Exploring the Future with Resource-Bounded Agents [*]
[INVITED CONTRIBUTION]

Michael Fisher (mfisher@liverpool.ac.uk)
Department of Computer Science, University of Liverpool, Liverpool L69 3BX, UK

Chiara Ghidini (ghidini@fbk.eu)
Fondazione Bruno Kessler - irst, 38050 Trento, Italy

Abstract. We here describe research into the formal specification and implementation of resource-bounded agents. In particular, we provide an overview of our work on incorporating resource limitations into executable agent specifications. In addition, we outline future directions, highlighting both their promise and their problems.

Keywords: agent-based systems, formal specification, resource-bounded agents

1. Introduction

Our overall programme of research is to "*provide a logical framework allowing us to describe both individual rational agents and complex organisational structures*". In this paper we describe one particular aspect of this work extending the above statement with "*.... and to incorporate bounds on the reasoning (about belief and time) the agent can carry out*". Further still, we aim to ensure that the logical descriptions we produce for resource-bounded agents are, in turn, executable. This aim to have executable logical specifications leads us to extend our statement of research one last time with ".... *and to ensure that these logical descriptions are directly executable*". Thus, in this paper we provide an overview of our previous and future work in this area, summarising results from the papers [10, 11, 12, 13].

2. Background

We see that, in the early 21st century, software is increasingly able to *migrate* through large physical/virtual distances, *spawn* new computations on a wide variety of platforms, and *access* vast amounts of information, etc. Sophisticated software applications are also expected to handle more and more tasks both *autonomously* and 'intelligently'. However, this autonomous and 'intelligent' behaviour is also allowing software to evolve *unexpected* autonomous behaviour. This presents developers with several problems: "how

[*] The authors would like to thank the organisers of the "Logics for Resource-Bounded Agents" workshop in 2007 for their invitation to present this work.

can we *understand* what such software is doing" and "how can we describe exactly what it *should* be doing"? It was partially in an attempt to answer such questions that the concept of an *autonomous agent* was introduced [22, 21].

2.1. AGENTS

An agent is an autonomous entity controlling not only its own state (as an *object* does), but also controlling its own actions and choices [22]. While the general concept of an agent has been very influential in the modelling and development of autonomous systems, a more specific categorisation is increasingly useful. The concept of a *rational agent* embodies an agent that is not only autonomous, but is in some sense 'intelligent'. Rational agents typically have their own goals and information and carry out reasoning in order to decide what to do next. By contrast with simpler agents, rational agents are expected to make decisions that are both 'rational' and explainable, employing *flexible autonomous action*. As such, the concept of a rational agent provides a key abstraction for describing and reasoning about such sophisticated systems [23]. Rational agents must be able to adapt their autonomous behaviour to cater for the dynamic nature of their dynamic environment, requirements, and knowledge (with any resource constraints). Traditionally, such agents involve pro-activeness, social ability, and deliberation [20].

2.2. FORMAL AGENT SPECIFICATIONS

The key problems concerning autonomous software, even when modelled in terms of rational agents, remain as

1. *programming* software to do what we require, and
2. *verifying* software to ensure the required behaviour will occur.

Formal logic helps with both of these, providing an unambiguous notation, in which the formal properties of logical descriptions are well understood. Importantly, logics can be designed to capture many agent varieties. In particular, choosing the appropriate logic provides a level of abstraction close to the key concepts of the software.

Unsurprisingly, there are many (logic-based) rational agent theories. The predominant approach is that of the *Belief-Desire-Intention (BDI)* theory [18]. Here, beliefs represent the information the agent has about itself and its environment, desires represent its long-term goals and intentions represent the goals that the agent is actively trying to achieve. Many theories, and indeed many practical systems, have been built upon these basic concepts. In providing a logical formalism for rational agents in general (including BDI agents), combinations of *modal* and *temporal* logics are typically used [5]. For example, modal logics (technically, **KD45** or **S5** modal logics) are often used to represent *knowledge* or *belief* aspects, while temporal (and sometimes dynamic) logics are used to represent the underlying dynamic nature of such agents. Motivational aspects, such as *desires*, *goals*, *intentions*, or *wishes* are also typically captured using modal logics (technically, **KD** modal logics).

Such logical combinations have been used in many works on the formal specification of rational agents, for example [19, 17, 20, 8].

2.3. Executable Specifications

Formal specifications are very useful for describing the (required) behaviour of rational agents, but how can we use these specifications? One approach is to use the specifications as a basis from which to develop agent implementations. Indeed, there are now very many agent programming languages [2]. But what is the link between the practical programming language and its semantics [3]? And how can we be sure that the program produced *does* actually implement the logical specification?

Our particular aim has been to bring logical agent theories and programming languages closer together, providing both clear program semantics and formal agent verification. One approach is to attempt to *directly execute* rational agent specifications — in this way, we can be more confident that the required behaviour is being exhibited by the system. Here, execution of a formula, φ, of a logic, L, is taken to mean constructing a model, $\mathcal{M}$, for φ, i.e. $\mathcal{M} \models_L \varphi$. This not only provides a close link between the theory and implementation, but also provides high-level, logical concepts within the programming language [6].

Our approach actually begins simply with agent specifications given in a *linear temporal logic*. Temporal logic [5] is an extension of classical logic with the notion of temporal order built in. With such logics we can describe many complex, dynamic properties, though they all reduce to describing what must be true *now*, what must be true *next*, and what is guaranteed to be true at *some* point in the future. This, seemingly simple, view gives us the flexibility to represent a wide range of computational activities [7].

In its basic form, temporal logic can be seen as an extension of classical logic, incorporating additional operators relating to temporal order[1]. These operators are typically:

'$\bigcirc$'	"in the next moment in time";
'$\square$'	"at every future moment";
'$\lozenge$'	"at some future moment".

[1] For simplicity, we use a discrete, linear model of time, isomorphic to the Natural Numbers.

These operators give us useful expressive power, and even with such a simple temporal logic as a basis, we are able to describe individual agents, not only their dynamic behaviour, but also how their knowledge/beliefs evolve and how their goals evolve. In a multi-agent context, this also allows us to describe the structure and evolution of communication, dynamic organisational structures, and how computation within each organisation evolves.

Our approach to executable agent specifications can be best described by starting with the basic temporal logic given above and then extending the logic executed in various ways. So, given a simple temporal formula, as outlined above, we execute this using the *Imperative Future* approach [1]:

– transform the temporal specification into a *normal form* [7];

– from the initial constraints, *forward chain* through the set of temporal rules constraining the *next* state of the agent; and

– constrain the execution by attempting to satisfy eventualities (aka goals), such as $\Diamond g$ (i.e. g eventually becomes true).

In addition, we require some strategy for handling *outstanding* eventualities (see below). The normal form [7] essentially categorises formulae into 3 varieties: *initial rules*, of the form **start** $\Rightarrow \varphi$, which indicate properties of the initial state; *step rules*, of the form $\psi \Rightarrow \bigcirc\varphi$, which indicate properties of the *next* state; and *sometime rules*, of the form $\psi \Rightarrow \Diamond\varphi$, which indicate properties of the future.

2.4. Execution Example

Imagine a 'car' agent which can *start_engine*, *turn* and *stop*, but can also break (*start_fails*) or start (*started*). In addition, it is able to broadcast information to other agents. A simple specification, already in our normal form, might be:

1. **start** $\Rightarrow \neg moving$
2. $\neg moving \Rightarrow \bigcirc start\_engine$
3. $start\_engine \Rightarrow \bigcirc(start\_fails \lor started)$
4. **true** $\Rightarrow \bigcirc\neg(start\_fails \land moving)$
5. $start\_engine \Rightarrow \Diamond moving$
6. $start\_fails \Rightarrow \bigcirc start\_fails$
7. $(started \land moving) \Rightarrow \bigcirc bcast($"here we go!"$)$

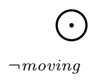
8. $bcast(\text{“here we go!”}) \Rightarrow \bigcirc(start\_fails \vee moving)$

Informally, the meaning of these formulae (or ‘rules’) is as follows.

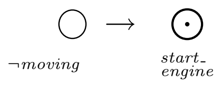
1. *moving* is false at the beginning of time.
2. If *moving* is false, *start_engine* will be true in the next moment.
3. Whenever *start_engine* is true, then either *start_fails* or *started* will be made true in the next moment in time.
4. At any future time, we cannot have both *start_fails* and *moving* being true.
5. Whenever *start_engine* is true, a commitment to eventually make *moving* true is given.
6. If *start_fails* is true, then it will also be true in the next state (note that this effectively means that, once *start_fails* is true, then it will always be true).
7. If both *started* and *moving* are true then, in he next moment, the agent broadcasts “here we go!”.
8. Finally, once it has broadcast this, then either *start_fails* or *moving* will again be made true in the next moment in time.

We will briefly show how execution is attempted under the *imperative future* view and so how model construction occurs[2].

Step 1: from rule 1, build an initial state in which *moving* is false.

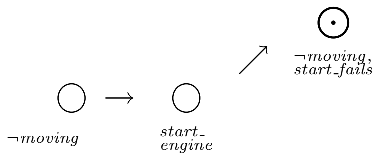


Step 2: from rule 2, build a next state in which *start_engine* is true.

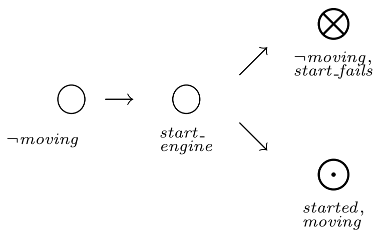


[2] In the following, ‘$\bigcirc$’ represents a constructed state, ‘$\odot$’ represents the current state, and ‘$\otimes$’ represents a state we have backtracked from.

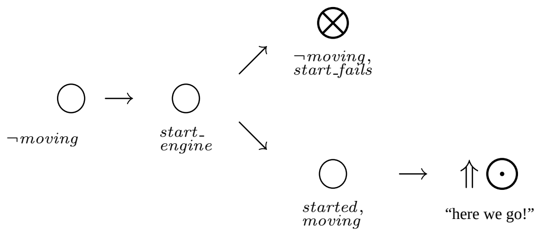
Step 3: rule 3 gives us a choice of making either *start_fails* or *started* true in the next state. We begin by exploring the '*start_fails*' possibility, and note that rule 5 means that we must also make *moving* true somewhere down this branch.



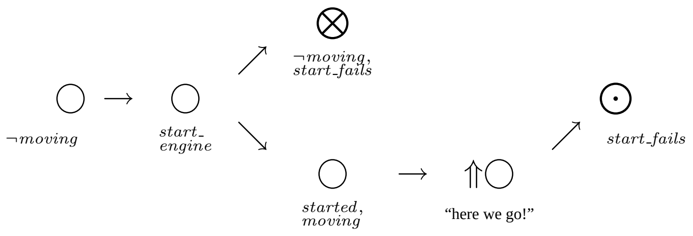
Step 4: after some further execution it is recognised that we actually cannot satisfy $\Diamond moving$ since *start_fails* is true along this branch, forcing *moving* to be false everywhere along this branch. So, we fail in exploring/executing this branch. We return to the other option in rule 3, namely that *started* becomes true in the next state. We can now also satisfy $\Diamond moving$ from rule 5, by making *moving* true here.



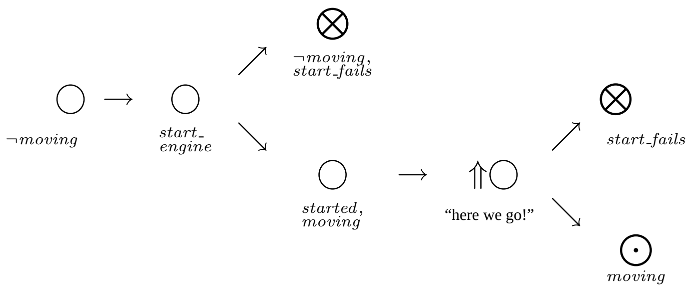
Step 5: since *started* and *moving* are both true, then rule 7 leads us to broadcast "here we go!". Other agents in the same environment can receive this message. Within the execution, however, we are now not allowed to backtrack past this broadcast event (effectively like a 'cut' in Prolog); this is graphically represented by '⇑'.



Step 6: now, rule 8 provides us with a choice and, again, if we take the *start_fails* option, this leads us to problems.



Step 7: however, if we take the *moving* choice then execution (and model construction) can continue.



And so on. Execution can continue in this way, can recognise that a previous state has re-occurred (and so can loop round), or can terminate (if all branches explored lead to contradictions). Various correctness results are available in the original papers; see [1], for example.

2.5. Deliberation

An important aspect in the above execution approach is the handling of eventualities of the form '$\diamond\varphi$'. These are effectively *goals* that the agent is trying to satisfy during execution. In the above example, there was just one eventuality, but there are usually several. However, it is likely that only a subset of these can be satisfied at any moment in time (for example, $\diamond p$ and $\diamond\neg p$). So, we must decide which eventualities to attempt at each moment in time. This turns out to be crucial both to the correctness of the execution mechanism, and to the agent's ability to *deliberate*.

Outstanding eventualities (i.e. those that must be satisfied, but have yet to be) are stored in a list. The eventualities are attempted in order from the beginning of the list. However, in between each state constructed, the list can be re-ordered. In order to retain correctness, this re-ordering must be *fair* [9].

This ensures that no eventuality will remain 'untried', i.e. that all eventualities are attempted sometimes.

As well as affecting correctness, this re-ordering of eventualities is central to deliberation. If we recall the standard view of deliberation within agent-based systems, it is essentially the process of deciding which goals the agent should tackle at present, and which approaches the agent should use to tackle these goals. Since goals in standard (for example, BDI) agents correspond to eventualities within our framework, then our re-ordering of eventualities can be seen as providing an ordered list of goals to tackle. In addition, since the re-ordering process can take into account various other aspects, then we can incorporate deciding what plans (if we have a notion of plans) to use in satisfying the goals.

To explain further, consider the simple example below.

2.6. Deliberation Example

Suppose we begin with the following list of goals/eventualities:

$$[\Diamond \mathsf{be\_famous}, \Diamond \mathsf{sleep}, \Diamond \mathsf{eat\_lunch}, \Diamond \mathsf{make\_lunch}] \,.$$

What shall the execution mechanism do?

– The standard approach would be to execute these oldest-first, say:

$$[\Diamond \mathsf{be\_famous}, \Diamond \mathsf{sleep}, \Diamond \mathsf{eat\_lunch}, \Diamond \mathsf{make\_lunch}] \,.$$

– However, during deliberation the agent might decide to re-order this list based on, for example, the *importance* of each goal to the agent. Thus, the agent might decide that becoming famous is the most important goal (and that sleeping is less important):

$$[\Diamond \mathsf{be\_famous}, \Diamond \mathsf{eat\_lunch}, \Diamond \mathsf{sleep}, \Diamond \mathsf{make\_lunch}] \,.$$

– Again, the agent might re-order the list, this time based on what it has available plans for. For example, the agent might not have any mechanism for becoming famous, so it moves this goal to lower priority (i.e. later in the list). However, it does have a way to make $\mathsf{eat\_lunch}$ true and so it moves this goal to the front of the list. While examining this goal, it notes that a sub-goal, in this case $\mathsf{make\_lunch}$, must be achieved before $\mathsf{eat\_lunch}$ and so it puts this sub-goal at the front of the list:

$$[\Diamond \mathsf{make\_lunch}, \Diamond \mathsf{eat\_lunch}, \Diamond \mathsf{sleep}, \Diamond \mathsf{be\_famous}] \,.$$

– And so on....

In this way, agents have a flexible and powerful mechanism for deciding between (and swapping between) goals/eventualities.

2.7. Belief Contexts

In addition to a basic temporal specification of an agent, providing the dynamic behaviour for that agent, we also incorporate a number of other formal components. The most relevant to our discussion here is that of agent *beliefs*. These represent the agent's view of itself, other agents and its environment. Such beliefs need not necessarily describe facts, they just capture the agent's understanding (and possible misunderstanding) of its world.

Formally, adding beliefs to the agent specification involves adding a belief operator $B_i\phi$, meaning that "agent i believes ϕ". This allows us to describe more sophisticated agents that are able to reason about their beliefs. For example, agent i is now able to represent and reason about its own beliefs:

$$(buy\_ticket \wedge B_i lucky) \Rightarrow B_i \Diamond lottery\_winner$$

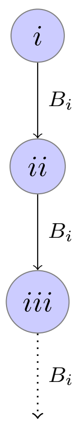
and is able to interpret external events/communications:

$$\begin{array}{rcl} in\_view(frog) & \Rightarrow & B_{Bart}\, can\_see(frog) \\ advertisement(donut) & \Rightarrow & B_{Homer}\, good(donut) \end{array}$$

While it is standard for such beliefs to be described using a **KD45** modal logic [16], we instead use a *multi-context logic of belief* [15, 14]. This can simulate a **KD45** modal logic, but allows us much greater control of the belief structures constructed. As we will see later, this is important when we want to restrict the depth of beliefs explored during execution.

The reason why multi-context logic facilitates the control of the belief structures is that it distributes nested belief into different and separated modules (also called *contexts*) which interact with each other. Let us restrict our discussion just to the simple case of a single agent i who is acting in a world, who has both beliefs about this world and beliefs about its own beliefs and it is able to reason about them. Multi-context logic represents agent i via a chain of contexts[3]:

[3] For a more detailed description of multi-context logic for belief, see [4] and [14].



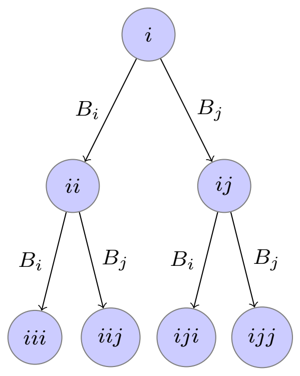
Level 0: basic beliefs of i;

Level 1: beliefs that i ascribes to itself;

Level 2: beliefs about beliefs that i ascribes to itself;

Level n: and so on...

The structure depicted above can be easily extended to represent an agent i having beliefs about the world, beliefs about its own beliefs and beliefs about other agents beliefs. For instance, we represent agent i having beliefs also about another agent j as follows



Level 0: basic beliefs of i;

Level 1: beliefs that i ascribes to itself or to j;

Level 3: and so on...

We can simulate all the typical properties of belief, and in particular those of modal **KD45** through constraints between adjacent pairs of contexts, connected via B_x labelled edges. For instance, we can simulate modal **K** by imposing the condition that

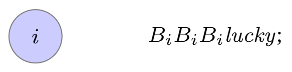
> *$B_i\phi$ is in a context at level n if, and only if, ϕ is in any $n+1$ level context reachable via a B_i labelled edge;*

or we can simulate the property $B_i\phi \Rightarrow B_iB_i\phi$ of modal **4** by imposing a constraint such as:

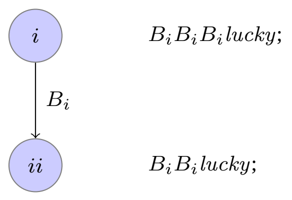
> *if $B_i\phi$ is in a context at level n then $B_i\phi$ is in any $n+1$ level context reachable via a B_i labelled edge.*

The precise logical formalisation of these intuitive constraints, together with its sound and complete axiomatisation, can be found in [14].

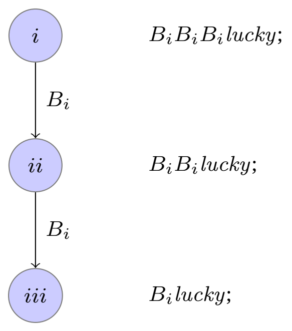
The linear or tree shaped structure is potentially of infinite depth. Infinite structures reflect the fact that, in using modal logic, people model i as an ideal agent able to express and reason about beliefs of arbitrary nesting. The peculiarity of multi-context logic is that we can also bind the length of the chain to a certain depth n, thus ensuring that there are no B_x reachable contexts beyond a certain level, still allowing i to express formulae of arbitrary nested belief. The main idea is to treat belief formulae as propositional atoms and to constrain the truth value of $B_i\phi$ in a context at level n with that of ϕ in all $n+1$ level contexts reachable via a B_i labelled edge, if any. Thus, assume that i is an agent able to manage only formulae with 2 nested beliefs. A formula $B_iB_iB_i lucky$ will be true in context i



if, and only if, $B_iB_i lucky$ will be true in context ii



if, and only if, $B_i lucky$ will be true at the ”bottom” context iii



Once level 2 is reached there are no further B_i labelled edges and the formula $B_i lucky$ is treated as a propositional atomic formula with an arbitrary truth value. In this work, to mimic the behaviour of modal **KD45**, we set the truth value of $B_i lucky$ (and all formulae $B_i\phi$) at this “bottom” context to **false**.

2.8. Adding Belief to the Temporal Structure

Now, agent specifications include beliefs. Thus, as we execute a specification, we must explore a belief structure in addition to a temporal structure. In addition we might also explore temporal sequences within belief worlds, obtaining more complex structures as the one in Figure 1.

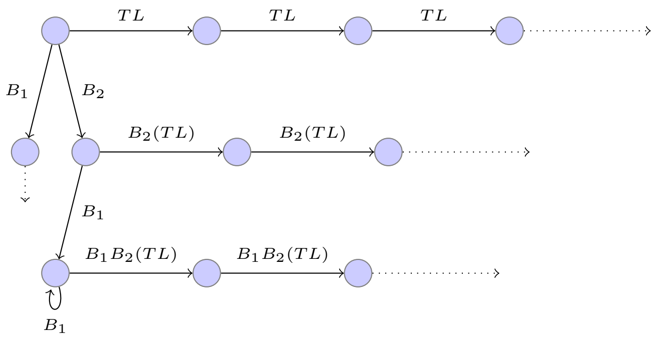


Figure 1. Model exploration during execution

Here, the basic temporal sequence (labelled by 'TL') is being constructed. However, at certain points, belief contexts (e.g. B_1 and B_2) must be explored in order to decide how to proceed. In addition, within these belief contexts, temporal execution itself can be simulated, e.g. $B_2(TL)$ and $B_1B_2(TL)$. The extension of the basic execution algorithm to incorporate belief is described in [10].

3. Resource-Bounded Agents

With unlimited time and space, we can let the agents carry out all the exploration necessary to build potentially large/deep structures like the one in Figure 1 and therefore take their time in constructing the execution. However, in more realistic scenarios we wish to *restrict* the exploration/reasoning that the agent can carry out. Below we consider two, complementary, approaches restricting belief exploration [10] and temporal exploration [12].

3.1. Bounds on Belief Exploration

If, in the original agent specification, beliefs are heavily nested, then reasoning about such beliefs tends to be very resource-intensive. It is hard enough to reason about beliefs or beliefs about beliefs, but reasoning about beliefs about beliefs about beliefs, and so on, is very difficult. So, we here adopt a form of resource-bounding which restricts the depth of nesting of beliefs that can be considered. This captures a form of resource-bounded reasoning and ensures that the agent does not spend all its time ‘thinking’.

As we said in the previous section, Figure 1 shows a sample ‘normal’ execution for an agent. Once we set a belief bound we cut off exploration below this in the hierarchy of belief contexts, as in Figure 2.

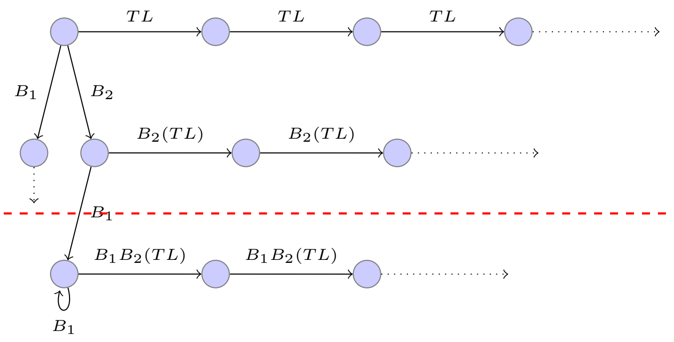


Figure 2. Exploration with a restricting belief bound

Three specific things to notice about the structures in this approach:

1. Once the depth bound k is reached there are effectively no further B_i labelled edges to explore. At this point $B_i\phi$ = $\neg B_i\neg\phi$ = **false**.

2. We allow *syntactic* control of the belief bound by means of “special” propositional constants of the form $belief\_bound(k)$. Therefore $belief\_bound(100)$ would allow quite a lot of reasoning, while $belief\_bound(1)$ would allow very little. On the negative side, this approach reduces the expressivity of the language to syntactically specify changes in the belief depth bound only among a finite set of values; on the positive side, this enables us to maintain the formalisation as simple as possible and to use the execution mechanism described in [10] in order to provide the prototype implementation.

3. The value of $belief\_bound(.)$ can change over time. The main idea is that in any temporal state of the basic temporal sequence ‘TL’, if $belief\_bound(k)$ is satisfied, then exploration of belief contexts is limited to depth k at that point. Clearly, we must impose the following properties on $belief\_bound(.)$ in order to ensure that the bound is always well-defined.

   a) In every state, there is at most one k such that $belief\_bound(k)$ is satisfied.

$$\Box \bigwedge_{k=0}^{m} \bigwedge_{h=0,h\neq k}^{m} \neg(belief\_bound(k) \wedge belief\_bound(h))$$

b) In every state, there is at least one k such that $\mathit{belief\_bound}(k)$ is satisfied.

$$\Box \bigvee_{k=0}^{m} \mathit{belief\_bound}(k)$$

Thus, for every specification, the above two properties must be established. A particular problem here is (b). This only states that there is *some* belief depth bound. In reality we would have further formulae specifying how the value of the bound relates to previous values. Pushing this idea forward we could also have agents able to reason about their own limits in a more sophisticated way, and able to decide about these limits. A discussion about these aspects of $\mathit{belief\_bound}(.)$ can be found in Section 4.

3.2. Bounds on Temporal Exploration

An obvious analogue of restricting reasoning about belief is to restrict the hypothetical temporal reasoning that is allowed [12]. This bounded temporal exploration allows us to restrict the diagram from Figure 2 still further, to give that in Figure 3.

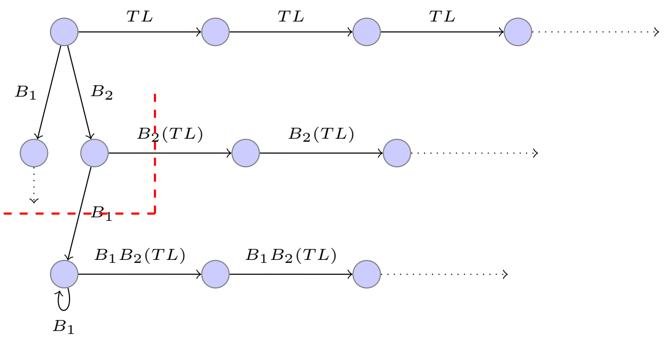


Figure 3. Restricting hypothetical temporal exploration in addition to belief exploration.

Thus, in addition to bounding the depth of nesting of belief contexts, as earlier, we can also bound the depth of nesting of simulated temporal states. We might do this as an alternative to belief bounds, or in combination with it. For example, in Figure 3, we restrict both the belief and temporal dimensions.

This restriction of temporal exploration works in a similar way to the belief bounding, limiting the number of temporal states that can be constructed in a hypothetical sequence (i.e. one within a belief context). This restriction can be achieved in a number of ways. The most obvious is to provide a dual to $\mathit{belief\_bound}(.)$, namely $\mathit{tl\_bound}(.)$ which provides a numerical bound on the number of temporal states that can be constructed. Again the bound will be selected from a fixed set of numbers. An alternative approach, explored in [12], is to use specific varieties of temporal operators with which we can describe finite sequences. For example, we might say that we are only allowed 5 nested '$\bigcirc$' operators before we reach the end of the possible temporal sequence.

This approach allows us to bound the temporal, in addition to the belief, exploration within agent execution.

3.3. EXAMPLE

We now look at an example, derived from that presented in [13, 11]. Consider two teams of agents, acting as football players, in the situation depicted in Figure 4.

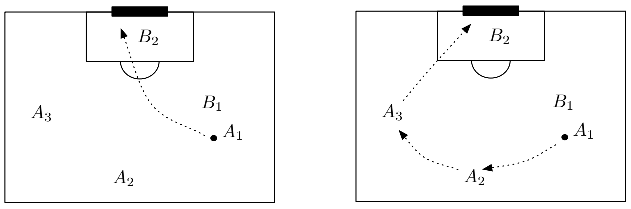


Figure 4. Two possibilities: (a) A_1 shoots; (b) A_1 passes to A_2.

A_1, A_2, and A_3 belong to the same team, team A, whereas B_1 and B_2 belong to the opposing team, team B. A_1 is the player currently in control of the ball. The goal of A_1 is to help its own team to score. More precisely A_1 must establish what is the action that (from its point of view) is more likely to help its own team in scoring. For the sake of simplicity we suppose that A_1 can choose between two possible actions, namely

1. trying to score, and
2. giving the ball to another member of its own team.

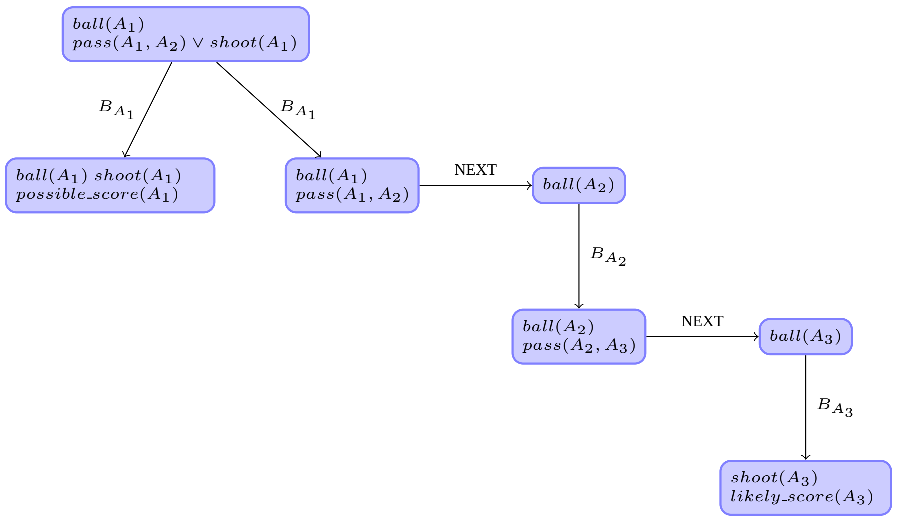
In order to decide what to do next, A_1 should reason about its knowledge about the game and its beliefs about both the current situation and the other players. Nevertheless, at different stages of the game, A_1 may have different constraints on how much time it can spend on reasoning. We here consider two simple cases. In the first scenario, A_1 has plenty of resources and so it may have a reasonable amount of time for reasoning about beliefs. In the second scenario the game is going to end very soon and a quick decision is required. This fact will modify the amount of time A_1 is able to spend in reasoning about beliefs. In particular we consider here the extreme situation in which A_1 does not have time to perform *any* reasoning about belief, but it has to decide what to do next considering only its own basic knowledge about the current situation.

Let us consider a simplified version of the example presented in [11], where we concentrate on the relevant knowledge of A_1.

1. **start** $\Rightarrow ball(A_1)$
2. **start** $\Rightarrow pass(A_1, A_2) \vee shoot(A_1)$
3. **start** $\Rightarrow B_{A_1}[shoot(A_1) \Rightarrow \Diamond possible\_score(A_1)]$
4. **start** $\Rightarrow B_{A_1}[pass(A_1, A_2) \Rightarrow \bigcirc B_{A_2}[pass(A_2, A_3) \wedge \bigcirc B_{A_3}(shoot(A_3) \wedge \Diamond likely\_score(A_3))]]$
5. $ball(X) \wedge pass(X, Y) \Rightarrow \bigcirc ball(Y)$
6. $[ball(X) \bigwedge_{Y \in S, X \neq Y} \neg pass(X, Y)] \Rightarrow \bigcirc ball(X)$
7. $[ball(Y) \wedge \neg pass(Y, X)] \Rightarrow \bigcirc \neg ball(X)$
8. $ball(X) \Rightarrow \bigwedge_{Y \in S, X \neq Y} \neg ball(Y)$
9. $ball(X) \Rightarrow B_X ball(X)$

Informally, the meaning of these formulae (or rules) is as follows.

1. A_1 is in charge of the ball at the beginning of time;
2. A_1 can pass the ball to A_2 or shoot;
3. A_1 believes to have some possibility of scoring in the case of shooting;
4. A_1 believes that it can pass the ball to A_2, and that A_2 will pass to A_3 who is in a better position to score;
5. if a player X is in control of the ball, and X passes the ball to a player Y, then at the next moment in time Y is in control of the ball;
6. if a player X is in control of the ball, and does not pass the ball then at the next moment in time X is still in control of the ball;
7. if a player X does not receive the ball then at the next moment in time X is still not in control of the ball
8. at each moment in time there is a unique player X in control of the ball;
9. if X is in control of the ball then X believes it is in control of the ball.



$ball(A_1)$ $pass(A_1, A_2) \vee shoot(A_1)$
B_{A_1}
B_{A_1}
$ball(A_1)$ $shoot(A_1)$ $possible\_score(A_1)$
$ball(A_1)$ $pass(A_1, A_2)$
NEXT
$ball(A_2)$
B_{A_2}
$ball(A_2)$ $pass(A_2, A_3)$
NEXT
$ball(A_3)$
B_{A_3}
$shoot(A_3)$ $likely\_score(A_3)$

Figure 5. Execution of the football example.

Rules 5 – 9 are valid in any belief context. We omit the transformation of these formulae in normal form and show directly how execution is attempted and how the model depicted in Figure 5 is constructed.

Step 1: from rules 1 and 2 build an initial state in which $ball(A_1)$ and $pass(A_1, A_2) \vee shoot(A_1)$ are true.

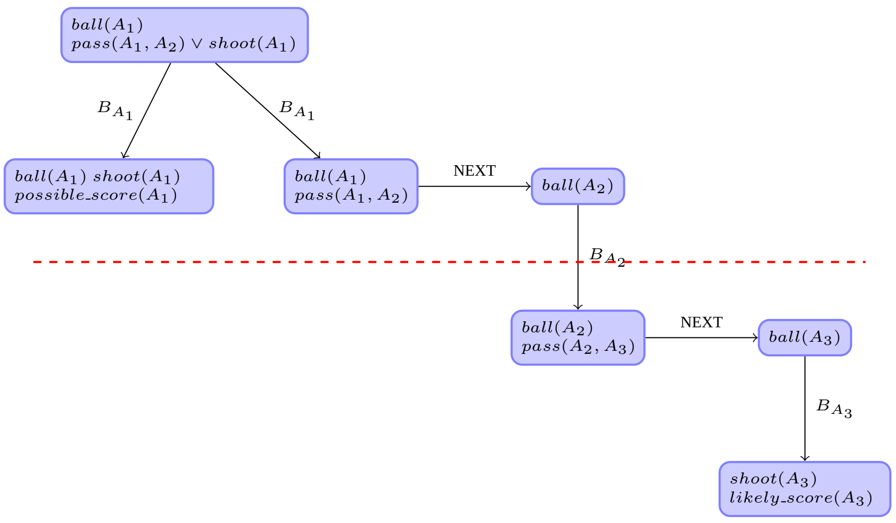
Step 2: two alternative paths are explored, one in which $shoot(A_1)$ is true, and the other in which $pass(A_1, A_2)$ is true. In the former $possible\_score(A_1)$ is made true using rule 3.

Step 3: the execution continues exploring the second alternative, and uses rule 5 to make $\bigcirc ball(A_2)$ true.

Step 4: now rule 4 is used to explore the beliefs of A_1 about A_2 that A_2 will pass the ball to A_3, and then again rule 5 to effectively pass the ball to A_3 in the next moment in time.

Step 5: Finally rule 4 is used again to explore the beliefs of A_1 about A_2 about A_3 saying that A_3 will shoot and likely score if he is in charge of the ball.

If we assume that A_1 has some internal ordering concerning eventualities and $\diamondsuit likely\_score(.)$ is preferred to $\diamondsuit possible\_score(.)$, then the agent may choose to pass the ball to A_2 on the basis of the hypothetical reasoning it has performed. As we can see from Figure 5 this hypothetical reasoning can happen under the assumption that A_1 has enough resources to explore belief about belief about belief. Assume now that the game is going to end soon, and that A_1 prefers to react immediately to the current situation than to spend time



$ball(A_1)$ $pass(A_1, A_2) \vee shoot(A_1)$
B_{A_1}
B_{A_1}
$ball(A_1)$ $shoot(A_1)$ $possible\_score(A_1)$
$ball(A_1)$ $pass(A_1, A_2)$
NEXT
$ball(A_2)$
B_{A_2}
$ball(A_2)$ $pass(A_2, A_3)$
NEXT
$ball(A_3)$
B_{A_3}
$shoot(A_3)$ $likely\_score(A_3)$

Figure 6. Adding belief bound to the football example.

”thinking”. In this situation the agent could have a very small belief bound such as $belief\_bound(1)$. In this case most of the hypothetical reasoning is cut away, as we can see in Figure 6, and the agent A_1 only has the choice of shooting directly and trying to score.

4. Exploring the Future

Once we have bounding of belief and temporal exploration, and we have syntactic control of this, through predicates such as $belief\_bound()$ and $tl\_bound()$, then we can consider a variety of more sophisticated extensions. In this section, we will outline some of these aspects.

4.1. Agents Adapting their Bounds to the Environment

With an explicit representation of the agent’s limitations, through predicates such as $belief\_bound()$ and $tl\_bound()$, we can think of ways (using formulae) of specifying how the value of the bound relates to previous values or to the current situation of the agent. As a very simple example of this consider the following formulae which define how the belief bound might evolve depending upon whether the agent is in $danger$, is $happy$, or is $cautious$.

$$
\begin{aligned}
\Box\,(belief\_bound(k) \wedge danger &\Rightarrow \bigcirc\, belief\_bound(k_{danger}))\\
\Box\,(belief\_bound(k) \wedge \neg danger \wedge happy &\Rightarrow \bigcirc\, belief\_bound(i_{happy}))\\
\Box\,(belief\_bound(k) \wedge \neg danger \wedge \neg happy \wedge cautious &\Rightarrow \bigcirc\, belief\_bound(i_{cautious}))\\
\Box\,(belief\_bound(k) \wedge \neg danger \wedge \neg happy \wedge \neg cautious &\Rightarrow \bigcirc\, belief\_bound(k))
\end{aligned}
$$

Here the values of k_{danger}, is k_{happy}, and $k_{cautious}$ are provided “a priori” within the specification, but nevertheless provide a simple and effective way of adapting to different situations.

4.2. AGENTS REASONING ABOUT THEIR OWN LIMITS

Once an agent has an explicit representation of its limitations, through predicates such as $belief\_bound()$ and $tl\_bound()$, then it is able to reason about these. In particular, the agent can assess its own bounds to decide what to do. For example, it may be that, under the constraint that $belief\_bound(34)$ then we can only do α, but if $belief\_bound(35)$ then the agent has a choice of doing α *or* β. So, deliberation must be extended to incorporate beliefs about $belief\_bound()$ and $tl\_bound()$, for example $B_{A1} belief\_bound(50)$. This allows the agent to select its activities based on whether it believes it has enough time/space to consider these. (It is important to note that, when $B_{A1} belief\_bound(50)$ is true, this doesn’t imply that $belief\_bound(50)$ is necessarily true, but just describes agent $A1$’s belief about its belief bound — it could be wrong!)

This leads on to questions of estimating future $belief\_bound()$ or $tl\_bound()$ values (for example, what value will $belief\_bound()$ have in the *next* moment?), what will the agent do if it does not have enough resources (for example, ask another agent to carry out some of the exploration?), and should we even take the cost of deliberation into account?!

4.3. WORKING WITH OTHER AGENTS’ LIMITS

While an agent might well know its own $belief\_bound()$ and $tl\_bound()$ values, it is unlikely to know this for other agents. Unless told explicitly by another agent, an agent must estimate (or even guess!) these, for example from observations or execution histories. Once an agent can describe another’s $belief\_bound()$ or $tl\_bound()$, then it can reason about them and use them in deliberation.

Reasoning about another agent’s limits can help with

– *cooperation* — in that the agent can plan/decide to do something that the other agent can cope with,

– *competition* — if the agent knows that others have small limits, then the agent can tackle complex problems with less competition,

– *negotiation* — similarly if an agent knows the extent of an opponents capabilities, it might use more sophisticated arguments or negotiation strategies.

4.4. Deciding on your limits

Once the agent is able to *set* its own bounds, then it can decide in which situations to set them appropriately. Thus the agent might deliberate about its environment and decide whether to have a small $belief\_bound()/tl\_bound()$ (i.e. for a quick response) or a large one (i.e. the agent can take its time)?

4.5. Extending the Power

In the examples given in this paper, agents are able to specify belief/temporal bounds using $belief\_bound()/tl\_bound()$, where the particular bound is taken from a fixed set of possibilities. Though this is simple and tractable, there might well be situations where more complex constraints on bounding are required. Can we extend the complexity of the bound, for example incorporating arithmetic or real numbers, while still retaining some tractability?

5. Concluding Remarks

In this paper we have provided an overview of work on executable agent specifications, particularly focusing on resource-boundedness. Within our framework, this resource boundedness is achieved through explicit bounds on the depth of nesting of both belief and temporal contexts. This allows close control of model exploration as execution proceeds; full details of this approach can be found in [13].

In addition to providing an overview of this work, we have indicated some extensions being explored. With all such extensions, it is usually a case of balancing the additional expressive power achieved and enhanced applicability with both tractability and implementation issues, while at the same time ensuring continued correctness.

References

1. Barringer, H., M. Fisher, D. Gabbay, R. Owens, and M. Reynolds (eds.): 1996, *The Imperative Future: Principles of Executable Temporal Logics*. Chichester, United Kingdom: Research Studies Press.
2. Bordini, R. H., M. Dastani, J. Dix, and A. El Fallah Seghrouchni (eds.): 2005, *Multi-Agent Programming: Languages, Platforms and Applications*. Springer-Verlag.
3. Bradshaw, J., M. Greaves, H. Holmback, T. Karygiannis, B. Silverman, N. Suri, and A. Wong: 1999, 'Agents for the Masses?'. *IEEE Intelligent Systems* **14**(2), 53–63.
4. Cimatti, A. and L. Serafini: 1996, 'Mechanizing Multi-Agent Reasoning with Belief Contexts'. In: *Practical Reasoning, International Conference on Formal and Applied Practical Reasoning, FAPR'96*. pp. 694–696.
5. Emerson, E. A.: 1990, 'Temporal and Modal Logic'. In: J. van Leeuwen (ed.): *Handbook of Theoretical Computer Science*. Elsevier, pp. 996–1072.

6. Fisher, M.: 1996, ‘An Introduction to Executable Temporal Logics’. *Knowledge Engineering Review* **11**(1), 43–56.
7. Fisher, M.: 1997, ‘A Normal Form for Temporal Logic and its Application in Theorem-Proving and Execution’. *J. Logic and Computation* **7**(4), 429–456.
8. Fisher, M.: 2005, ‘Temporal Development Methods for Agent-Based Systems’. *J. Autonomous Agents and Multi-Agent Systems* **10**(1), 41–66.
9. Fisher, M.: 2008, ‘Executable Specification of Deliberative Agents’. (Submitted).
10. Fisher, M. and C. Ghidini: 1999, ‘Programming Resource-Bounded Deliberative Agents’. In: *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI’99)*. pp. 200–206.
11. Fisher, M. and C. Ghidini: 2000, ‘Agents Playing with Dynamic Resource Bounds’. In: *ECAI Workshop on Balancing Reactivity and Social Deliberation in Multi-Agent Systems*. Berlin, Germany.
12. Fisher, M. and C. Ghidini: 2002, ‘Agents with Bounded Temporal Resources’. *Lecture Notes in Computer Science* **2403**, 169–184.
13. Fisher, M. and C. Ghidini: 2008, ‘Executable Specifications of Resource-Bounded Agents’. (Submitted).
14. Ghidini, C.: 1999, ‘Modelling (Un)Bounded Beliefs’. In: *Proc. Second International and Interdisciplinary Conf. on Modeling and Using Context (CONTEXT)*. Trento, Italy.
15. Ghidini, C. and F. Giunchiglia: 2001, ‘Local Models Semantics, or Contextual Reasoning = Locality + Compatibility’. *Artificial Intelligence* **127**(4), 221–259.
16. Halpern, J. Y. and Y. Moses: 1992, ‘A Guide to Completeness and Complexity for Modal Logics of Knowledge and Belief’. *Artificial Intelligence* **54**, 319–379.
17. Rao, A. S.: 1998, ‘Decision Procedures for Propositional Linear-Time Belief-Desire-Intention Logics’. *J. Logic and Computation* **8**(3), 293–342.
18. Rao, A. S. and M. Georgeff: 1995, ‘BDI Agents: from theory to practice’. In: *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS)*. San Francisco, CA, pp. 312–319.
19. van Linder, B., W. van der Hoek, and J. J. C. Meyer: 1996, ‘Formalising Motivational Attitudes of Agents’. In: *Intelligent Agents II*, Vol. 1037 of *Lecture Notes in Computer Science*. Springer-Verlag, pp. 17–32.
20. Wooldridge, M.: 2000, *Reasoning about Rational Agents*. MIT Press.
21. Wooldridge, M. and P. Ciancarini: 2001, ‘Agent-Oriented Software Engineering: The State of the Art’. In: *Agent-Oriented Software Engineering*, Vol. 1957 of *Lecture Notes in Artificial Intelligence*.
22. Wooldridge, M. and N. R. Jennings: 1995, ‘Intelligent Agents: Theory and Practice’. *The Knowledge Engineering Review* **10**(2), 115–152.
23. Wooldridge, M. and A. Rao (eds.): 1999, *Foundations of Rational Agency*, Applied Logic Series. Kluwer Academic Publishers.