AgentSpeak(L): BDI Agents speak out in a logical computable language

Anand S. Rao

Australian Artificial Intelligence Institute
Level 6, 171 La Trobe Street, Melbourne
Victoria 3000, Australia
Email: anand@aaii.oz.au

Abstract. Belief-Desire-Intention (BDI) agents have been investigated by many researchers from both a theoretical specification perspective and a practical design perspective. However, there still remains a large gap between theory and practice. The main reason for this has been the complexity of theorem-proving or model-checking in these expressive specification logics. Hence, the implemented BDI systems have tended to use the three major attitudes as data structures, rather than as modal operators. In this paper, we provide an alternative formalization of BDI agents by providing an operational and proof-theoretic semantics of a language AgentSpeak(L). This language can be viewed as an abstraction of one of the implemented BDI systems (i.e., PRS) and allows agent programs to be written and interpreted in a manner similar to that of horn-clause logic programs. We show how to perform derivations in this logic using a simple example. These derivations can then be used to prove the properties satisfied by BDI agents.

1 Introduction

The specification, design, verification, and applications of a particular type of agents, called BDI agents, have received a great deal of attention in recent years. BDI agents are systems that are situated in a changing environment, receive continuous perceptual input, and take actions to affect their environment, all based on their internal mental state. Beliefs, desires, and intentions are the three primary mental attitudes and they capture the informational, motivational, and decision components of an agent, respectively. In addition to these attitudes, other notions such as commitments, capabilities, know-how, etc. have been investigated. Sophisticated, multi-modal, temporal, action, and dynamic logics have been used to formalize some of these notions [2, 6, 8, 13, 18, 20, 21]. The complexity of theorem-proving and the completeness of these logics have not been clear [12, 23].

On the other hand, there are a number of implementations of BDI agents [1, 3, 10, 17] that are being used successfully in critical application domains. These implementations have made a number of simplifying assumptions and modelled the attitudes of beliefs, desires, and intentions as data structures. Also, user written plans or programs speed up the computation in these systems. The complexity of the code written for these systems and the simplifying assumptions made by them have meant that the implemented systems have lacked a strong theoretical underpinning. The specification logics have

shed very little light on the practical problems. As a result the two streams of work seem to be diverging.

Our earlier attempt to bridge this gap between theory and practice has concentrated on providing an abstract BDI architecture [14], that serves both as an idealization of an implemented system and also as a vehicle for investigating certain theoretical properties. Due to its abstraction this work was unable to show a one-to-one correspondence between the model theory, proof theory, and the abstract interpreter. The holy grail of BDI agent research is to show such a one-to-one correspondence with a reasonably useful and expressive language.

This paper makes another attempt at specifying such a logical language. Unlike some of the previous attempts, it takes as its starting point one of the implemented systems and formalizes its operational semantics. The implemented system being considered is the Procedural Reasoning System (PRS) [5] and its more recent incarnation, the Distributed Multi-Agent Reasoning System (dMARS). The language AgentSpeak(L) can be viewed as a simplified, textual language of PRS or dMARS. The language and its operational semantics are similar to the implemented system in their essential details. The implemented system has more language constructs to make the task of agent programming easier.

AgentSpeak(L) is a programming language based on a restricted first-order language with events and actions. The behaviour of the agent (i.e., its interaction with the environment) is dictated by the programs written in AgentSpeak(L). The beliefs, desires, and intentions of the agent are not explicitly represented as modal formulas. Instead, we as designers can ascribe these notions to agents written in AgentSpeak(L). The current state of the agent, which is a model of itself, its environment, and other agents, can be viewed as its current belief state; states which the agent wants to bring about based on its external or internal stimuli can be viewed as desires; and the adoption of programs to satisfy such stimuli can be viewed as intentions. This shift in perspective of taking a simple specification language as the execution model of an agent and then ascribing the mental attitudes of beliefs, desires, and intentions, from an external viewpoint is likely to have a better chance of unifying theory and practice.

In Section 2 we discuss the agent language AgentSpeak(L). The specification language consists of a set of base beliefs (or facts in the logic programming sense) and a set of plans. Plans are context-sensitive, event-invoked recipes that allow hierarchical decomposition of goals as well as the execution of actions. Although syntactically plans look similar to the definite clauses of logic programming languages, they are quite different in their behaviour.

Section 3 formalizes the operational semantics of AgentSpeak(L). At run-time an agent can be viewed as consisting of a set of beliefs, a set of plans, a set of intentions, a set of events, a set of actions, and a set of selection functions. The selection of plans, their adoption as intentions, and the execution of these intentions are described formally in this section. An interpreter for AgentSpeak(L) is given and a simple example is used to illustrate some of the definitions and the operational semantics of the language.

In Section 4, we provide the proof theory of the language. The proof theory is given as a labeled transition system. Proof rules define the transition of the agent from one configuration to the next. These transitions have a direct relationship to the operational se-

mantics of the language and hence help to establish the strong correspondence between the AgentSpeak(L) interpreter and its proof theory.

The primary contribution of this work is in opening up an alternative, restricted, first-order characterization of BDI agents. We hope that the operational and proof-theoretic semantics of AgentSpeak(L) will stimulate research in both the pragmatic and theoretical aspects of BDI agents.

2 Agent Programs

In this section, we introduce the language for writing agent programs. The *alphabet* of the formal language consists of variables, constants, function symbols, predicate symbols, action symbols, connectives, quantifiers, and punctuation symbols. Apart from first-order connectives, we also use ! (for achievement), ? (for test), ; (for sequencing), and $\leftarrow$ (for implication)[1]. Standard first-order definitions of terms, first-order formulas, closed formulas, and free and bound occurrences of variables are used.

Definition 1. If b is a predicate symbol, and t_1,...,t_n are terms then $b(t_1,...,t_n)$ or $b(\mathbf{t})$ is a belief atom. If $b(\mathbf{t})$ and $c(\mathbf{s})$ are belief atoms, $b(\mathbf{t}) \wedge c(\mathbf{s})$, and $\neg b(\mathbf{t})$ are *beliefs*. A belief atom or its negation will be referred to as a *belief literal*. A ground belief atom will be called a *base belief*.

For example, let us consider a traffic-world simulation, where there are four adjacent lanes and cars can appear in any lane and move in the same lane from north to south. Waste paper can appear on any of the lanes and a robot has to pick up the waste paper and place it in the bin. While doing this the robot must not be in the same lane as the car, as it runs the risk of getting run over by the car. Consider that we are writing agent programs for such a robot.

The beliefs of such an agent represent the configuration of the lanes and the locations of the robot, cars, waste, and the bin (i.e., `adjacent(X,Y)`, `location(robot, X)`, `location(car, X)`, etc.). The base beliefs of such an agent are ground instances of belief atoms (i.e., `adjacent(a,b)`, `location(robot, a)`, etc.).

A goal[2] is a state of the system which the agent wants to bring about. We consider two types of goals: an *achievement goal* and a *test goal*. An achievement goal, written as $!g(\mathbf{t})$ states that the agent wants to achieve a state where $g(\mathbf{t})$ is a true belief. A test goal, written as $?g(\mathbf{t})$ states that the agent wants to test if the formula $g(\mathbf{t})$ is a true belief or not. In our example, clearing the waste on a particular lane can be stated as an achievement goal, i.e., `!cleared(b)`, and seeing if the car is in a particular lane can be stated as a test goal, i.e., `?location(car, b)`.

Definition 2. If g is a predicate symbol, and t_1,...,t_n are terms then $!g(t_1,...,t_n)$ (or $!g(\mathbf{t})$) and $?g(t_1,...,t_n)$ (or $?g(\mathbf{t})$) are *goals*.

[1] In the agent programs we use `&` for $\wedge$, `not` for $\neg$, `<-` for $\leftarrow$. Also, like PROLOG, we require that all negations be ground when evaluated. We use the convention that variables are written in upper-case and constants in lower-case.

[2] In this paper, we discuss only goals, and not desires. Goals can be viewed as adopted desires.

When an agent acquires a new goal or notices a change in its environment, it may trigger additions or deletions to its goals or beliefs. We refer to these events as *triggering events*. We consider the addition/deletion of beliefs/goals as the four triggering events. Addition is denoted by the operator $+$ and deletion is denoted by the operator $-$. In our example, noticing the waste in a certain lane `X`, written as $+$`location(waste, X)` or acquiring the goal to clear the lane `X`, written as $+!$`cleared(X)` are example of two triggering events.

Definition 3. If $b(\mathbf{t})$ is a belief atom, $!g(\mathbf{t})$ and $?g(\mathbf{t})$ are goals, then $+b(\mathbf{t})$, $-b(\mathbf{t})+!g(\mathbf{t})$, $+?g(\mathbf{t})$, $-!g(\mathbf{t})$, $-?g(\mathbf{t})$ are *triggering events*.

The purpose of an agent is to observe the environment, and based on its observation and its goals, execute certain actions. These actions may change the state of the environment. For example, if `move` is an action symbol, the robot moving from lane `X` to lane `Y`, written as `move(X,Y)`, is an action. This action results in an environmental state where the robot is in lane `Y` and is no longer in lane `X`.

Definition 4. If a is an action symbol and t_1,...,t_n are first-order terms, then $a(t_1,...,t_n)$ or $a(\mathbf{t})$ is an action.

An agent has plans which specify the means by which an agent should satisfy an end. A plan consists of a head and a body. The head of a plan consists of a triggering event and a context, separated by a ":". The triggering event specifies why the plan was triggered, i.e., the addition or deletion of a belief or goal. The context of a plan specifies those beliefs that should hold in the agent's set of base beliefs, when the plan is triggered. The body of a plan is a sequence of goals or actions. It specifies the goals the agent should achieve or test, and the actions the agent should execute. For example, we want to write a plan that gets triggered when some waste appears on a particular lane. If the robot is in the same lane as the waste, it will perform the action of picking up the waste, followed by achieving the goal of reaching the bin location, followed by performing the primitive action of putting it in the bin. This plan can be written as:

```
+location(waste,X):location(robot,X) &
                   location(bin,Y)
                   <- pick(waste);
                      !location(robot,Y);
                      drop(waste).            (P1)
```

Consider the plan for the robot to change locations. If it has acquired the goal to move to a location `X` and it is already in location `X`, it does not have to do anything and hence the body is `true`. If the context is such that it is not at the desired location then it needs to find an adjacent lane with no cars in it, and then move to that lane.

```
+!location(robot,X):location(robot,X) <- true.  (P2)

+!location(robot,X):location(robot,Y) &
                     (not (X = Y)) &
```

```
adjacent(Y,Z) &
(not (location(car, Z)))
        <- move(Y,Z);
           +!location(robot,X). (P3)
```

More formally, we have the following definition of plans.

Definition 5. If e is a triggering event, b_1,...,b_m are belief literals, and h_1,...,h_n are goals or actions then e:$b_1 \wedge \ldots \wedge b_m \leftarrow h_1$;...;$h_n$ is a *plan*. The expression to the left of the arrow is referred to as the *head* of the plan and the expression to the right of the arrow is referred to as the *body* of the plan. The expression to the right of the colon in the head of a plan is referred to as the *context*. For convenience, we shall rewrite an empty body with the expression $true$.

With this we complete the specification of an agent. In summary, a designer specifies an agent by writing a set of base beliefs and a set of plans. This is similar to a logic programming specification of facts and rules. However, some of the major differences between a logic program and an agent program are as follows:

- In a pure logic program there is no difference between a goal in the body of a rule and the head of a rule. In an agent program the head consists of a triggering event, rather than a goal. This allows for a more expressive invocation of plans by allowing both data-directed (using addition/deletion of beliefs) and goal-directed (using addition/deletion of goals) invocations.
- Rules in a pure logic program are not context-sensitive as plans.
- Rules execute successfully returning a binding for unbound variables; however, execution of plans generates a sequence of ground actions that affect the environment.
- While a goal is being queried the execution of that query cannot be interrupted in a logic program. However, the plans in an agent program can be interrupted.

3 Operational Semantics

Informally, an agent consists of a set of base beliefs, B, a set of plans, P, a set of events, E, a set of actions, A, a set of intentions, I, and three selection functions, $\mathcal{S}_\mathcal{E}$, $\mathcal{S}_\mathcal{O}$, and $\mathcal{S}_\mathcal{I}$. When the agent notices a change in the environment or an external user has asked the system to adopt a goal, an appropriate triggering event is generated. These events correspond to external events. An agent can also generate internal events. Events, internal or external, are asynchronously added to the set of events E. The selection function $\mathcal{S}_\mathcal{E}$ selects an event to process from the set of events E. This event is removed from E and is used to unify with the triggering events of the plans in the set P. The plans whose triggering events so unify are called relevant plans and the unifier is called the relevant unifier. Next, the relevant unifier is applied to the context condition and a correct answer substitution is obtained for the context, such that the context is a logical consequence of the set of base beliefs, B. Such plans are called applicable plans or options and the composition of the relevant unifier with the correct answer substitution is called the applicable unifier.

For each event there may be many applicable plans or options. The selection function $\mathcal{S}_{\mathcal{O}}$ chooses one of these plans. Applying the applicable unifier to the chosen option yields the intended means of responding to the triggering event. Each intention is a stack of partially instantiated plans or intention frames. In the case of an external event the intended means is used to create a new intention, which is added to the set of intentions I. In the case of an internal event to add a goal the intended means is pushed on top of an existing intention that triggered the internal event.

Next, the selection function $\mathcal{S}_{\mathcal{I}}$ selects an intention to execute. When the agent executes an intention, it executes the first goal or action of the body of the top of the intention. Executing an achievement goal is equivalent to generating an internal event to add the goal to the current intention. Executing a test goal is equivalent to finding a substitution for the goal which makes it a logical consequence of the base beliefs. If such a substitution is found the test goal is removed from the body of the top of the intention and the substitution is applied to the rest of the body of the top of the intention. Executing an action results in the action being added to the set of actions, A, and it being removed from the body of the top of the intention.

The agent now goes to the set of events, E, and the whole cycle continues until there are no events in E or there is no runnable intention. Now we formalize the above process[3].

The state of an agent at any instant of time can be formally defined as follows:

Definition 6. An *agent* is given by a tuple <E,B,P,I,A,$\mathcal{S}_{\mathcal{E}}$,$\mathcal{S}_{\mathcal{O}}$,$\mathcal{S}_{\mathcal{I}}$>, where E is a set of events, B is a set of base beliefs, P is a set of plans, I is a set of intentions, and A is a set of actions. The selection function $\mathcal{S}_{\mathcal{E}}$ selects an event from the set E; the selection function $\mathcal{S}_{\mathcal{O}}$ selects an option or an applicable plan (see Definition 10) from a set of applicable plans; and $\mathcal{S}_{\mathcal{I}}$ selects an intention from the set I.

The sets B, P, and A are as defined before and are relatively straightforward. Here we describe the sets E and I.

Definition 7. The set I is a set of intentions. Each *intention* is a stack of *partially instantiated plans*, i.e., plans where some of the variables have been instantiated. An intention is denoted by $[p_1 \ddagger \ldots \ddagger p_z]$, where p_1 is the bottom of the stack and p_z is the top of the stack. The elements of the stack are delimited by $\ddagger$. For convenience, we shall refer to the intention `[+!true:true <- true]` as the *true intention* and denote it by T.

Definition 8. The set E consists of events. Each event is a tuple $<e, i>$, where e is a triggering event and i is an intention. If the intention i is the *true intention*, the event is called an *external event*; otherwise it is an *internal event*.

Now we can formally define the notion of relevant and applicable plans and unifiers. As we saw earlier, a triggering event d from the set of events, E, is to be unified with the triggering event of all the plans in the set P. The *most general unifier (mgu)* that unifies these two events is called the relevant unifier. The intention i could be wither the true intention or an existing intention which triggered this event. More formally,

[3] The reader can refer to the Appendix for some basic definitions from first-order logic and horn clause logic.

Definition 9. Let $\mathcal{S}_{\mathcal{E}}$(E) = ϵ = $< d, i >$ and let p be $e : b_1 \wedge \ldots \wedge b_m \leftarrow h_1; \ldots; h_n$. The plan p is a *relevant plan* with respect to an event ϵ iff there exists a most general unifier σ such that $d\sigma = e\sigma$. σ is called the *relevant unifier* for ϵ.

For example, assume that the triggering event of the event selected from E is

```
+!location(robot,b).
```

The two plans `P2` and `P3` are relevant for this event with the relevant unifier being {`X/b`}.

A relevant plan is also applicable if there exists a substitution which, when composed with the relevant unifier and applied to the context, is a logical consequence of the set of base beliefs B. In other words, the context condition of a relevant plan needs to be a logical consequence of B, for it to be an applicable plan. More formally,

Definition 10. A plan p, denoted by $e : b_1 \wedge \ldots \wedge b_m \leftarrow h_1; \ldots; h_n$ is an *applicable plan* with respect to an event ϵ iff there exists a relevant unifier σ for ϵ and there exists a substitution θ such that $\forall(b_1 \wedge ... \wedge b_m)\sigma\theta$ is a logical consequence of B. The composition $\sigma\theta$ is referred to as the *applicable unifier* for ϵ and θ is referred to as the *correct answer substitution.*

Continuing with the same example, consider that the set of base beliefs is given by

```
adjacent(a,b).
adjacent(b,c).
adjacent(c,d).
location(robot,a).
location(waste,b).
location(bin,d).
```

The applicable unifier is {`X/b, Y/a, Z/b`} and only plan `P3` is applicable.

Depending on the type of the event (i.e., internal or external), the intention will be different. In the case of external events, the intended means is obtained by first selecting an applicable plan for that event and then applying the applicable unifier to the body of the plan. This intended means is used to create a new intention which is added to the set of intentions I.

Definition 11. Let $\mathcal{S}_{\mathcal{O}}(O_\epsilon)$ = p, where O_ϵ is the set of all applicable plans or options for the event ϵ = $< d, i >$ and p is $e : b_1 \wedge \ldots \wedge b_m \leftarrow h_1; \ldots; h_n$. The plan p is *intended* with respect to an event ϵ, where i is the true intention iff there exists an applicable unifier σ such that $[+!true : true \leftarrow true \ddagger (e : b_1 \wedge \ldots \wedge b_m \leftarrow h_1; \ldots; h_n)\sigma] \in$ I.

In our example, the only applicable plan `P3` will be intended with the intention I now being

```
[+!location(robot,b): location(robot,a) &
                      not(b = a) &
                      adjacent(a, b) &
                      not(location(car,b)) <-
                           move(a,b);
                           +!location(robot,b)].
```

In the case of internal events the intended means for the achievement goal is pushed on top of the existing intention that triggered the internal event.

Definition 12. Let $\mathcal{S}_{\mathcal{O}}(O_\epsilon) = p$, where O_ϵ is the set of all applicable plans or options for the event $\epsilon = < d, [p_1 \ddagger \ldots \ddagger f : c_1 \wedge \ldots \wedge c_y \leftarrow !g(\mathbf{t}); h_2; \ldots; h_n] >$, and p is $+!g(\mathbf{s})$: $b_1 \wedge \ldots \wedge b_m \leftarrow k_1; \ldots; k_j$. The plan p is *intended* with respect to an event ϵ iff there exists an applicable unifier σ such that $[p_1 \ddagger \ldots \ddagger f : c_1 \wedge \ldots \wedge c_y \leftarrow !g(\mathbf{t}); h_2; \ldots; h_n \ddagger (+!g(\mathbf{s}) : b_1 \wedge \ldots \wedge b_m)\sigma \leftarrow (k_1; \ldots; k_j)\sigma; (h_2; \ldots; h_n)\sigma] \in$ I.

The above definition is very similar to SLD-resolution of logic programming languages. However, the primary difference between the two is that the goal g is called indirectly by generating an event. This gives the agent better real-time control as it can change its focus of attention, if needed, by adopting and executing a different intention. Thus, one can view agent programs as multi-threaded interruptible logic programming clauses.

When an intention is selected and executed, the first formula in the body of the top of the intention can be: (a) an achievement goal; (b) a test goal; or (c) an action; or (d) $true$. In the case of an achievement goal the system executes it by generating an event; in the case of a test goal it looks for a mgu that will unify the goal with the set of base beliefs of the agent, and if such an mgu exists it applies it to the rest of the means; in the case of an action the system adds it to the set of actions A; and in the last case the top of the intention and the achievement goal that was satisfied are removed and the substitution is applied to the rest of the body of that intention.

Definition 13. Let $\mathcal{S}_{\mathcal{I}}(I) = i$, where i is $[p_1 \ddagger \ldots \ddagger f : c_1 \wedge \ldots \wedge c_y \leftarrow !g(\mathbf{t}); h_2; \ldots; h_n]$. The intention i is said to have been *executed* iff $< +!g(\mathbf{t}), i > \in$ E.

Definition 14. Let $\mathcal{S}_{\mathcal{I}}(I) = i$, where i is $[p_1 \ddagger \ldots \ddagger f : c_1 \wedge \ldots \wedge c_y \leftarrow ?g(\mathbf{t}); h_2; \ldots; h_n]$. The intention i is said to have been *executed* iff there exists a substitution θ such that $\forall g(\mathbf{t})\theta$ is a logical consequence of B and i is replaced by $[p_1 \ddagger \ldots \ddagger (f : c_1 \wedge \ldots \wedge c_y)\theta \leftarrow h_2\theta; \ldots; h_n\theta]$.

Definition 15. Let $\mathcal{S}_{\mathcal{I}}(I) = i$, where i is $[p_1 \ddagger \ldots \ddagger f : c_1 \wedge \ldots \wedge c_y \leftarrow a(\mathbf{t}); h_2; \ldots; h_n]$. The intention i is said to have been *executed* iff $a(\mathbf{t}) \in A$, and i is replaced by $[p_1 \ddagger \ldots \ddagger f : c_1 \wedge \ldots \wedge c_y \leftarrow h_2; \ldots; h_n]$.

Definition 16. Let $\mathcal{S}_{\mathcal{I}}(I) = i$, where i is $[p_1 \ddagger \ldots \ddagger p_{z-1} \ddagger !g(\mathbf{t}) : c_1 \wedge \ldots \wedge c_y \leftarrow true]$, where p_{z-1} is $e : b_1 \wedge \ldots \wedge b_x \leftarrow !g(\mathbf{s}); h_2; \ldots; h_n$. The intention i is said to have been *executed* iff there exists a substitution θ such that g(**t**)θ = g(**s**)θ and i is replaced by $[p_1 \ddagger \ldots \ddagger p_{z-1} \ddagger (e : b_1 \wedge \ldots \wedge b_x)\theta \leftarrow (h_2; \ldots; h_n)\theta]$.

Continuing our example, we would execute I and by Definition 15 we would add {`move(a,b)`} to A and change I to be as follows:

```
[+!location(robot,b): location(robot,a) &
                      not(b = a) &
                      adjacent(a, b) &
                      not(location(car,b)) <-
                         +!location(robot,b)].
```

In the next iteration, after the robot moves from `a` to `b` the environment will send the agent a belief update event to change the location of the robot to `b`. This will result in the belief `location(robot,b)` being added to the set B and the event `+location (robot,b)` being added to the set of events, E. As there are no relevant plans for this the system will choose the above intention to execute. Executing this will result in an intention add event being generated and added to the set of events, E; in other words E is {`<+!location (robot,b),i>`}, where `i` is the same intention as before. By Definition 12 the relevant plan in this case is `P1` with the relevant unifier {`X/b`}. This plan is also applicable and the applicable unifier is the same. As the body of this plan is `true`, the intention is satisfied and the set of events is empty. This terminates the execution until the next event is added into the set E.

From the above definitions and description of the operational semantics of the language AgentSpeak(L) we can write an interpreter for AgentSpeak(L). Figure 1 describes such an interpreter. We use the function top to return the top of an intention stack; the function $head$ to return the head of an intended plan; the function $body$ to return the body of an intended plan. In addition, the functions $first$ and $rest$ are used to return the first element of a sequence, and all but the first element of a sequence. The function *push* takes an intention frame and an intention (i.e., stack of intention frames) and pushes the intention frame on to the top of the intention. The function *pop* takes an intention as an argument and returns the top of the intention.

4 Proof Theory

So far we have presented the operational semantics of AgentSpeak(L). Now we briefly discuss its proof theory based on labeled transition systems.

Definition 17. A *BDI transition system* is a pair $\langle \Gamma, \vdash \rangle$ consisting of:

- A set Γ of *BDI configurations*; and
- A binary transition relation $\vdash \subseteq \Gamma \times \Gamma$.

We define a BDI configuration as follows:

Definition 18. A *BDI configuration* is a tuple of $\langle E_i, B_i, I_i, A_i, i \rangle$, where $E_i \subseteq E$, $B_i \subseteq B$, $I_i \subseteq I$, $A_i \subseteq A$, and i is the label of the transition.

Note that we have not taken the set of plans, P, in the configuration as we have assumed it to be constant. Also, we do not explicitly keep track of goals as they appear as intentions when adopted by the agent. Now we can write transition rules that take an agent from one configuration to its subsequent configuration.

The following proof rule *IntendEnd* gives the transition for intending a plan at the top level. It states how the agent's set of intentions I changes in response to an external event that has been chosen (by the $\mathcal{S}_\mathcal{E}$ function) to be processed.

$$(IntendEnd)\ \frac{< \{\ldots, < +!g(\mathbf{t}), T >, \ldots\}, B_i, I_i, A_i, i >}{< \{\ldots\}, B_i, I_i \cup \{[p\sigma\theta]\}, A_i, i+1 >}$$

Algorithm Interpreter()
`while` E $\neq \emptyset$ `do`
  ϵ = $< d, i >$ = $\mathcal{S}_{\mathcal{E}}$(E);
  E = E/ϵ;
  O_ϵ = $\{p\theta \mid \theta$ is an applicable unifier for event ϵ and plan $p\}$
  `if` external-event(ϵ) `then` I = I $\cup$ [$\mathcal{S}_{\mathcal{O}}(O_\epsilon)$];
  `else` push($\mathcal{S}_{\mathcal{O}}(O_\epsilon)\sigma$, i), where σ is an applicable unifier for ϵ;
  `case` $first(body(top(\mathcal{S}_{\mathcal{I}}(\text{I}))))$ = $true$
    x = pop($\mathcal{S}_{\mathcal{I}}$(I));
    push($head(top(\mathcal{S}_{\mathcal{I}}(\text{I})))\theta \leftarrow rest(body(top(\mathcal{S}_{\mathcal{I}}(\text{I}))))\theta$, $\mathcal{S}_{\mathcal{I}}$(I)),
    where θ is an mgu such that xθ = $head(top(\mathcal{S}_{\mathcal{I}}(\text{I})))\theta$;
  `case` $first(body(top(\mathcal{S}_{\mathcal{I}}(\text{I}))))$ = $!g(\mathbf{t})$
    E = E $\cup$ <+!g(**t**),$\mathcal{S}_{\mathcal{I}}$(I)>
  `case` $first(body(top(\mathcal{S}_{\mathcal{I}}(\text{I}))))$ = $?g(\mathbf{t})$
    pop($\mathcal{S}_{\mathcal{I}}$(I));
    push($head(top(\mathcal{S}_{\mathcal{I}}(\text{I})))\theta \leftarrow rest(body(top(\mathcal{S}_{\mathcal{I}}(\text{I}))))\theta$, $\mathcal{S}_{\mathcal{I}}$(I)),
    where θ is the correct answer substitution
  `case` $first(body(top(\mathcal{S}_{\mathcal{I}}(\text{I}))))$ = $a(\mathbf{t})$
    pop($\mathcal{S}_{\mathcal{I}}$(I));
    push($head(top(\mathcal{S}_{\mathcal{I}}(\text{I}))) \leftarrow rest(body(top(\mathcal{S}_{\mathcal{I}}(\text{I}))))$, $\mathcal{S}_{\mathcal{I}}$(I));
    A = A $\cup$ $\{a(\mathbf{t})\}$;
`endwhile`.

Fig. 1. Algorithm for the BDI Interpreter

where $p = +!g(\mathbf{s}) : b_1 \wedge \ldots \wedge b_m \leftarrow h_1; \ldots; h_n \in P$, $\mathcal{S}_{\mathcal{E}}$(E) = $< +!g(\mathbf{t}), T >$, g(**t**)σ = g(**s**)σ and $\forall$ ($b_1 \wedge ... \wedge b_m$)$\theta$ is a logical consequence of B_i.

The proof rule *IntendMeans* is similar to the previous proof rule, except that the applicable plan is pushed at the top of the intention given as the second argument of the chosen event. More formally we have,

$$(IntendMeans)\frac{< \{\ldots, < +!g(\mathbf{t}), j >, \ldots\}, B_i, \{\ldots, [p_1 \ddagger \ldots \ddagger p_z], \ldots\}, A_i, i >}{< \{\ldots\}, B_i, \{\ldots, [p_1 \ddagger \ldots \ddagger p_z \ddagger p\sigma\theta], \ldots\}, A_i, i+1 >}$$

where $p_z = f : c_1 \wedge \ldots \wedge c_y \leftarrow !g(\mathbf{t}); h_2; \ldots; h_n$, $p = +!g(\mathbf{s}) : b_1 \wedge \ldots \wedge b_m \leftarrow k_1; \ldots; k_x$, $\mathcal{S}_{\mathcal{E}}$(E) = $< +!g(\mathbf{t}), j >$, j is $[p_1 \ddagger \ldots \ddagger p_n] >$, g(**t**)$\sigma$ = g(**s**)σ and $\forall$ ($c_1 \wedge ... \wedge c_y$)$\theta$ is a logical consequence of B_i.

Next, we have four proof rules for execution. The four proof rules are based on the type of the goal or action that appears as the first literal of the body of the top of an intention chosen to be executed by the function $\mathcal{S}_{\mathcal{I}}$. We give the execution proof rule for achieve *ExecAch*, the other proof rules can be written analogously.

$$(ExecAch)\frac{< E_i, B_i, \{\ldots, [p_1 \ddagger \ldots \ddagger f : c_1 \wedge \ldots \wedge c_y \leftarrow !g(\mathbf{t}); h_2; \ldots; h_n], \ldots\}, A_i, i >}{< E_i \cup \{< +!g(\mathbf{t}), j >\}, B_i, \{\ldots, [p_1 \ddagger \ldots \ddagger p_z], \ldots\}, A_i, i+1 >}$$

where $\mathcal{S}_{\mathcal{I}}(I_i)$ = j = $[p_1 \ddagger \ldots \ddagger p_z]$ and $p_z = f : c_1 \wedge \ldots \wedge c_y \leftarrow !g(\mathbf{t}); h_2; \ldots; h_n]$.

Although we have given the proof rules only for additions of goals, similar proof rules apply for deletion of goals, and addition and deletion of beliefs.

With these proof rules one can formally define derivations and refutations. The definition of derivations is straightforward and is a sequence of transitions using the above proof rules.

Definition 19. A *BDI derivation* is a finite or infinite sequence of BDI configurations, i.e., $\gamma_0, \ldots, \gamma_i, \ldots$.

The notion of refutation in AgentSpeak(L) is with respect to a particular intention. In other words, the refutation for an intention starts when an intention is adopted and ends when the intention stack is empty. Thus, using the above proof rules we can formally prove certain behavioural properties, such as safety and liveness of agent systems, as was done elsewhere [15]. Furthermore, there is a one-to-one correspondence between the proof rules discussed in this section and the operational semantics discussed in the previous section. Such a correspondence has not been possible before, because the proof theory (usually based on multi-modal logics) has been far removed from the realities of the operational semantics.

In addition to the internal events considered in this paper (i.e., addition of intentions), one can extend the operational semantics and proof rules with respect to other internal events, such as deletion of intentions, and success and failure events for actions, plans, goals, and intentions.

The body of the plans considered in this paper includes only sequences of goals or actions. Other dynamic logic operators, such as non-deterministic or, parallel, and iteration, operators can be allowed in the body of plans. In addition, assertion and deletion of beliefs in plan bodies can also be included. Another useful feature of the implemented system dMARS is different post-conditions for successful and failure executions of plans. The operational semantics and proof rules can once again be modified to account for the above constructs.

5 Comparisons and Conclusion

A number of agent-oriented languages such as AGENT0 [17], PLACA (PLAnning Communicating Agents) [19], AgentSpeak [22], SLP [16, 4], and CONGOLOG [9] have been proposed in the literature.

AGENT0 and its successor PLACA can model beliefs, commitments, capabilities, and communications between agents. These attitudes are treated as data structures of an agent program. An interpreter that can execute such agent programs are described. However, the authors do not provide a formal proof theory or justify how the data structures capture the model-theoretic semantics of beliefs, commitments, and capabilities. In contrast, the work described here discusses the connections between the interpreter and a proof theory based on labeled transition systems.

SLP or Stream Logic Programming is based on reactive, guarded, horn clauses. A clause in SLP consists of a guard and a behaviour. The guard is further decomposed into an head and a boolean constraint. The boolean constraint is similar to our context. The head in SLP is an object and the body is a network of concurrent objects connected by

communication message slots. Behaviour is specified by object replacement. The execution model of SLP and AgentSpeak(L) are fundamentally different. The behaviour of an agent to a particular external stimuli is captured in a single intention, as a stack of committed sub-behaviours. This provides a global coherence absent in SLP. For example, consider an agent that wants to drop its intention because it no longer needs to achieve a given top-level goal. Killing such an intention would be much easier in AgentSpeak(L) than in SLP.

The semantics of CONGOLOG is based on situation calculus. Although it provides a richer set of actions than what has been discussed here, it is essentially a single intention (or single-threaded) system, unlike AgentSpeak(L). The language AgentSpeak [22] is an object-oriented analogue of AgentSpeak(L).

AgentSpeak(L) is a textual and simplified version of the language used to program the Procedural Reasoning System [3] and its successor dMARS. These implementations have been in use since the mid-1980s. Other agent-oriented systems, such as COSY [1], INTERRAP [10], and GRATE* [7], have been built based on the BDI architecture. The formal operational semantics given here could apply to some of these systems as well. However, a more thorough analysis of these systems and their relation to AgentSpeak(L) is beyond the scope of this paper.

Bridging the gap between theory and practice in the field of agents, and in particular the area of BDI agents, has proved elusive. In this paper, we provide an alternative approach by providing the operational semantics of AgentSpeak(L) which abstracts an implemented BDI system. The primary contribution of this work is in opening up an alternative, restricted, first-order characterization of BDI agents and showing a one-to-one correspondence between the operational and proof-theoretic semantics of such a characterization. We are confident that this approach is likely to be more fruitful than the previous approaches in bridging the gap between theory and practice in this area and will stimulate research in both the pragmatic and theoretical aspects of BDI agents.

Acknowledgements: The research reported in this paper was funded partly by the Generic Industry Research and Development Grant on *Distributed Real-Time Artificial Intelligence* and partly by the *Cooperative Research Centre for Intelligent Decision Systems*. The author wishes to thank Michael Georgeff and Lawrence Cavedon for their valuable input and comments on this paper.

Appendix

Definition 20. An atom of the form $s = t$, where s and t are terms is called an *equation*.

Definition 21. A *substitution* is a finite set $\{x_1/t_1,...,x_n/t_n\}$, where $x_1,...,x_n$ are distinct variables, and $t_1,...,t_n$ are terms such that $x_i \neq t_i$ for any i from 1..n.

Definition 22. The application of a substitution $\theta = \{x_1/t_1,...,x_n/t_n\}$ to a variable x_i, written as $x_i\theta$, yields t_i iff $x_i/t_i \in \theta$ and x_i otherwise. The application of θ to a term or formula is the term or formula obtained by simultaneously replacing every occurrence of x_i by t_i for all i from 1 to n.

Definition 23. Let $\theta = \{x_1/t_1,...,x_n/t_n\}$ and $\sigma = \{y_1/s_1,...,y_m/s_m\}$. The *composition* $\theta\sigma$ of θ and σ is the substitution obtained from the set: $\{x_1/t_1\sigma,...,x_n/t_n\sigma\} \cup \theta$ by removing all $x_i/t_i\sigma$ for which $x_i = t_i\sigma$ $(1 \leq i \leq n)$and removing those y_j/t_j for which $y_j \in \{x_1,...,x_n\}$ $(1 \leq j \leq m)$ [11].

Definition 24. A substitution σ is a *solution* or *unifier* of a set of equations $\{s_1 = t_1, ..., s_n = t_n\}$ iff $s_i\sigma = t_i\sigma$ for all $i = 1,...,n$. A substitution σ is *more general* than θ iff there is a substitution ω such that $\sigma\omega = \theta$. A *most general unifier (mgu)* of two terms (atoms) is a maximally general unifier of the terms.

References

1. B. Burmeister and K. Sundermeyer. Cooperative problem-solving guided by intentions and perception. In E. Werner and Y. Demazeau, editors, *Decentralized A.I. 3*, Amsterdam, The Netherlands, 1992. North Holland.
2. P. R. Cohen and H. J. Levesque. Intention is choice with commitment. *Artificial Intelligence*, 42(3), 1990.
3. M. P. Georgeff and A. L. Lansky. Procedural knowledge. In *Proceedings of the IEEE Special Issue on Knowledge Representation*, volume 74, pages 1383–1398, 1986.
4. M. M. Huntbach, N. R. Jennings, and G. A. Ringwood. How agents do it in stream logic programming. In *Proceedings of the International Conference on Multi-Agent Systems (ICMAS-95)*, San Francisco, USA, June, 1995.
5. F. F. Ingrand, M. P. Georgeff, and A. S. Rao. An architecture for real-time reasoning and system control. *IEEE Expert*, 7(6), 1992.
6. N. R. Jennings. On being responsible. In Y. Demazeau and E. Werner, editors, *Decentralized A.I. 3*. North Holland, Amsterdam, The Netherlands, 1992.
7. N. R. Jennings. Specification and implementation of belief, desire, joint-intention architecture for collaborative problem solving. *Journal of Intelligent and Cooperative Information Systems*, 2(3):289–318, 1993.
8. D. Kinny, M. Ljungberg, A. S. Rao, E. A. Sonenberg, G. Tidhar, and E. Werner. Planned team activity. In *Artificial Social Systems, Lecture Notes in Artificial Intelligence (LNAI-830)*, Amsterdam, Netherlands, 1994. Springer Verlag.
9. Y. Lesperance, H. J. Levesque, F. Lin, D. Marcu, R. Reiter, and R. B. Scherl. Foundations of a logical approach to agent programming. In *Working notes of the IJCAI-95 Workshop on Agent Theories, Architectures, and Languages*, Montreal, Canada, 1995.
10. J. P. Muller, M. Pischel, and M. Thiel. Modelling reactive behaviour in vertically layered agent architectures. In *Intelligent Agents: Theories, Architectures, and Languages. Lecture Notes in Artificial Intelligence LNAI 890*, Heidelberg, Germany, 1995. Springer Verlag.
11. U Nilsson. Abstract interpretations and abstract machines. Technical Report Dissertation No 265, Department of Computer and Information Science, Linkoping University, Linkoping, Sweden, 1992.
12. A. S. Rao. Decision procedures for propositional linear-time belief-desire-intention logics. In *Working notes of the IJCAI-95 Workshop on Agent Theories, Architectures, and Languages*, Montreal, Canada, 1995.
13. A. S. Rao and M. P. Georgeff. Modeling rational agents within a BDI-architecture. In J. Allen, R. Fikes, and E. Sandewall, editors, *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning*. Morgan Kaufmann Publishers, San Mateo, CA, 1991.

14. A. S. Rao and M. P. Georgeff. An abstract architecture for rational agents. In C. Rich, W. Swartout, and B. Nebel, editors, *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning*. Morgan Kaufmann Publishers, San Mateo, CA, 1992.
15. A. S. Rao and M. P. Georgeff. A model-theoretic approach to the verification of situated reasoning systems. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI-93)*, Chamberey, France, 1993.
16. G. A. Ringwood. A brief history of stream parallel logic programming. *Logic Programming Newsletter*, 7(2):2–4, 1994.
17. Y. Shoham. Agent-oriented programming. *Artificial Intelligence*, 60(1):51–92, 1993.
18. M. Singh and N. Asher. Towards a formal theory of intentions. In J. van Eijck, editor, *Logics in AI*, volume LNAI:478, pages 472–486. Springer Verlag, Amsterdam, Netherlands, 1990.
19. S. R. Thomas. The PLACA agent programming language. In *Intelligent Agents: Theories, Architectures, and Languages. Lecture Notes in Artificial Intelligence LNAI 890*, Amsterdam, Netherlands, 1995. Springer Verlag.
20. W. van der Hoek, B. van Linder, and J.-J. Ch. Meyer. A logic of capabilities. In *Proceedings of the Third International Symposium on the Logical Foundations of Computer Science (LFCS'94), Lecture Notes in Computer Science LNCS 813*. Springer Verlag, Heidelberg, Germany, 1994.
21. B. van Linder, W. van der Hoek, and J. J. Ch. Meyer. How to motivate your agents? In *Working notes of the IJCAI-95 Workshop on Agent Theories, Architectures, and Languages*, Montreal, Canada, 1995.
22. D. Weerasooriya, A. S. Rao, and K. Ramamohanarao. Design of a concurrent agent-oriented language. In *Intelligent Agents: Theories, Architectures, and Languages. Lecture Notes in Artificial Intelligence LNAI 890*, Amsterdam, Netherlands, 1995. Springer Verlag.
23. M. Wooldridge and M. Fisher. A decision procedure for a temporal belief logic. In *Proceedings of the First International Conference on Temporal Logic*, Bonn, Germany, 1994.

This article was processed using the LaTeX macro package with LLNCS style