-related alternative (the agent would perform very poorly were it in a world where it had no fuel initially).

A portion of the reachability graph for the waypoint agent is shown in Figure 4 (in the diagram, *a* and *b*-related links between a state and itself are omitted, and states resulting from successful executions are marked with *). The actions labelling the transitions correspond to an atomic action formula π such that $do(\pi)$ is satisfied at the state.

The following lists some examples of the kind of properties of the agent that could be tested using such a graph:

$$\begin{aligned}
& \forall \square (full \Leftrightarrow Bfull) \\
& \forall \diamond at_4 \\
& \forall (Ivisit_i U at_i) \\
& \text{if } warn \in \sigma \text{ then } \sigma \models \forall (Irefuel U Bfull)
\end{aligned}$$

The first two formulae are not satisfied at the initial state of the graph, the second since the agent may become stuck handling an infinite sequence of fuel warnings. This points to one complicated aspect of event handling in the PRS-like architecture. The third and fourth properties represent the persistence of intentions.

6 Conclusion

This paper has provided a formalization of the execution model of a family of BDI agents based on the PRS architecture, and shown how precise notions of belief and intention can be represented in reachability graphs, structures that are suitable for applying model checking algorithms to this class of agents. By clarifying the mental notions underlying this class of agents and by systematically relating these notions to the operational behaviour of the agent interpreter, a more robust methodology underpinning the verification of complex BDI agents is made possible. Moreover, the approach enables a precise semantic theory of PRS-like agents based on these notions to be defined, allowing agent program designers to reason logically about the mental states of the agents with confidence that this reasoning faithfully represents the behaviour of the agent.

However, the (external) notion of intention as encoded in reachability graphs and BDI models differs from the agent’s (internal) notion in that there can be actions that occur in the agent’s intention structures that are not *intentions* as defined in these models. A simple example of this occurs when the agent has adopted a number of plans that are not compatible (i.e. it is not possible to execute all the agent’s plans to successful completion, even when no adverse events occur in the world). In such a case, the theory says that the agent does not really “intend” to execute those plans; it only intends to execute the compatible parts of the plans. Thus the notion of intention encoded in the reachability graph is more a notion of commitment (the agent intends those actions it is “committed” to performing), whereas the notion of intention used in the architecture is something weaker again (the agent is not even committed to carrying out its “intentions”). It remains to be seen whether this notion of intention, which is clearly much simplified from Bratman’s original theory [2], is useful from the agent programmer’s point of view. However, our work presents, for the first time, one way of precisely capturing this complex mental notion in semantic structures that are amenable to verification techniques.

Acknowledgements

This work is funded by an Australian Research Council Discovery Project Grant. Krystian Ji is supported by a UNSW International Postgraduate Research Scholarship and a scholarship from NICTA, National ICT Australia Ltd. National ICT Australia is funded through the Australian Government’s *Backing Australia’s Ability* initiative, in part through the Australian Research Council.

References

1. Bordini, R.H., Fisher, M., Pardavila, C. & Wooldridge, M. (2003) 'Model Checking AgentSpeak.' *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems*, 409–416.
2. Bratman, M.E. (1987) *Intention, Plans, and Practical Reason*. Harvard University Press, Cambridge, MA.
3. Cohen, P.R. & Levesque, H.J. (1990) 'Intention is Choice with Commitment.' *Artificial Intelligence*, **42**, 213–261.
4. Fagin, R., Halpern, J.Y., Moses, Y. & Vardi, M.Y. (1995) *Reasoning About Knowledge*. MIT Press, Cambridge, MA.
5. Georgeff, M.P. & Lansky, A.L. (1987) 'Reactive Reasoning and Planning.' *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI-87)*, 677–682.
6. Harel, D. (1979) *First-Order Dynamic Logic*. Springer-Verlag, Berlin.
7. Holzmann, G.J. (1997) 'The Model Checker SPIN.' *IEEE Transactions on Software Engineering*, **23**(5), 279–295.
8. Penczek, W. & Lomuscio, A. (2003) 'Verifying Epistemic Properties of Multi-agent Systems via Bounded Model Checking.' *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems*, 209–216.
9. Raimondi, F. & Lomuscio, A. (2004) 'Verification of Multiagent Systems via Ordered Binary Decision Diagrams: An Algorithm and Its Implementation.' *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, 630–637.
10. Rao, A.S. (1996) 'AgentSpeak(L): BDI Agents Speak Out in a Logical Computable Language.' in Van de Velde, W. & Perram, J.W. (Eds) *Agents Breaking Away*. Springer-Verlag, Berlin.
11. Rao, A.S. & Georgeff, M.P. (1991) 'Modeling Rational Agents within a BDI-Architecture.' *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning (KR'91)*, 473–484.
12. Rao, A.S. & Georgeff, M.P. (1992) 'An Abstract Architecture for Rational Agents.' *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning (KR'92)*, 439–449.
13. Rao, A.S. & Georgeff, M.P. (1993) 'A Model-Theoretic Approach to the Verification of Situated Reasoning Systems.' *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, 318–324.
14. Rao, A.S. & Georgeff, M.P. (1998) 'Decision Procedures for BDI Logics.' *Journal of Logic and Computation*, **8**, 293–343.
15. van der Meyden, R. & Shilov, N.V. (1999) 'Model Checking Knowledge and Time in Systems with Perfect Recall (Extended Abstract).' in Pandu Rangan, C., Raman, V. & Ramanujam, R. (Eds) *Foundations of Software Technology and Theoretical Computer Science: 19th Conference*. Springer-Verlag, Berlin.
16. Wobcke, W.R. (2001) 'An Operational Semantics for a PRS-like Agent Architecture.' in Stumptner, M., Corbett, D. & Brooks, M. (Eds) *AI 2001: Advances in Artificial Intelligence*. Springer-Verlag, Berlin.
17. Wobcke, W.R. (2002) 'Modelling PRS-like Agents' Mental States.' in Ishizuka, M. & Sattar, A. (Eds) *PRICAI 2002: Trends in Artificial Intelligence*. Springer-Verlag, Berlin.
18. Wobcke, W.R. (2004) 'Model Theory for PRS-Like Agents: Modelling Belief Update and Action Attempts.' in Zhang, C., Guesgen, H.W. & Yeap, W.K. (Eds) *PRICAI 2004: Trends in Artificial Intelligence*. Springer-Verlag, Berlin.
19. Wooldridge, M.J. (2000) *Reasoning About Rational Agents*. MIT Press, Cambridge, MA.

Appendix. Reachability Graph Construction Algorithm

Reachability Graph Construction:

```

 $\Sigma_0 := \{\};$ 
for each  $s \in S$  do
   $e := \text{generate-event}(s);$     %  $\epsilon$  if no event
  for each  $\iota \in \text{options}(\text{trigger-plans}(e, B))$  do
     $I := \{\iota\};$ 
    create  $\sigma_{\{s,e,B,I\}}$  and add to  $\Sigma_0$ ;
    set  $i(\sigma_{\{s,e,B,I\}}) = \text{false};$ 
    set  $b(\sigma_{\{s,e,B,I\}}, \sigma_{\{s,e,B,I\}})$  if  $s \models B$ ;
    create  $\sigma_{\{s,s',e,B,0\}}$  for each  $s' \neq s, s' \models B$ ;
    set  $i(\sigma_{\{s,s',e,B,0\}}) = \text{false};$ 
    set  $b(\sigma_{\{s,e,B,I\}}, \sigma_{\{s,s',e,B,0\}})$  for each  $s' \models B$ ;
  set  $a(\sigma_{\{s,e,B,I\}}, \sigma_{\{s',e',B,I\}})$  for each  $\sigma_{\{s,e,B,I\}}, \sigma_{\{s',e',B,I\}} \in \Sigma_0$ ;
   $i := 0$ ;
do
   $\Sigma_{i+1} := \{\};$ 
  for each new  $\sigma_{\{s,e,B,I\}} \in \Sigma_i$  do
     $\Pi := \text{intentions}(B, I);$     %  $\{\text{env}\}$  if no intention
    replace  $\sigma_{\{s,e,B,I\}}$  by  $|\Pi|$  copies  $\sigma_{\{s,e,B,I\}}^\pi$  of itself;    % split state to represent choices
    set  $i(\sigma_{\{s,e,B,I\}}^\pi) = i(\sigma_{\{s,e,B,I\}})$ ;
    for each  $\sigma_{\{s,s',e,B,i\}}$  such that  $b(\sigma_{\{s,e,B,I\}}, \sigma_{\{s,s',e,B,i\}})$  do
      set  $b(\sigma_{\{s,e,B,I\}}^\pi, \sigma_{\{s,s',e,B,i\}})$ ;
    set  $a(\sigma_{\{s,e,B,I\}}^\pi, \sigma_{\{s,e,B,I\}}^\psi)$  for each such  $\sigma_{\{s,e,B,I\}}^\pi, \sigma_{\{s,e,B,I\}}^\psi$ ;
    for each  $\pi \in \Pi$  do
      for each initial primitive action  $a$  of  $\pi$  (or of an expansion of an initial action  $\text{achieve } \gamma$  ) do
        for each  $(s, s') \in \mathcal{R}_{\text{attempt } a}$  do
           $o := \text{observation}(s', \text{attempt } a);$     % observation associated with action
           $B' := \text{revise}(B, o);$ 
           $I' := \text{drop}(B', \text{update-intention}(I));$     % update due to test or  $\text{achieve } \gamma$  expansion
          create  $\sigma_{\{s',e',B',I'\}}$  and add to  $\Sigma_{i+1}$ ;
          set  $R(\sigma_{\{s,e,B,I\}}^\pi, \sigma_{\{s',e',B',I'\}})$ ;
          set  $i(\sigma_{\{s',e',B',I'\}})$  according to whether  $s' \models \text{post}(a);$     % intend success
          set  $b(\sigma_{\{s',e',B',I'\}}, \sigma_{\{s',e',B',I'\}})$  if  $s' \models B'$ ;
          for each  $\sigma_{\{s,s',e',B,i\}}$  such that  $b(\sigma_{\{s,e,B,I\}}, \sigma_{\{s,s',e',B,i\}})$  do
             $s'' := \text{revise}(s', o);$ 
            create  $\sigma_{\{s',s'',e',B',i+1\}}$  for each  $s'' \models B'$ ;
            set  $R(\sigma_{\{s,s',e',B,i\}}, \sigma_{\{s',s'',e',B',i+1\}})$ ;
            set  $i(\sigma_{\{s',s'',e',B',i+1\}}) = \text{false};$ 
            set  $b(\sigma_{\{s',e',B,I\}}, \sigma_{\{s',s'',e',B',i+1\}})$  for each  $s'' \neq s', s'' \models B'$ ;
           $e' := \text{generate-event}(s', \text{attempt } a);$     %  $\epsilon$  if no event
          if  $e' \neq \epsilon$ 
            for each  $\iota \in \text{options}(\text{trigger-plans}(e', B'))$  do
               $I' := \text{drop}(B', \text{update-intention}(I)) \cup \{\iota\};$ 
              create  $\sigma_{\{s',e',B',I'\}}$  and add to  $\Sigma_{i+1}$ ;
              set  $R(\sigma_{\{s,e,B,I\}}^\pi, \sigma_{\{s',e',B',I'\}})$ ;
              set  $i(\sigma_{\{s',e',B',I'\}}) = \text{false};$     % unexpected events are unintended
              set  $b(\sigma_{\{s',e',B',I'\}}, \sigma_{\{s',e',B',I'\}})$  if  $s' \models B'$ ;
              for each  $\sigma_{\{s,s',e',B,i\}}$  such that  $b(\sigma_{\{s,e,B,I\}}, \sigma_{\{s,s',e',B,i\}})$  do
                 $s'' := \text{revise}(s', o);$ 
                create  $\sigma_{\{s',s'',e',B',i+1\}}$  for each  $s'' \models B'$ ;
                set  $R(\sigma_{\{s,s',e',B,i\}}, \sigma_{\{s',s'',e',B',i+1\}})$ ;
                set  $i(\sigma_{\{s',s'',e',B',i+1\}}) = \text{false};$ 
                set  $b(\sigma_{\{s',e',B',I'\}}, \sigma_{\{s',s'',e',B',i+1\}})$  for each  $s'' \neq s', s'' \models B'$ ;
            set  $a(\sigma_{\{s,e,B,I\}}^\pi, \sigma_{\{s',e',B',I'\}})$  if  $I = I'$  for each  $\sigma_{\{s,e,B,I\}}^\pi, \sigma_{\{s',e',B',I'\}} \in \Sigma_{i+1}$ 
              that are both  $R$ -successors of  $a$ -related states;
           $i := i + 1$ 
  until no new states are generated

```