An Operational Semantics for AgentSpeak(RT)

Konstantin Vikhorev[1], Natasha Alechina[1], Rafael H. Bordini[2], and Brian Logan[1]

[1] School of Computer Science
University of Nottingham
Nottingham NG8 1BB UK
{kxv,nza,bsl}@cs.nott.ac.uk
[2] Institute of Informatics
Federal University of Rio Grande do Sul
Porto Alegre, Brazil
r.bordini@inf.ufrgs.br

Abstract. In this paper we give an operational semantics for the real-time agent programming language AgentSpeak(RT). AgentSpeak(RT) was introduced in [21], and extends AgentSpeak(L) with deadlines and priorities for intentions. The version of AgentSpeak(RT) presented in this paper differs in certain aspects from that in [21], mainly to incorporate both hard and soft deadlines, and allow for the concurrent execution of intentions.

1 Introduction

In this paper we give an operational semantics for the AgentSpeak(RT) real-time agent programming language introduced in [21]. In AgentSpeak(RT), an agent's intentions have priorities and deadlines. In a dynamic environment, a real-time BDI agent that has more tasks that it can feasibly accomplish by their deadlines should make a rational choice regarding which tasks to commit to: that is, it should try to accomplish higher priority tasks, but also try to execute tasks so that they are accomplished by their deadlines. An AgentSpeak(RT) agent commits to a set of intentions that are 'maximally feasible': no more intentions can be added to the schedule if the scheduled intentions are to remain feasible at the specified confidence level, and moreover, intentions which are dropped are incompatible with some scheduled higher priority intention(s).

In the version of AgentSpeak(RT) introduced in [21], only hard deadlines were supported: that is, intentions were dropped if was impossible to execute them by their deadlines. This is reasonable for many tasks and environments, for example writing conference papers, sending bids to auctions, or catching trains. However, some tasks have soft deadlines: it may be desirable to finish a certain task by its deadline, but the work still has to be done after the deadline is passed. For example, the agent may need to charge its battery every 24 hours, but if it is delayed for some reason in reaching the charging station after 24 hours and one second, it is still important to reach the charging station. For this reason, we introduce *soft deadlines* in this version of AgentSpeak(RT) we consider. Another important difference from [21] is that we assume that tasks may be executed concurrently. For example, an agent may be simultaneously moving to a new location and communicating with other agents. However, some intentions need to

be executed atomically to prevent undesired interactions between different actions. For this reason, in the version of AgentSpeak(RT) we consider here, we introduce the notion of *atomic* plans (which cannot be executed simultaneously with other atomic plans).

The rest of this paper is organised as follows. In section 2, we briefly describe the AgentSpeak(RT) architecture and the modifications relative to [21]. In section 3, we define the operational semantics of AgentSpeak(RT). We survey related work in section 4 and conclude in section 5.

2 The AgentSpeak(RT) Architecture

In this section we introduce the AgentSpeak(RT) agent programming language and its associated interpreter. Note that the version of the language presented here differs from the one in [21].

We assume that an AgentSpeak(RT) agent operates in a real-time task environment, that is, top-level goals may optionally specify a deadline and/or a priority. An AgentSpeak(RT) agent responds to events by adopting and executing intentions. A developer can specify the required level of confidence for the successful execution of intentions in terms of a probability α. An AgentSpeak(RT) agent should schedule its intentions so as to ensure that the probability that intentions complete by their deadlines is at least α. If not all intentions can be executed with the required level of confidence due to lack of time, the agent favours intentions responding to high priority events.

The syntax and semantics of AgentSpeak(RT) with various minor modifications is based on AgentSpeak(L) [18]. To illustrate the syntax of AgentSpeak(RT) we use a simple running example of an agent which removes litter from a parking lot. Each evening, the agent is given a set of goals to achieve, each of which specifies the removal of a particular item of litter from particular parking space. In addition, the agent may detect additional litter while moving around the lot. There is a deadline for the removal of litter e.g., before the barrier is opened in the morning (we assume the agent can't cope with parking cars), and it is more important to remove some types of litter (e.g., broken glass) than others (e.g., paper).

The AgentSpeak(RT) architecture consists of five main components: a belief base, a set of events, a plan library, an intention structure, and an interpreter.

2.1 Beliefs and Goals

The agent's beliefs represent the agent's information about its environment, e.g., sensory input, information about other agents, etc. Beliefs are represented as ground atomic formulas. For example, the agent may believe that it is in space1 and there is some litter in space2:

```
at(robot,space1)
litter(paper,space2)
```

A belief atom or its negation is called a *belief literal*. A ground belief atom is called a *base belief*, and the agent's belief base is a conjunction of base beliefs.

A goal is a state the agent wishes to bring about or a query to be evaluated. An achievement goal, written $!g(t_1, \ldots, t_n)$ where $t_i, \ldots, t_n$ are terms, specifies that the agent wishes to achieve a state in which $g(t_1, \ldots, t_n)$ is a true belief. A test goal, written $?g(t_1, \ldots, t_n)$, specifies that the agent wishes to determine if $g(t_1, \ldots, t_n)$ is a true belief. For example, the goals

```
!remove(paper,space2)
?parked(X,space2)
```

indicate that the agent wants to remove the paper in space2, and determine if there is a car parked in space2.[3]

2.2 Events

Events correspond to changes in the agent's beliefs or the acquisition of new achievement goals. An addition event, denoted by $+$, indicates the addition of a base belief or an achievement goal. A deletion event, denoted by $-$, indicates the retraction of a base belief.[4] Events can be internal or external. External events originate outside the agent, while internal events result from the execution of the agent's program. As in AgentSpeak(L), all belief change events are external (originating in the agent's environment), while goal change events may be external (goals originated by a user or another agent) or internal (subgoals generated by the agent's program in response to an external event).

To allow the specification of real-time tasks, external goal addition events may optionally specify a deadline and a priority. A *deadline* specifies the time by which a goal should be achieved. Deadlines are expressed as real time values in some appropriate units, e.g, a user may specify a deadline for a goal as "4pm on Friday". Deadlines in AgentSpeak(RT) may be hard or soft. For a hard deadline it is assumed that there is no value in achieving a goal after the deadline has passed. For a soft deadline, it may still make sense to continue trying to achieve the goal after the deadline has passed, provided that this does not interfere with higher priority goals. A *priority* specifies the relative importance of achieving the goal. Priorities define a partial order over events and are expressed as non-negative integer values, with larger values taken to indicate higher priority. For example, the event

```
+!remove(paper,space2)[8am,10]
```

indicates the acquisition of a goal to remove some paper from space2 with deadline 8am and priority 10. By default the deadline is equal to infinity and the priority is equal to zero.

2.3 Plans

Plans specify sequences of actions and subgoals an agent can use to achieve its goals or respond to changes in its beliefs. The head of a plan consists of a triggering event which

[3] As in Prolog, constants are written in lower case and variables in upper case, and all negations must be ground when evaluated.

[4] In the interests of brevity, we do not consider goal deletion events.

specifies the kind of event the plan can be used to respond to, and a belief context which specifies the beliefs that must be true for the plan to be applicable. The body of a plan specifies a sequence of actions which need to be executed and subgoals which need to be achieved in response to the triggering event.

Actions are the basic operations an agent can perform to change its environment in order to achieve its goals. Actions are denoted by *action symbols* and are written $a(t_1, \ldots, t_n)$ where a is an action symbol and $t_1, \ldots, t_n$ are the (ground) arguments to the action. For example, the action

```
move(trashcan)
```

will cause the agent to move from a parking space to the trashcan.

Plans may also contain achievement and test (sub)goals. Achievement subgoals allow an agent to choose a course of action as part of a larger plan on the basis of its current beliefs. An achievement subgoal $!\,g(t_1, \ldots, t_n)$ gives rise to a internal goal addition event $+!\,g(t_1, \ldots, t_n)$ which may in turn trigger subplans at the next execution cycle. Test goals are evaluated against the agent's belief base, possibly binding variables in the plan. For example, the plan

```
+litter(L,S) : at(robot,S1) & not parked(C,S) <-
  move(S); pickup(L); move(trashcan); deposit(L).
```

causes the agent to remove litter from the parking space the agent is in if there is no car parked in the space.

The BNF for plans is given below:

belief-event	::= "+" *atomic-formula* | "−" *atomic-formula*
goal-event	::= "+!" *atomic-formula* [*realtime-spec*]
belief-plan	::= "@" *label* ["`atomic`"] *belief-event* ["`:`" *context*] "<−" (*body* | "`!`" *atomic-formula realtime-spec*) "`.`"
goal-plan	::= "@" *label* ["`atomic`"] *goal-event* ["`:`" *context*] "<−" *body* "`.`"
context	::= *true* | *literal* ("`&`" *literal*)*
literal	::= *atomic-formula* | "`not`" *atomic-formula*
atomic-formula	::= $p(t_1, \ldots, t_n)$
realtime-spec	::= "[" (($hd(time)$ | $sd(time)$) "," *number*) | ($hd(time)$ | $sd(time)$) | *number* "]"
body	::= *true* | *step* ("`;`" *step*)*
step	::= $a(t_1, \ldots, t_n)$ | "`!`" *atomic-formula* | "`?`" *atomic-formula*

where *label* is a string uniquely identifying a plan, p and a are respectively predicate and action symbols of arity $n \geq 0$, and $t_1, \ldots, t_n$ are terms.

AgentSpeak(RT) allows a potentially unbounded number of plans to execute concurrently (assuming actions are not executing on the same CPU as the interpreter). However, plans may be declared as requiring exclusive access to a single 'lock'. Intentions which do not contain an atomic plan may execute concurrently. If two plans are mutually exclusive, the execution of the intentions containing the plans must be serialised, as explained below.

The concurrent execution of intentions in AgentSpeak(RT) is similar to capabilities provided by atomic plans in Jason [1] and 2APL [6]. An atomic plan is a plan which should be executed ensuring that its execution is not interleaved with the execution of the goals and actions of other plans of the same agent. The resulting agent system is more expressive than Jason and 2APL in one sense, as Jason and 2APL cannot run non-atomic plans in parallel with an atomic one. However, it is less expressive in another sense, as in Jason and 2APL a non-atomic plan can have an atomic subplan.

In order to determine whether a plan can achieve a goal by a deadline with a given level of confidence, each action and plan has an associated *execution time profile* which specifies the probability that the action or plan will terminate successfully as a function of execution time. The expected execution time for an action or plan ϕ at confidence level α is given by $et(\phi, \alpha)$. The execution time profile will typically be influenced by the characteristics of the environment in which the agent will operate. For example, the probability of a plan to move to a location terminating successfully within a given time may be lower in environments with many obstacles than in environments with fewer obstacles.

Execution time profiles can be derived from an analysis of the agent's actions, plans and environment, or using automated techniques, e.g., stochastic simulation. In the simple case of a plan consisting of a sequence of actions, the execution time profile for the plan can computed from the execution time profiles of its constituent actions. However for plans which contain subgoals, the execution time will depend on the relative frequency with which the alternative plans for a subgoal are selected in the agent's task environment.

2.4 Intentions

Plans triggered by changes in beliefs or the acquisition of an external (top-level) achievement goal give rise to new intentions. Plans triggered by the processing of an achievement subgoal in an already intended plan are pushed onto the intention containing the subgoal. Each intention consists of a stack of partially executed plans, a set of substitutions for plan variables, a set of shared resources, a deadline and priority. The set of variable substitutions for each plan in an intention results from matching the belief context of the plan and any test goals it contains against the agent's belief base. The deadline and priority of an intention are determined by the triggering event of the root plan.

Each intention can be in one of two states: *executing* and *executable*. An intention is executing if the first action in the topmost plan in the stack of partially executed plans which forms the intention is currently executing. If the first step in the topmost plan is a goal or an action which is not currently executing, the intention is said to be executable.

2.5 The AgentSpeak(RT) Interpreter

The interpreter is the main component of the agent. It manipulates the agent's belief base, event queue and intention structure, deliberates about which plan to select in response to belief and goal change events, and schedules and executes intentions.

Algorithm 1 AgentSpeak(RT) Interpreter Cycle

$E := E \cup G \cup \mathit{belief\text{-}events}(B, P)$
$B := \mathit{update\text{-}beliefs}(B, P)$
for all $(e, \tau) \in E$ **do**
  $O_e := \{\pi\theta \mid \theta$ is an applicable unifier for e and plan $\pi\}$
  $\pi\theta := S_O(O_e)$
  if $\pi\theta \neq \emptyset$ **and** $\tau \notin I$ **then**
    $I := I \cup \pi\theta$
  else if $\pi\theta \neq \emptyset$ **and** $\tau \in I$ **then**
    $I := (I \setminus \tau) \cup \mathit{push}(\pi\theta\sigma, \tau)$ where σ is an mgu for $\pi\theta$ and τ
  else if $\pi\theta = \emptyset$ **and** $\tau \in I$ **then**
    $I := I \setminus \tau$
  end if
end for
$I :=$ SCHEDULE(I)
for $\tau \in I$ **do**
  if $s(\tau) = \mathit{now} \wedge \mathit{executable}(\tau)$ **then**
    if $\mathit{completed}(\mathit{first}(\mathit{body}(\mathit{top}(\tau))))$ **then**
      $\pi := \mathit{pop}(\tau)$
      $\mathit{push}(\mathit{head}(\pi) \leftarrow \mathit{rest}(\mathit{body}(\pi)), \tau)$
    end if
    if $\mathit{first}(\mathit{body}(\mathit{top}(\tau))) = \mathit{true}$ **then**
      $\pi := \mathit{pop}(\tau)$, $\pi' := \mathit{pop}(\tau)$
      $\mathit{push}((\mathit{head}(\pi') \leftarrow \mathit{rest}(\mathit{body}(\pi')))\theta, \tau)$
      where θ is an mgu such that $\mathit{head}(\pi)\theta = \pi'\theta$
    else if $\mathit{first}(\mathit{body}(\mathit{top}(\tau))) = !g(t_1, \ldots, t_n)$ **then**
      $E = \{(+!g(t_1, \ldots, t_n), \tau)\}$
    else if $\mathit{first}(\mathit{body}(\mathit{top}(\tau))) = ?g(t_1, \ldots, t_n)$ **then**
      if $?g(t_1, \ldots, t_n)\theta$ is an answer substitution **then**
        $\pi := \mathit{pop}(\tau)$
        $\mathit{push}((\mathit{head}(\pi) \leftarrow \mathit{rest}(\mathit{body}(\pi)))\theta, \tau)$
      else
        $I := I \setminus \tau$
      end if
    else if $\mathit{first}(\mathit{body}(\mathit{top}(\tau))) = a(t_1, \ldots, t_n)$ **then**
      $\mathit{execute}(a(t_1, \ldots, t_n))$
    end if
    break
  end if
end for

The interpreter code is shown in Algorithm 1. B is the agent's belief base, E is the set of events, I is a partially ordered set of intentions. The functions *head* and *body* return the head and body of an intended plan, and *first* and *rest* are used to return the first and all but the first elements of a sequence. The function *top* returns the topmost plan in an intention. The function *pop* removes and returns the topmost plan of an intention and the function *push* takes a plan (and any substitution) and an intention and pushes the plan onto the top of the intention. The function *executable* takes an intention and returns true if the intention is executable and the function *completed* returns true if the first step in an executable intention is an action that has completed execution. The function *execute* initiates the execution of an action in a separate thread.

In contrast to AgentSpeak(L) which processes a single event at each interpreter cycle, to ensure reactivity, AgentSpeak(RT) iterates through the set of events E, and, for each event $e \in E$, generates a set of applicable plans O_e. A plan is *relevant* if its triggering event can be unified with e and a relevant plan is *applicable* if its belief context is true in B'. In general, there may be many applicable plans or options for each event. A selection function S_O chooses one of these plans for each event to give a set of options $O = \{S_O(O_e) \mid e \in E\}$. S_O is a partial function, i.e., it is not defined if O_e is empty. If the event was triggered by a subgoal of an existing intention, failure to find a applicable plan for the subgoal, i.e., if $O_e = \emptyset$, aborts the intention which posted the subgoal and the intention is removed from I. For each plan π in the set of applicable plans, if the triggering event for π was internal, the plan is pushed on top of the existing intention in I that generated the triggering event. If the triggering event for π was external, a new intention τ is created and added to I.

The scheduling algorithm is applied to I and returns a priority-maximal set of feasible intentions together with their start times. Finally, an executable intention is chosen from I for execution. An intention is executable if the first step in the topmost plan in the stack of partially executed plans that forms the intention is a goal or an action which is not currently executing (i.e., it has either completed executing or has yet to begin execution). If the first step in an executable intention is an action which has completed execution, the completed action is removed from the plan. Execution then proceeds from the next step of the topmost plan in the intention.

Executing an executable intention involves executing the first goal or action of the body of the topmost plan in the stack of partially executed plans which forms the intention. Executing an achievement goal adds a corresponding internal goal addition event to E'. Executing a test goal involves finding a unifying substitution for the goal and the agent's base beliefs. If a substitution is found, the test goal is removed from the body of the plan and the substitution is applied to rest of the body of plan. If no such substitution exists, the intention is dropped and removed from I. Executing an action results in the invocation of the Java code that implements the action and changes the state of the intention from executable to executing. We assume that action execution is performed in a separate thread, and execution of the AgentSpeak(RT) interpreter resumes immediately after initiating the action. Reaching the end of a plan (denoted by *true* below) causes the plan to be popped from the intention and any substitutions for variables appearing in the head of the popped plan are applied to the topmost plan in the intention.

The AgentSpeak(RT) Scheduler A schedule is a priority-maximal set feasible intentions together with their start times. A set of intentions $\{\tau_1, \ldots, \tau_n\}$ is *feasible* if

1. each intention will complete execution before its deadline with probability at least α, that is, for each scheduled intention τ_i

$$s(\tau_i) + et(\tau_i, \alpha) - ex(\tau_i) \leq d(\tau_i)$$

   where $s(\tau_i)$ is the time at which τ_i will next execute, $ex(\tau_i)$ is the time τ_i has spent executing up to this point, and $d(\tau_i)$ is the deadline for τ_i; and
2. if τ_i is an atomic intention, no intention τ_j scheduled to execute concurrently with τ_i $\{\tau_j \mid s(\tau_i) < s(\tau_j) + et(\tau_j, \alpha) \wedge s(\tau_j) < s(\tau_i) + et(\tau_i, \alpha)\}$ is atomic.

A set of intentions is priority-maximal if no more intentions can be added to the schedule if the scheduled intentions are to remain feasible at the specified confidence level, and intentions which are dropped are incompatible with some scheduled higher priority intention(s).

Algorithm 2 Scheduling Algorithm

```
function SCHEDULE(I)
    Γ_s := ∅, Γ_p := ∅
    for all τ ∈ I in descending order of priority do
        if ¬atomic(τ) then
            s(τ) := now
            if Γ_p ∪ {τ} is feasible then
                Γ_p := Γ_p ∪ {τ}
            end if
        else
            t := now
            Γ'_s := ∅
            for all τ' ∈ Γ_s do
                if d(τ') ≤ d(τ) then
                    Γ'_s := Γ'_s ∪ {τ'}
                    t := s(τ') + et(τ', α) − ex(τ')
                else
                    s(τ') := s(τ') + et(τ, α) − ex(τ)
                    Γ'_s := Γ'_s ∪ {τ'}
                end if
            end for
            s(τ) := t
            if Γ'_s ∪ {τ} is feasible then
                Γ_s = Γ'_s ∪ {τ}
            end if
        end if
    end for
    return Γ_p ∪ Γ_s
end function
```

Scheduling in AgentSpeak(RT) is pre-emptive in that the adoption of a new high-priority intention τ may prevent previously scheduled intentions with priority lower than τ (including currently executing intentions) being added to the new schedule. Intentions which exceed their expected execution time and/or their deadline may or may not be dropped, depending on whether the deadline is hard or soft and the amount of uncommitted or 'slack' time in the schedule. If an intention has a hard deadline that has been exceeded, the intention is dropped. If an intention has a soft deadline that has been exceeded, its deadline is reset to ∞ and its priority to 0. The agent will continue to purse the intention if it can be executed concurrently with other, higher priority intentions. However if the intention is atomic, it will be scheduled after all other atomic intentions. An intention τ which has exceeded its expected execution time but not its deadline has its priority reduced to 0 and its expected execution time reset to $ex(\tau) + \delta_a$, where δ_a is the expected time required to execute the next step in the intention. τ will only be scheduled if, after scheduling all higher priority intentions, there is sufficient slack in the schedule to execute at least one step in τ before its deadline. Given sufficient slack in the schedule, τ can therefore still complete successfully. It will be however dropped if it exceeds its deadline

The scheduling algorithm is shown in Algorithm 2. We assume that the deadlines, priorities and expected execution times of the input intentions I are adjusted as described above. The set of candidate intentions is processed in descending order of priority. For each intention τ, if the intention is atomic it is added to the schedule if it can be inserted into the schedule in deadline order while meeting its own and all currently scheduled deadlines. If the intention is not atomic an attempt is made to schedule it at $s(\tau) := now$. Intentions which are not feasible in the context of the current schedule are dropped. The resulting schedule can be computed in polynomial time (in fact, quadratic time) in the size of the set I, and is priority-maximal (see [21]).

3 Operational Semantics

This section gives semantics to AgentSpeak(RT) based on the operational semantics for AgentSpeak, by showing which rules have to be changed to new ones that are specific to AgentSpeak(RT). An earlier version of the operational semantics for AgentSpeak appeared in [4]. The semantics rules for communication appeared in [14] and were later improved and extended in [19]. The latter version (but without communication rules) forms the basis for this section.[5]

The operational semantics is given by a set of rules that define a transition relation between configurations $\langle ag, C, T, s\rangle$ where:

- An agent program ag is formed by a set of beliefs bs and a set of plans ps (as defined by the BNF in section 2.3 above).
- An agent's circumstance C is a tuple $\langle I, E, A\rangle$ where:
  - I is a set of *intentions* $\{\tau, \tau', \ldots\}$; each intention i is a stack of partially instantiated plans.

[5] Note that in the rules below, the notation for events, plans and intentions has been modified to be consistent with that in [21].

  - E is a set of *events* $\{(e, \tau), (e', \tau'), \ldots\}$. Each event is a pair (e, τ), where e is a triggering event and τ is an intention (a stack of plans in case of an internal event, or the empty intention T in case of an external event).
  - A is a set of *actions* to be performed in the environment.
- T is a tuple $\langle R, Ap, \iota, \varepsilon, \rho \rangle$ which keeps track of temporary information that is required in subsequent stages within a single reasoning cycle. Note that structure of each of these components have been changed from the original semantics because AgentSpeak(RT) handles all outstanding events in a single reasoning cycle, which is a significant change from original AgentSpeak. The components of T are:
  - R for the mapping from each of the events to the set of its *relevant plans*.
  - Ap for the sets of *applicable plans* (the relevant plans whose contexts are true), again a set for each of the events currently in the set of events.
  - ι, ε, and ρ record, in the original semantics, a particular intention, event, and applicable plan (respectively) being considered along the execution of one reasoning cycle; ι is not used here, and ε and ρ have been changed to be respectively a set rather than a single event and a mapping from each of the outstanding events to the selected applicable plan (i.e., intended means) to handle it.
- The current step s within an agent's reasoning cycle is symbolically annotated by $s \in \{\mathsf{SelEv}, \mathsf{RelPl}, \mathsf{ApplPl}, \mathsf{SelAppl}, \mathsf{AddIM}, \mathsf{SelInt}, \mathsf{ExecInt}, \mathsf{ClrInt}\}$, which stands for: selecting a set of events, retrieving all relevant plans, checking which of those are applicable, selecting applicable plans (the intended means), adding the new intended means to the set of intentions, selecting an intention, executing the selected intention, and clearing an intention or intended means that may have finished in the previous step.

In the interests of readability, we adopt the following notational conventions in our semantic rules:

- If C is an AgentSpeak agent circumstance, we write C_E to make reference to the component E of C. Similarly for all the other components of a configuration.
- We write $\tau[\pi]$ to denote the intention that has plan π on top of intention τ.

New Rules for Event Selection

As AgentSpeak(RT) typically handles all outstanding events, the **SelEv** rules have been changed as follows.

$$\frac{S_E(C_E) = SEs}{\langle ag, C, T, \mathsf{SelEv} \rangle \longrightarrow \langle ag, C', T', \mathsf{RelPl} \rangle} \quad \textbf{(SelEv}_\mathbf{1}\textbf{)}$$
$$\textit{where: } C'_E = C_E \setminus SEs$$
$$T'_\varepsilon = SEs$$

Above, SEs is a set of events, those that have been selected by S_E; by default in AgentSpeak(RT), S_E selects all the events in the set of events.

Rule $\mathbf{SelEv_2}$ skips to the intention execution part of the cycle, in case there is no event to handle. It is the same as in the original AgentSpeak semantics:

$$\frac{C_E = \{\}}{\langle ag, C, T, \mathsf{SelEv}\rangle \longrightarrow \langle ag, C, T, \mathsf{SelInt}\rangle} \qquad \textbf{(SelEv}_\mathbf{2}\textbf{)}$$

New Rules for Relevant Plans

In contrast to the original AgentSpeak, we now have to keep track of the set of relevant plans not for one chosen event but for a set of events (typically all currently outstanding events). T_ε therefore keeps track of this set of events. For each of the events in T_ε, we find a set of relevant plans for it and keep track of that in T_R.

$$\frac{T_\varepsilon = \{(e,\tau)\} \cup RES \qquad \mathsf{RelPlans}(ag_{ps}, e) \neq \{\}}{\langle ag, C, T, \mathsf{RelPl}\rangle \longrightarrow \langle ag, C, T', \mathsf{RelPl}\rangle} \qquad \textbf{(Rel}_\mathbf{1}\textbf{)}$$

$$\textit{where: } \begin{array}{l} T'_R = T_R \cup \{(e,\tau) \mapsto \mathsf{RelPlans}(ag_{ps}, e)\} \\ T'_\varepsilon \ = RES \end{array}$$

Events with no relevant plans are ignored, as shown in the rule below.

$$\frac{T_\varepsilon = \{(e,\tau)\} \cup RES \qquad \mathsf{RelPlans}(ag_{ps}, e) = \{\}}{\langle ag, C, T, \mathsf{RelPl}\rangle \longrightarrow \langle ag, C, T', \mathsf{RelPl}\rangle} \qquad \textbf{(Rel}_\mathbf{2}\textbf{)}$$

$$\textit{where: } T'_\varepsilon = RES$$

Finally, we need an additional rule that is used to go to the next stage of the reasoning cycle when relevant plans have been found for all previously selected events.

$$\frac{T_\varepsilon = \{\}}{\langle ag, C, T, \mathsf{RelPl}\rangle \longrightarrow \langle ag, C, T, \mathsf{ApplPl}\rangle} \qquad \textbf{(Rel}_\mathbf{3}\textbf{)}$$

New Rules for Applicable Plans

Again we need a couple of rules to handle each of the mappings from events to a set of relevant plans, filtering them to keep only the applicable ones (in T_{Ap}). Rule $\mathbf{Appl_1}$ handles the normal case (i.e., where applicable plans are found); $\mathbf{Appl_2}$ says that events with no applicable plans are ignored;[6] $\mathbf{Appl_3}$ deals with the case where we have updated all mappings and there are events with sets of applicable plans for which intended means need to be selected; finally, $\mathbf{Appl_4}$ handles the case where no new intended means will result in this reasoning cycle.

$$\frac{T_R = \{ev \mapsto RPs\} \cup ERs \qquad \mathsf{AppPlans}(ag_{bs}, RPs) \neq \{\}}{\langle ag, C, T, \mathsf{ApplPl}\rangle \longrightarrow \langle ag, C, T', \mathsf{ApplPl}\rangle} \qquad \textbf{(Appl}_\mathbf{1}\textbf{)}$$

$$\textit{where: } \begin{array}{l} T'_{Ap} = T_{Ap} \cup \{ev \mapsto \mathsf{AppPlans}(ag_{bs}, ER)\} \\ T'_R \ = ERs \end{array}$$

[6] Note that we do not consider here the plan failure handling mechanism introduced in some of the extensions of AgentSpeak.

$$\frac{T_R = \{ER\} \cup ERs \qquad \mathsf{AppPlans}(ag_{bs}, ER) = \{\}}{\langle ag, C, T, \mathsf{ApplPl} \rangle \longrightarrow \langle ag, C, T', \mathsf{ApplPl} \rangle} \quad \textbf{(Appl}_\mathbf{2}\textbf{)}$$

$$\textit{where: } T'_R = ERs$$

$$\frac{T_R = \{\} \qquad T_{Ap} \neq \{\}}{\langle ag, C, T, \mathsf{ApplPl} \rangle \longrightarrow \langle ag, C, T, \mathsf{SelAppl} \rangle} \quad \textbf{(Appl}_\mathbf{3}\textbf{)}$$

$$\frac{T_R = \{\} \qquad T_{Ap} = \{\}}{\langle ag, C, T, \mathsf{ApplPl} \rangle \longrightarrow \langle ag, C, T, \mathsf{SelInt} \rangle} \quad \textbf{(Appl}_\mathbf{4}\textbf{)}$$

New Rules for Selecting an Applicable Plan

As before, we need two rules: one to handle a particular mapping from an event to applicable plans and one for when all mappings have been processed.

$$\frac{T_{Ap} = \{ev \mapsto APs\} \cup EAs \qquad \mathcal{S}_{\mathcal{O}}(APs) = (\pi, \theta)}{\langle ag, C, T, \mathsf{SelAppl} \rangle \longrightarrow \langle ag, C, T', \mathsf{SelAppl} \rangle} \quad \textbf{(SelAppl}_\mathbf{1}\textbf{)}$$

$$\begin{array}{rl} \textit{where: } T'_\rho & = T_\rho \cup \{ev \mapsto (\pi, \theta)\} \\ T'_{Ap} & = EAs \end{array}$$

$$\frac{T_{Ap} = \{\}}{\langle ag, C, T, \mathsf{SelAppl} \rangle \longrightarrow \langle ag, C, T, \mathsf{AddIM} \rangle} \quad \textbf{(SelAppl}_\mathbf{2}\textbf{)}$$

New Rules for Adding an Intended Means to the Set of Intentions

For each mapping of an event to the chosen intended means, we need to move it to the set of intentions; as before this requires two rules, depeding on whether the particular event was internal or external. Then we need rule **EndIM** for when all mappings have been processed.

It is important to note that the set C_I updated here is subsequently *ordered* by the schedulling algorithm presented in section 2, as stated in rules **EndIM**. In future work, we aim to formalize the scheduling of intentions within the operational semantics, by giving further rules that describe the schedulling of the intentions based on the real-time criteria.

$$\frac{T_\rho = \{(e, \mathsf{T}) \mapsto (\pi, \theta)\} \cup EIs}{\langle ag, C, T, \mathsf{AddIM} \rangle \longrightarrow \langle ag, C', T', \mathsf{AddIM} \rangle} \quad \textbf{(ExtEv)}$$

$$\begin{array}{rl} \textit{where: } C'_I & = C_I \cup \{\ [\pi\theta]\ \} \\ T'_\rho & = EIs \end{array}$$

$$\frac{T_\rho = \{(e,\tau) \mapsto (\pi,\theta)\} \cup EIs}{\langle ag, C, T, \mathsf{AddIM}\rangle \longrightarrow \langle ag, C', T', \mathsf{AddIM}\rangle} \quad \textbf{(IntEv)}$$

$$\textit{where: } C'_I = C_I \cup \{\ \tau[(\pi\theta)]\ \} \qquad T'_\rho = EIs$$

$$\frac{T_\rho = \{\}}{\langle ag, C, T, \mathsf{AddIM}\rangle \longrightarrow \langle ag, C', T, \mathsf{SelInt}\rangle} \quad \textbf{(EndIM)}$$

$$\textit{where: } C'_I = \textsc{schedule}(C_I)$$

4 Related Work

Two strands of work on agent programming languages are related to work reported in this paper.

One strand is work on agent programming languages designed for developing agents with real-time capabilities. For example, the Procedural Reasoning System (PRS) [10] and PRS-like systems, e.g., JAM [12] and SPARK [15], have features such as meta-level reasoning which facilitate the development of agents for real time environments. However, to guarantee real time behaviour, these systems have to be programmed for each particular task environment—there are no general methods or tools which allow the agent developer to specify that a particular goal should be achieved by a specified time or that an action should be performed within a particular interval of an event occurring. In contrast, AgentSpeak(RT) provides a high-level programmatic interface to a standardised real-time reasoning mechanism for tasks with different priorities and deadlines.

More closely related to real-time aspects of AgentSpeak(RT) are architectures such as the Soft Real-Time Agent Architecture [22] and AgentSpeak(XL) [2]. These architectures use the TÆMS (Task Analysis, Environment Modelling, and Simulation) framework [7] together with Design-To-Criteria scheduling [23] to schedule intentions. TÆMS provides a high-level framework for specifying the expected quality, cost and duration of of methods (actions) and relationships between tasks (plans). Like AgentSpeak(RT), methods and tasks can have deadlines, and TÆMS assumes the availability of probability distributions over expected execution times (and quality and costs). DTC decides which tasks to perform, how to perform them, and the order in which they should be performed, so as to satisfy hard constraints (e.g., deadlines) and maximise the agent's objective function. In comparison to AgentSpeak(RT), TÆMS allows the specification of more complex interactions between tasks, and DTC can produce schedules which allow interleaved or parallel execution of tasks. However the view of 'real-time' used in these systems is different from that taken by AgentSpeak(RT), for example in considering only soft deadlines (all tasks still have value after their deadline).

As mentioned in the Introduction, AgentSpeak(RT) was first introduced in [21]. An earlier version of this work (extending PRS rather than AgentSpeak with priorities and deadlines) was reported in [20].

Another strand of related work is research on the formal semantics of agent programming languages. Many agent-oriented programming languages have been formalised

using the operational semantics approach, for example, AgentSpeak[19], GOAL [11], 2APL [6], and CAN [24], as well as the AIL effort towards unifying semantics [8]. However, to the best of our knowledge this work has not dealt with issues relating to real-time agency, and to this extent the work presented here is novel. There are, of course, various other approaches in the literature that have logical semantics, for example, MINERVA [13], and others, such as CLAIM [9] that have a formal model based on process algebra, but the operational semantics approach seems more popular in the agent programming language community. There are also important agent programming languages and platforms that have no formal semantics, such as JADEX [17] and SPARK [16], for example.

5 Conclusion

In this paper we described a modified version of AgentSpeak(RT) with parallel execution of intentions and with soft as well as hard dealdines. It is intended to be more programmer-friendly and flexible compared to the original version introduced in [21] which was designed to provide provable probabilistic guarantees of real-time behaviour. We provide an operational semantics for the language in order to make it precise and facilitate analysis of program written in the language.

References

1. Bordini, R.H., Hübner, J.F., Vieira, R.: Multi-agent programming : languages, platforms and applications, chap. Jason and the Golden Fleece of agent-oriented programming, pp. 3–37. Multiagent Systems, Artificial Societies, and Simulated Organizations, Springer, New Yor, USA (2005)
2. Bordini, R., Bazzan, A.L.C., Jannone, R.d.O., Basso, D.M., Vicari, R.M., Lesser, V.R.: AgentSpeak(XL): Efficient intention selection in BDI agents via decision-theoretic task scheduling. In: Proceedings of the First International Conference on Autonomous Agents and Multiagent Systems (AAMAS'02). pp. 1294–1302 (2002)
3. Bordini, R.H., Dastani, M., Dix, J., Fallah-Seghrouchni, A.E. (eds.): Multi-Agent Programming: Languages, Platforms and Applications, Multiagent Systems, Artificial Societies, and Simulated Organizations, vol. 15. Springer (2005)
4. Bordini, R.H., Moreira, Á.F.: Proving BDI properties of agent-oriented programming languages: The asymmetry thesis principles in AgentSpeak(L). Annals of Mathematics and Artificial Intelligence 42(1–3), 197–226 (Sep 2004), special Issue on Computational Logic in Multi-Agent Systems
5. Dastani, M., Fallah-Seghrouchni, A.E., Ricci, A., Winikoff, M. (eds.): Programming Multi-Agent Systems, 5th International Workshop, ProMAS 2007, Honolulu, HI, USA, May 15, 2007, Revised and Invited Papers, Lecture Notes in Computer Science, vol. 4908. Springer (2008)
6. Dastani, M., Hobo, D., Meyer, J.J.C.: Practical Extensions in Agent Programming Languages. In: Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS '07). pp. 1–3. ACM, New York, NY, USA (2007), `www.cs.uu.nl/docs/vakken/map/2aplposter.pdf`
7. Decker, K.S., Lesser, V.R.: Quantitative modeling of complex environments. International Journal of Intelligent Systems in Accounting, Finance and Management 2, 215–234 (1993)

8. Dennis, L.A., Farwer, B., Bordini, R.H., Fisher, M., Wooldridge, M.: A common semantic basis for bdi languages. In: Dastani et al. [5], pp. 124–139
9. Fallah-Seghrouchni, A.E., Suna, A.: Claim and sympa: A programming environment for intelligent and mobile agents. In: Bordini et al. [3], pp. 95–122
10. Georgeff, M.P., Lansky, A.L.: Procedural knowledge. Proceedings of the IEEE, Special Issue on Knowledge Representation 74(10), 1383–1398 (1986)
11. Hindriks, K.V.: Modules as policy-based intentions: Modular agent programming in goal. In: Dastani et al. [5], pp. 156–171
12. Huber, M.J.: JAM: a BDI-theoretic mobile agent architecture. In: Proceedings of the Third Annual Conference on Autonomous Agents (AGENTS'99). pp. 236–243 (1999)
13. Leite, J.A., Alferes, J.J., Pereira, L.M.: Minerva - a dynamic logic programming agent architecture. In: Meyer, J.J.C., Tambe, M. (eds.) ATAL. Lecture Notes in Computer Science, vol. 2333, pp. 141–157. Springer (2001)
14. Moreira, Á.F., Vieira, R., Bordini, R.H.: Extending the operational semantics of a BDI agent-oriented programming language for introducing speech-act based communication. In: Leite, J., Omicini, A., Sterling, L., Torroni, P. (eds.) Declarative Agent Languages and Technologies, Proc. of the First Int. Workshop (DALT-03), held with AAMAS-03, 15 July, 2003, Melbourne, Australia. pp. 135–154. No. 2990 in LNAI, Springer-Verlag, Berlin (2004)
15. Morley, D., Myers, K.: The SPARK agent framework. In: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'04). pp. 714–721 (2004)
16. Morley, D.N., Myers, K.L.: The spark agent framework. In: AAMAS. pp. 714–721. IEEE Computer Society (2004)
17. Pokahr, A., Braubach, L., Lamersdorf, W.: Jadex: A bdi reasoning engine. In: Bordini et al. [3], pp. 149–174
18. Rao, A.S.: Agentspeak(l): BDI agents speak out in a logical computable language. In: MAAMAW'96: Proceedings of the 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World: Agents Breaking Away. pp. 42–55 (1996)
19. Vieira, R., Moreira, Á.F., Wooldridge, M., Bordini, R.H.: On the formal semantics of speech-act based communication in an agent-oriented programming language. J. Artif. Intell. Res. (JAIR) 29, 221–267 (2007)
20. Vikhorev, K., Alechina, N., Logan, B.: The ARTS real-time agent architecture. In: Dastani, M., El Fallah Segrouchni, A., Leite, J., Torroni, P. (eds.) Languages, Methodologies, and Development Tools for Multi-Agent Systems, Second International Workshop, LADS 2009, Torino, Italy, September 7-9, 2009, Revised Selected Papers, Lecture Notes in Computer Science, vol. 6039, pp. 1–15. Springer Berlin / Heidelberg (2010), http://dx.doi.org/10.1007/978-3-642-13338-1_1
21. Vikhorev, K., Alechina, N., Logan, B.: Agent programming with priorities and deadlines. In: Proceedings AAMAS 2011 (2011)
22. Vincent, R., Horling, B., Lesser, V., Wagner, T.: Implementing soft real-time agent control. In: Proceedings of the Fifth International Conference on Autonomous Agents (AGENTS'01). pp. 355–362 (2001)
23. Wagner, T., Garvey, A., Lesser, V.: Criteria-directed heuristic task scheduling. International Journal of Approximate Reasoning 19, 91–118 (1998)
24. Winikoff, M., Padgham, L., Harland, J., Thangarajah, J.: Declarative & procedural goals in intelligent agent systems. In: Fensel, D., Giunchiglia, F., McGuinness, D.L., Williams, M.A. (eds.) KR. pp. 470–481. Morgan Kaufmann (2002)