Auton Agent Multi-Agent Syst (2014) 28:682–719
DOI 10.1007/s10458-013-9238-9

An operational semantics for the goal life-cycle in BDI agents

James Harland · David N. Morley · John Thangarajah · Neil Yorke-Smith

Published online: 2 October 2013

Abstract A fundamental feature of intelligent agents is their ability to deliberate over their goals. Operating in an environment that may change in unpredictable ways, an agent needs to regularly evaluate whether its current set of goals is the most appropriate set to pursue. The management of goals is thus a key aspect of an agent's architecture. Focusing on BDI agents, we consider the various types of goals studied in the literature, including both achievement and maintenance goals. We develop a detailed description of goal states (such as whether goals have been suspended or not), and a comprehensive suite of operations that may be applied to goals (including dropping, aborting, suspending and resuming them). We provide an operational semantics corresponding to this detailed description in an abstract agent language (CAN), and demonstrate on a detailed real-life scenario. The three key contributions of our generic framework for goal states and transitions are (1) to encompass both goals of accomplishment and rich goals of monitoring, (2) to provide the first specification of abort and suspend for all the common goal types, and (3) to account for plan execution as well as the dynamics of subgoaling. Our semantics clarifies how an agent can manage its goals, based on the decisions that it chooses to make, and further provides a foundation for correctness verification of agent behaviour.

J. Harland · J. Thangarajah
RMIT University, Melbourne, Australia
e-mail: james.harland@rmit.edu.au

J. Thangarajah
e-mail: johnt@rmit.edu.au

D. N. Morley
SRI International, Menlo Park, CA, USA
e-mail: morley@ai.sri.com

N. Yorke-Smith (✉)
American University of Beirut, Beirut, Lebanon
e-mail: nysmith@aub.edu.lb

N. Yorke-Smith
University of Cambridge, Cambridge, UK

Keywords BDI agents · Goal management · Operational semantics

1 Introduction

Intelligent agent-based systems have been used to develop software applications embedded in complex dynamic environments as diverse as air traffic control, automated manufacturing, electronic auctions, Mars rovers, and personalized museum guides [18,27]. Agent systems of this kind are often designed and structured in terms of *mental attitudes*, such as *beliefs, desires* and *intentions* (BDI) [29] which allows a natural specification of sophisticated software systems in terms that are similar to human understanding.

Goals are a central aspect of BDI agent systems. Goals are a concrete realisation of the mental attitude of desires, and they provide a natural link to human design methods (such as business rules) [53]. The identification of goals naturally leads to iteratively asking questions such as "Why?", "How?", and "How else?". Further, goal decomposition leads to teasing out many levels of requirements and helps to clarify and refine requirements. This is particularly the case for non-functional requirements (such as saving as many humans as possible, or minimizing the time spent on a particular task), which are initially difficult to make precise, but can be specified as abstract goals to be refined later. We restrict ourselves to functional requirements in this article.

A distinction can be made between system design goals and goals that are part of the mental state of an agent at runtime. The former are specified a priori by the designer of the agent system, and are usually static. In fact, the examples of goals just given are system design goals. The latter goals, those part of the agent's mental state, are considered and adopted dynamically by the agent, such as a goal to explore a certain area, or to recharge a battery. Our focus is on this latter type of goal in this article.

Due to the centrality of goals in agent systems, it is common to thus ascribe a set of goals to an agent, which is equipped with various techniques to deliberate over and manage this set. In order to free the designer to concentrate on high-level behaviours, agents designed to work in dynamic environments must be able to reason about what actions they should take. They must incorporate deliberation into their execution cycle so that decisions can be reviewed and corrective action taken with an appropriate focus and frequency.

The centrality of reasoning over goals is seen in the literature. Aspects investigated include subgoaling and plan selection [35], detection and resolution of conflicts [32,41,43] or opportunities for cooperation [41,44], checking goal properties to specification [25,45], failure recovery and planning [5,15,30,34], and suspending and resuming [37], or dropping or aborting goals [36].

A variety of goals are described in the literature, including goals of *performance* of a task, *achievement* of a state, *querying* truth of a statement, *testing* veracity of beliefs, and *maintenance* of a condition [7,49]. An agent must manage such a variety of goals, while incorporating pertinent sources of information into its decisions over them, such as preferences, quality goals, motivational goals, and advice [26,45]. The complexity of agent goal management—which stems from this combination of the variety of goals and the breadth of deliberation considerations—is made more complex because each goal may be dropped, aborted, suspended, resumed or fail at arbitrary times. While goal properties are usually static (i.e., they are specified at design time in almost all systems and do not change during execution, although in principle they could change), crucially their behaviour is dynamic: a goal may undergo a variety of changes of state during its execution cycle [25]. This evolution may include its initial adoption by the agent, being actively pursued, being suspended and then

later resumed, and eventually succeeding (or failing). A goal may also be arbitrarily aborted or dropped by the agent. (Maintenance goals have a subtle life-cycle: the goal is retained even when the desired property is true; it is possible that such goals are never dropped.)

This article presents a systematic analysis of the behaviour of the above types of goals. We consider the complete life-cycle of goals, from their initial adoption by the agent to the time when they are no longer of interest, and all stages in between, including being suspended and resumed.

1.1 Scenario

To illustrate some of the complexity and richness of goal deliberation and management, we take running examples from a real-life domain. We will detail the domain and demonstrate our semantics in the sequel.

A Mars rover has landed on the planet's surface. This semi-autonomous robotic agent has initial tasks to perform, such as testing communications. Once diagnostics are successful, the rover will pursue a dynamic set of science experiments. More precisely, each Martian day (sol) it receives updates from Earth-based mission control on its goals, and it reports its results to Earth at the end of each sol. The specific experiments to perform include, for example, taking soil samples and performing various analyses with its built-in instruments. These experiments take a certain amount of time, and involve a specific predicted resource cost (such as energy). Further, resources may need to be reserved for communication, in particular sending results to Earth at the end of the sol. This requires proactive management, such as de-prioritizing tasks which will violate such resource requirements. The rover has the freedom to act proactively, and certain events, such as observing a green rock [4] or unusually high moisture content, trigger the rover to explore such 'science opportunities'. To pursue them may mean that the rover suspends any experiments or analyses that it is currently undertaking. Besides its other responsibilities, the rover will also need to consider maintenance conditions, such as maintaining a certain minimum amount of charge, or never moving out of communication range with the lander.

1.2 Contribution

Our work extends previous efforts in three directions. These contributions come in two areas. Our first area of innovation is to develop a rich and detailed specification of the appropriate operational behaviour when a goal is pursued, succeeded or failed, aborted, suspended, or resumed. Our first contribution is to include sophisticated maintenance goals, along the lines of Duff et al. [16], that encompass proactive behaviour (i.e., anticipating failure of a given condition) as well as reactive behaviour (i.e., waiting until the condition becomes false), and allow for different responses in each case. This contrasts over most work on maintenance goals, in which only the reactive behaviour is developed [25,49].

In the same area, our second contribution is to develop an appropriate set of *states* for goals (which generalizes the two states of suspended and active of van Riemsdijk et al. [49]), and a set of operations to move goals between these states. These operations are richer than previous works, by including suspending and resuming for all the common goal types, and the corresponding state transitions which can be non-trivial. We provide a detailed specification and a formal operational semantics.

Our second area of innovation is to address execution of plans to achieve goals within our semantics. The spirit of our work is shared by Morandini et al. [25], who build on van Riemsdijk et al. [49] by providing operational semantics for non-leaf goals, i.e., semantics

for subgoaling and goal achievement conditions. In this area, our third contribution is to encompass the same dynamic execution behaviour, but further consider plans as well as goals. Thus we consider the execution cycle, not only the design phase like Morandini et al. By expressing the formal operational semantics for our generic framework in the agent language CAN [31], we remain independent of the specifics of particular agent implementation. We have developed a prototype implementation of the CAN rules of the semantics, in Prolog, as a proof of concept. Our semantics clarifies how an agent can manage its goals, based on the decisions that it chooses to make, and further provides a foundation for correctness verification of agent behaviour.

1.3 Previous work

A preliminary version of this work was discussed at the DALT'10 workshop [39,40], and a brief summary appeared at the ECAI'10 conference [38]. This article expands the exposition throughout, formalizes the concept of plans, extensively reworks the operational semantics, extends the discussion of related work, and includes a significantly more detailed and formalized scenario. The new or extensively revised material approximates 50 % of the article.

1.4 Organization

The article is organized as follows. In Sect. 2 we provide background, including the introduction of various types of goals, and then in Sect. 3 describe the Mars rover domain. In Sect. 4 we specify goal management behaviours, in terms of a life-cycle of goal states and the transitions between them. In Sects. 5 and 6 we develop the operational semantics for our proposal and illustrate with a worked example. In Sect. 7 we discuss related work, and in Sect. 8 we present conclusions and directions for further work.

2 Background

In this section we give necessary background. We define the notion of goals and review mechanisms for goal suspension and resumption, enumerate the types of goals, and define the notion of plans that we will use.

2.1 Goals and goal manipulation

The popular *Belief-Desire-Intention* model of agency ascribes a set of propositional attitudes to the agent: traditionally, BDI [29]. In the BDI and other contemporary models of agency designed for cognitive agents, goals and their management hold a central role [11,49].

Winikoff et al. [52] argue for the importance of both declarative and procedural representations of goals. We follow their syntax of goals, using the above robot rescue scenario as a running example. Goals have a specification with both declarative and procedural aspects. We suppose that a goal g has a *context* (or *precondition*) that is a necessary condition before the goal may be adopted, a *success condition* S that denotes when the goal may be considered to have succeeded, and a *failure condition* F that denotes when it may be considered to have failed. The precondition of a goal can be seen as stating when the goal is applicable. It may be empty, i.e., always true. We say that a goal is *applicable* if its context is true, and *not applicable* otherwise.

For example, consider an autonomous rover on Mars. As it operates best during sunlight hours, the rover may be required to return to its base at dusk. This can be achieved by having a goal that requires the rover to return with a precondition that it is nearly dusk. When the time is reached, this goal is applicable, and hence becomes active. Similarly, 'science opportunity' goals, such as investigating a particular type of landscape if it is observed, can be modelled by an appropriate use of preconditions. Further discussion on these points is in Sects. 3 and 6.

2.1.1 Mechanisms for abort, suspend, and resume

Thangarajah et al. studied mechanisms by which an agent can correctly drop or abort [36], or suspend and resume [37] goals and plans. Since our treatment of the goal life-cycle allows for the agent performing these operations on its goals, we provide a brief summary of that work in order to make this article self-contained. We describe primarily the mechanisms for goals; those for plans are similar. For details, we refer to the original papers. A detailed understanding of the operation of the mechanisms is not required for this article.

First, should an agent decide to *drop* a goal, it simply does so. The goal and any plans related to it are halted; the goal is discarded with no clean-up and no further action. Second, by contrast, should an agent decide to *abort* a goal, the agent performs any clean-up actions that might be required, before dropping the goal. These clean-up actions are described in the goal's *abort method*. For example, the rover notifies mission control that it is abandoning some science goal, before it drops the goal. Aborting a goal thus given opportunity to ensure that any plans being executed for it are properly aborted, rather than being immediately discarded.

Third, should an agent decide to *suspend* a goal, it executes the *suspend method* attached to the goal. This will suspend any plans in execution for it. When suspended, a goal waits until the agent decides that it should resume (the agent may also decide to drop or abort a suspended goal). To do so, it executes any *resume method* attached to the goal. The change in world state since the goal was suspended means that plans that were in execution may or may not be valid. It is the responsibility of the resume method either to check and re-activate any previously suspended plans, or to drop them and restart the goal.

It is worth noting that suspend and resume methods, like abort methods, are assumed not to fail [37]. We also note that these methods, provided by the agent designer, are optional: the agent has default methods that apply if not overridden. Again, for details we refer to the cited works.

Finally, the decision of an agent to change the state of a goal will be indicated by an appropriate addition to its beliefs (as described below). This means that, for example, an agent's decision to suspend a goal g will be made apparent by the agent adding a formula $suspend(g)$ to its beliefs. The advantage of this approach is that it makes it straightforward to combine such top-level decisions (see Sect. 4.3) with other changes to be made, such as suspending a subgoal when its parent goal is suspended.

2.2 Goal types

Braubach et al. [7] are among those who survey the types of goals found in agent systems. The consensus in the literature agrees that perform, achieve, query, test, and maintain cover the widespread uses of goals [7,13,49,52].

We observe that querying and testing goals can be reduced to achievement and performance goals, respectively [49], and do not dwell on them. As noted, often a query goal reduces to

a simple test on the state of the agent [49]. This is why some agent programming languages, which have a construct for testing, do not call this a test goal. We remark that, as constructed in some languages, query goals can unify variables in plans.

We note the additional and compound goal types proposed in by Winikoff et al. [51]. These goal types are defined using Linear Temporal Logic (LTL). Chief among them is an 'achieve and maintain' goal type. Extending our work to such compound goals is reserved for the future.

Next we consider the goal types achieve and perform, and then goal type maintain. Let **B** denote the beliefs of the agent, represented for example as a set of formulae over a language ℓ. Let $\models : \ell \rightarrow \ell$ be an implication operator with the natural extension to sets of formulae, such as **B**, treating them as a conjunction. We assume the existence of belief update operators, which we will write as $\mathbf{B} \cup \{\phi\}$ and $\mathbf{B} \backslash \{\phi\}$ and which make 'minimal' changes to the belief sets such that $\mathbf{B} \cup \{\phi\} \models \phi$ and $\mathbf{B} \backslash \{\phi\} \not\models \phi$. The precise form of **B**, ℓ, and the update operators will not be central to our semantics. Our examples assume that standard first-order logic will be used; richer logics such as epistemic logics could be used, but apart from a minimum class of formulae that can be used as beliefs, the precise nature of the logic used is immaterial.

achieve(κ, S, F): *reach a state* S These goals, sometimes called *goals-to-be*, demand that some state of the world S be reached. The goal is applicable if its *context* κ is true. When the goal is applicable, an agent may adopt it, whereupon to reach S, the agent generates and executes plans. The goal should not be dropped until the state is achieved or is found to be unachievable, signified by the failure condition F. κ, S, and F are drawn from the language ℓ as **B**.

An achieve goal checks its success condition during plan execution and after a plan completes. If the success condition S is true (at any point during execution), the goal terminates successfully; if the failure condition F is true (at any point during execution), the goal terminates with failure. Otherwise, the goal must return to plan selection or generation, even if the previous plan completed successfully.

Example Task the rover to explore a rock called `Soufflé`. Formally, the goal is:

$$\text{achieve}(dawn, explored(\texttt{Soufflé}), dusk \wedge \neg explored(\texttt{Soufflé}))$$

where $dawn$ and $dusk$ are conditions about the Martian day, and $explored(X)$ denotes the completed exploration of Martian feature X.

Performance goals, sometimes called *goals-to-do*, demand that at least one plan in a given set of plans be successfully executed. They do not require any particular state of the world be achieved. A perform goal succeeds if one or more of its plans complete execution; it fails otherwise, such as if no plan is applicable or all applicable plans fail to execute. Hence, the difference between a performance goal and an achievement goal is that in the former the agent aims to accomplish some plan, while in the latter the agents aims to reach some state.

We observe that performance goals can be modelled using achievement goals, by means of a success condition that is the success of one or more of the plans. At the same time, it is worth noting that achievement goals can be simulated by performance goals provided the agent language has a mechanism to check whether a particular belief is true [47]. Hence, although perform goals are useful in practical agent programming languages, in the sequel we distinguish between two types of goals: achievement and maintenance.

Achievement goals are *goals of accomplishment*: they all directly result in activity. Maintenance goals, by contrast, are *goals of monitoring*, in that they may give rise to other goals

when particular triggering conditions are met, but they do not themselves directly cause activity. We remark further on the achievement–maintenance dichotomy in Sect. 7.

Let $\pi : \ell \times \mathbf{B} \rightarrow \{\top, \bot\}$ be some prediction mechanism, for example using lookahead reasoning (e.g., [20,44]). The prediction mechanism of an agent predicts whether a given formula will be true in the future, in light of its current beliefs. We leave the details of the temporal extent of π to previous works [3,16]. We will abbreviate $\pi(p, \mathbf{B})$ where $p \in \ell$ as $\pi(p)$.

We can now define a maintenance goal as follows.

maintain(κ, C, R, P, S, F)*: keep a condition C true* Maintenance goals monitor a *maintenance condition*, C, initiating a *recovery goal* (R or P: see below) to restore the condition to true when it becomes false. The maintenance condition is specified by the agent designer. The maintenance goal is applicable if its context κ is true. S and F are the success and failure conditions, respectively.

Note that a recovery goal is initiated, not a plan. More precisely, as introduced by Duff et al. [16], we allow a maintain goal to be *reactive*, waiting until the maintenance condition is found to be false, i.e., $\mathbf{B} \models \neg C$, and then acting to restore it by adopting a reactive recovery goal R; or to be *proactive*, waiting until the condition is *predicted* to become false, i.e., $\mathbf{B} \models \pi(\neg C)$ and then acting to prevent it by adopting a proactive preventative goal P. Although not specified in prior work, for simplicity and clarity we will insist that R and P be achieve goals. The maintenance goal continues until either the success condition S or failure condition F become true, whereupon it may be dropped.

Duff et al. specified R and P explicitly in their formulation of maintenance goal. We follow their work, noting that the agent designer might want to specify a goal R (respectively, P) to be triggered when $\mathbf{B} \models \neg C$ (respectively, $\mathbf{B} \models \pi(\neg C)$). Of course, the designer could specify R to have its success condition as the achievement of C (similarly for P) as a default behaviour. Our approach in this article is agnostic on how the agent finds a plan (or sequence of plans) to achieve a particular goal. In particular, this applies to R and P. The agent could look up plans in a plan library, or generate plans at run-time, or both, or neither.

Example Ensure that the rover is always adequately charged. Formally, the goal is:

$$\mathsf{maintain}(rover\_online, battery\_level \geq 10\,\%, R, P, false, battery\_failure)$$

where $battery\_level$ denotes the rover's present charge level, and 10 % is a threshold level above the critical low charge level. The recovery goal R is achieve($true, b, a$)$ttery\_level \geq 30\,\%battery\_failure$. The preventative goal P is achieve($true, charge \geq 60\,\%, battery$)$\_failure$. This maintenance goal is persistent for the life of the rover: hence it has no success condition (it continues for ever) and no failure condition other than hardware failure (indeed, if it fails, the rover is left at a critical low charge, whereupon emergency programming will override the agent executive).

We remark that in future work, it could be interesting to define an alternative semantics for proactive maintenance goals, namely to try R should P fail, and only drop the maintenance goal should R also fail. Such behaviour could be achieved with the current semantics of Duff et al., by programming the agent's plan to achieve P as: try to prevent violation of maintenance condition C, and if that fails, then try to restore it.

2.3 Plans

A goal states the outcome to be achieved, but not how to do it. A *plan* specifies how to achieve a goal. Put another way, the procedural aspect of goals is represented by plans,

which describe ways of realizing the goals. Since we have focus on the two goal types, achievement goals and maintenance goals, and since maintenance goals trigger plans indirectly through achievement R or P goals, we only need consider plans for achievement goals.

Many BDI-agent systems specify a particular mechanism for finding a plan to achieve a given achievement goal. These often involve *triggers* (a change of state that causes the plan to be performed), and a context or precondition that specifies the conditions under which the plan is applicable. Instead, we simply assume that given an achievement goal, we can abstract away the process of means-end reasoning to derive a plan using a procedure *mer*, which is used to associate plans with goals. As there may be several possible choices of plan for a given goal, depending on the agents beliefs and the agent's other goals, *mer* takes as input the given goal, the agent's beliefs, and the agent's goals.

For our purposes, a plan consist primarily of a *plan body*, a specification of things to do. Because of this, we will often use the terms plan and plan body interchangeably. A plan body consists of a combination of goals and atomic actions that are performed to achieve an achievement goal. When executed, a plan may either succeed by running to completion, or it may fail at some point.

While a plan does not itself have success and failure conditions, as these are specified by the achievement goal, a plan may have procedural success and failure *methods* (i.e., subplans which must not fail) that are invoked upon its success and failure respectively [52]. A plan may have other dedicated procedural methods attached, such as an abort clean-up method [36], and suspend and resume methods [37].

In order to evaluate the conditions used in plans (and other formulae), it is necessary to have an appropriate reasoning system, so that we can evaluate whether the agent's beliefs imply a given condition or formula. The precise specification of such a reasoning system is not required for the purposes of this article, but it would be reasonable to assume that the conditions used in plans are a subset of the formulae that can be used in the language of the beliefs **B** of the agent, so that such conditions can be evaluated. First-order classical logic will suffice for the examples used in this article.

We will use the CAN language to describe plan bodies [31]. We introduce CAN more widely in Sect. 5 where we use it for our operational semantics. A *plan body* consists of any of the following:

- An achievement goal as described above.
- A request to perform a primitive, atomic action a. To simplify the semantics, we assume that the success or failure of an action can be accurately predicated using a precondition predicate, $pre(a)$. The precondition is such that execution of the action will succeed if the precondition follows from the beliefs of the agent at the time the action is performed and will fail otherwise.
- The nil plan body always succeeds.
- The $fail$ plan body always fails.
- The belief update plan bodies $+\phi$ and $-\phi$, where $\phi \in \ell$ is a condition in the beliefs language, which change the agent's beliefs **B** by applying the belief update operators to make the condition ϕ true or false respectively.
- The test plan body $?\phi$, where ϕ is some condition, will succeed if ϕ is true at the time $?\phi$ is executed and fail if it is not.
- The wait plan body $\phi : P$, where ϕ is a condition and P is a plan body, will delay execution of P until ϕ is true.

- The sequential construct $P_1; P_2$, where P_1 and P_2 are plan bodies, executes P_1 and if that completes successfully, executes P_2. If either of those fail, then the sequential combination fails.
- The parallel construct $P_1 \| P_2$, where P_1 and P_2 are plan bodies, concurrently executes the two subplans.
- The 'or-else' construct $P_1 \triangleright P_2$, where P_1 and P_2 are plan bodies, first executes P_1 and then, only if P_1 fails executes P_2.
- The construct $(\!|\phi_1 : P_1, \phi_2 : P_2, \ldots |\!)$ contains a set of plan bodies guarded by conditions. Execution of this construct selects a guard ϕ_i that is currently true and executes the corresponding P_i, if that fails, a different guard condition ϕ_j that is now true is selected and P_j executed, and so on until there are no remaining guard conditions that are true, at which point it fails.

For example, the plan body consisting of three subgoals:

$$\begin{aligned}&\mathsf{achieve}(true, Loc(\texttt{rock1}), false);\\ &\mathsf{achieve}(true, SampleCollected, false);\\ &\mathsf{achieve}(true, SampleAnalysed, false)\end{aligned}$$

will sequentially create subgoals to move the rover to the location `rock1`, collect a sample, and analyse it.

It should also be noted that we have not included CAN's 'event plan body', which adds an event to the agent's beliefs, and which is commonly used in conjunction with the above plan constructs. We will discuss this further in Sect. 5.

Finally, recall that we do not specify any particular mechanism for associating a plan with an achievement goal. As noted earlier, we only assume that an agent has some mechanism that generates a plan for a given achievement goal.

3 Scenario domain

In 1997, the US space agency NASA landed the robotic rover *Sojourner* on Mars. This was the first successful US lander since the Viking missions of the 1970s, and the first successful Mars rover (although the Soviets were close in 1971 [1]). In 2004, NASA landed the pair of rovers, *Spirit* and *Opportunity*. Like, Sojourner, Spirit and Opportunity could travel up to 100 m per sol (Martian day). Communication with Earth is intermittent (the Deep Space Network communication system is only available a few times during each sol) and delayed (round trip communication time between Earth and Mars is between 8 and 42 min). Hence, since there is no prospect of teleoperation control, the rovers were tasked with higher-level command sequences and endowed with some autonomous wayfinding capability and other functions [2].

More recently, NASA landed the rover *Curiosity* on Mars in 2012. *Curiosity* is a step forward in terms of size, capability, and autonomy. It is powered by a nuclear radioisotope thermoelectric generator rather than by solar energy, for instance, and can travel up to 200 m per day, and has greater processing power, storage, and scientific capability [22].

One motivation for increased autonomy is greater robustness. "Several high-level goals and decisions could be taken into the work environment rather than made on Earth" state Siebra et al. [33]. Consider the example human error in specifying a (high-level) command sequence, which would result in the rover being outside of communications range with its lander. In this case, a maintenance goal can automatically trigger, causing the rover driving

back towards the lander [33]. Further, increased autonomy gives increased robustness and agility in responding to unexpected events.

Bordini et al. [4] summarize a scenario inspired by a real incident that occurred with *Sojourner* [50]. The rover was commanded the following high-level sequence for one sol:

1. Back up to the rock named Soufflé;
2. Place the arm with the spectrometer on the rock;
3. Do extensive measurements on the rock surface;
4. Perform a long traverse to another rock.

"In this particular sol operation, it turned out that the rover did not position itself correctly to approach the rock with the spectrometer arm. The misplaced spectrometer meant that no useful data was collected, and that particular rock could not be visited again, hence a science opportunity was lost." The authors point out the need for flexibility in rover operation.

Further, "an important kind of unexpected event is the detection of science events. The rover planner needs to decide if it will respond autonomously to such events, adding them to its agenda. If the rover is not prepared to deal with this new event, questions raised as a result of its analysis can be explicitly sent for additional analysis by ground teams" [33]. Hence, another motivation for increased autonomy is the amount of science that can be achieved.

The stated medium-term goal of NASA is to send manned human missions to Mars [21,33]. The intention is to progress from single agent rovers (current art), to multi-agent rover teams (future), to human and multi-agent teams working together on Mars (eventual). Thus, intelligent agent techniques are both warranted and required for deployed rovers.

3.1 Curiosity scenario

We use the Mars rover domain [4,17,50] as a running example in this article, culminating in a detailed execution trace of our semantics on a goal-rich vignette. We consider a single rover like *Curiosity*. The rover only operates during the Martian daytime, as its cameras and imaging devices function in light; however, because of its nuclear power, the rover is not dependent on sunlight or stored charge for communication. Each sol the rover receives new high-level goals from mission control. It has autonomy to prioritize among goals of the same priority. The first main type of achievement goal is movement (go to a location), which the rover achieves by lower-level waypoint planning and movement. The second type of goal is science (analyze a rock, obtain and analyze a soil sample, take imagery, etc). The third type of goal is communication (upload data and results). In addition, the rover is on the lookout for 'science opportunities', as described earlier. An important mission goal is investigating whether environmental conditions have ever been favourable for microbial life.

Besides achievement goals, the rover has various maintenance activities to perform, either autonomously or as specifically tasked. These can include to stay within a certain area, stay within communication range of the lander or orbiters, maintain within tilt limits, maintain battery levels, keep within temperature bounds, and perform daily maintenance activities.

The rover's resource include active and passive heating and cooling capabilities, a pair of batteries, a nuclear power source, multiple communication antenna (low and high x-band which can both communicate with Earth, and UHF which can communicate with relay orbiters), a robotic arm with sensors and a drill bit, multiple imaging devices (hazard avoidance cameras, navigation cameras, wide-angle mast camera, cameras on its robotic arms, etc), a suite of scientific instruments (spectroscopy, meteorological, radiometer, x-ray diffraction), and flash storage memory. Different activities may require different resources (instruments,

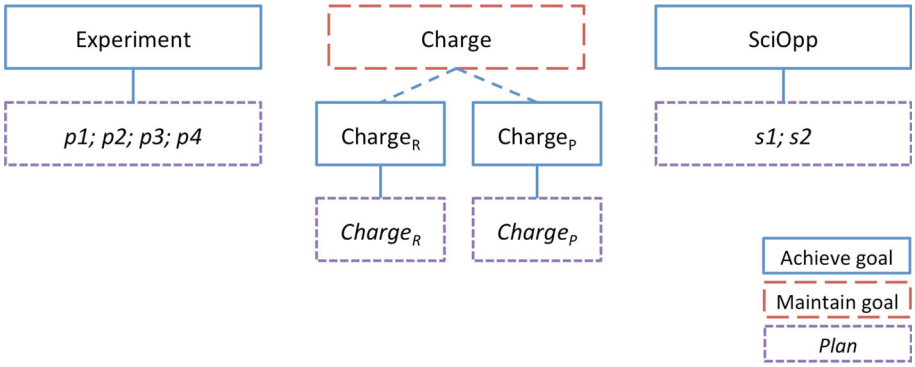

Fig. 1 Goal-plan trees for basic *Curiosity* scenario

communication), generate different amounts of heat, and have energy and storage requirements.

Clearly, modelling the complete functionality of *Curiosity* is a task of considerable complexity; in this article, we adopt a set of simplified examples based on the following scenario. *Curiosity* has been tasked with an achievement goal $Experiment$, which will involve it moving to a specific location and performing particular analyses on what it finds there. *Curiosity* has a maintenance goal $Charge$ which ensures that its batteries contain a certain minimum level of charge, and a science opportunity goal $SciOpp$, which is triggered when its mast or arm cameras detect a particular colour of rock, which it will then approach and investigate for signs of microbial life. The basic operation of *Curiosity* will thus involve moving to a specific location to carry out experiments and communicating their results to Earth, recharging batteries when necessary, and watching for opportunities to pursue its science opportunity goal.

There are many ways in which this simplified scenario could be extended, such as adding concurrent goals of achievement, more maintenance goals (such as temperature control, or requiring *Curiosity* to roam within a certain area), and 'scheduled' goals, such as communicating with Earth at a certain time or performing regular maintenance tasks. Section 6 examines this scenario in detail as a worked example of our operational semantics.

Figure 1 depicts *goal-plan trees* [41] for the scenario. A goal-plan tree (GPT) consists of alternating levels of goal nodes and plan or primitive action nodes. The default semantics of child nodes of a goal node is OR: that is, the goal is satisfied if any one of the child plan nodes succeeds. The default semantics of child nodes of a plan node is AND: that is, the plan succeeds if all of its child goal nodes are satisfied.

In the *Curiosity* scenario, the robot has three top-level goals, $Experiment$, $Charge$ (a maintenance goal), and $SciOpp$ described earlier. We can see the plan that can achieve $Experiment$, a four-step sequence, and the plan that can achieve $SciOpp$, a two-step sequence. Being a maintenance goal, $Charge$ has a different semantics. The dashed lines indicate that its children are adopted as necessary to maintain the goal's maintenance condition.

4 Goals states and transitions

We now move towards a characterization of goal states and the transitions a goal undergoes between these states. Our focus is the life-cycle of each particular goal that the agent has.

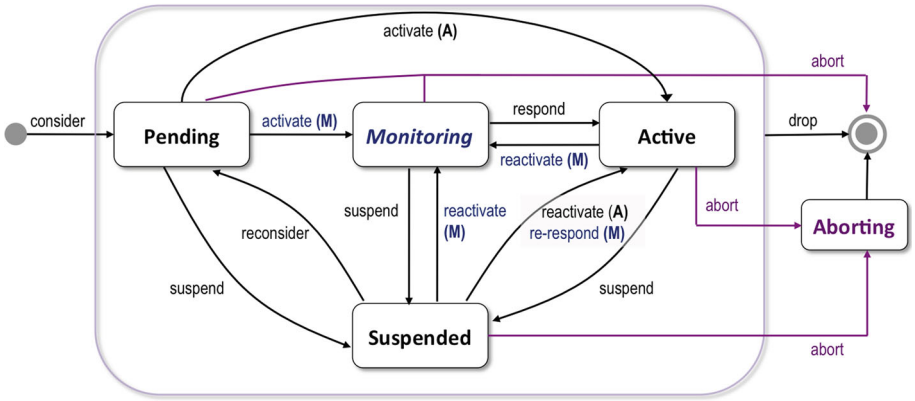

Fig. 2 Goal life-cycle composed of abstract states. A—achieve goal, M— maintain goal

Hence, our perspective is that of an individual goal, rather than the overall agent per se. This means that we will not be concerned with issues such as the agent's overall deliberation, generation of goals (see, e.g., [11]), prioritization of goals, or reasoning behind goal decisions (e.g., the reason *why* to suspend or resume a goal). These relevant topics are outside the scope of this article.

4.1 Abstract goal states

Our objective is to specify the life-cycle of goals and the mechanisms of the agent on its goals, i.e., how the agent can manipulate its goals. The life-cycle we capture as four states, Pending, Monitoring, Active, and Suspended, shown in Fig. 2, together with the auxiliary state Aborting, the initial state (left), and the terminal state (right).

The drop transition is depicted by a single arrow from the outer enclosing box to the terminal state. This single arrow indicates that the same transition to the terminal state occurs from each of the four states within the enclosing box. As explained below, states Pending and Monitoring share the same abort transition directly to the terminal state, while states Active and Suspended both transition to Aborting by their abort transitions. Note that transitions for succeed and fail are not shown explicitly, because when a goal succeeds or fails, it is dropped, i.e., it follows the drop transition.

Some transitions are essentially controlled by conditions, while others depend on an explicit agent decision (or a combination of conditions and a decision), as will be made precise in our formal semantics.

A new candidate goal may arise from a source external or internal to the agent's control cycle [29]. External to the control cycle, it may arise from obligations or commitments concerning other agents, or from the agent's own motivations. Internal to the control cycle, it may arise from subgoaling within an executing plan. Either way, a new candidate goal begins life in the Pending state once the agent has decided to consider the goal.

It should be noted that the states and transitions in Fig. 2 refer to goals only. In general, when there is at least one goal in the Active state, there will be at least one plan executing. Note that plan execution only takes place when there are no changes in goal states, so that any change in the state of any goal takes preference over any executing plan. This point is made explicit in our semantics (see Sect. 5).

In the next section we overview the possible operations on goals. Then we describe the goal control cycle in detail, including the mechanisms to perform the goal operations of interest.

4.2 Goal operations

Goal operations are the mechanisms of an agent on its goals, i.e., how the agent can manipulate its goals. They correspond to transitions between states, as we detail in the next section. The transitions can be seen in Fig. 2.

We first informally summarize the set of goal operations that we consider:

- *consider*. Adopt a goal.
- *activate*. Begin activity on a goal. The nature of 'activity' differs between achieve and maintain goals.
- *respond* (maintain goal). Act to restore or prevent violation of the maintenance condition.
- *suspend*. Pause activity on a goal.
- *reconsider*. Resume (in state Pending) a goal which had no activity when suspended (i.e., was Pending).
- *reactivate* (achieve goal). Resume activity on a suspended goal.
- *reactivate* (maintain goal). Resume monitoring activity on a goal whose maintenance condition is not violated and not predicted to be violated.
- *re-respond* (maintain goal). Resume activity on a suspended goal, whose maintenance condition is violated or is predicted to be violated.
- *abort*. Terminate a goal, performing clean-up operations as may be necessary.
- *drop*. Immediately discard a goal, without performing any clean-up. This includes terminating a goal after success or failure.

4.3 Transitions between states

The heart of our work is the effects of different operations that an agent may apply to its goals of different types, in each of the four basic states introduced. We next describe in detail the life-cycle of goals over these states.

Each operation corresponds to a transition between states. There are two types of such transitions. The first type of transitions are operations that follow from a *top-level command*: a decision by the agent's deliberation to impose an operation upon a goal. For instance, an agent might choose to abort a goal that (otherwise) it might have continued with.

The second type of transitions are operations that are mandated by the life-cycle of the goal or the nature of the agent's design. These transitions are *triggered by conditions*; the agent does not make a deliberate decision. For instance, consider a goal that has a plan attached, which the agent is executing. A common design for an agent implies that if the goal is aborted, the plan should be dropped: there is no deliberative decision required.

Recall that each operation on a goal that results in a change of the goal's state is manifest by an addition to the agent's beliefs about the goal (Sect. 2.1). For example, for a decision to suspend a goal g, the agent adds a formula $suspend(g)$ to its beliefs. Note that it is beyond the scope of this article to consider *how* the agent comes to this conclusion: our focus is on managing the consequences of such decisions, rather than on the reasons for making them.

Before we discuss the goal states, we remark further about dropping goals. To any goal in any state except Aborting, the *drop* operation implies that the goal g and any plans related to the goal are halted; the goal is discarded with no further action. That is, any plans in execution in service of g are stopped immediately and removed from the agent's mental state, and g itself is removed from the agent's mental state. Figure 2 depicts by the grouping oval that the drop operation can apply to any of the four goal states within the oval. The agent may choose to drop a goal if, for example, it believes the goal is accomplished, is no longer required,

impossible, or if it inhibits a higher priority goal. Note that there are three essential cases here: the goal is dropped because it has succeeded, dropped because it has failed, or dropped because the agent has decided to drop it; the last is a top-level command.

4.3.1 Pending state

Goals in the Pending state are inactive, awaiting the agent to deliberate over them and execute a particular operation. The *activate* operation on an achieve goal transitions the goal to the Active state where the goal is pursued. By contrast, the *activate* operation on a maintain goal transitions the goal to the Monitoring state where the agent will monitor the goal's maintenance condition. (The agent does not monitor the condition while the goal is Pending.)

The *suspend* operation takes a goal to the Suspended state. The *abort* operation does not require any clean-up in this case, since no plans for the goal are in execution. Hence in this case, as the abort method is trivial, the goal transitions immediately to the terminal state. Finally, if the success or failure condition of an active goal become true in the Pending state, the goal is dropped.

4.3.2 Monitoring state

The Monitoring state is shown in italics in Fig. 2 to emphasize that it exclusively applies to goals of monitoring: maintain goals that (actively) check for a triggering condition to be known. Recall that this triggering can be the violation of the maintenance condition (reactive) or its predicted violation (proactive). In this state, as in Pending, no plans are being executed.

Maintain goals transition into the Monitoring state when they are (1) activated from Pending, (2) reactivated from Suspended, or (3) reactivated from Active when the recovery or preventative subgoal succeeds. Should the maintenance condition be violated—or, in the proactive case, should it be predicted to be violated—then the goal transitions to the Active state with the *respond* operation. Note that *respond* is not a top-level command but a triggered transition. The *suspend* operation moves the goal to the Suspended state, whilst (as for Pending), *abort* will immediately drop the goal since no plans are in execution and no hence clean-up is required. The goal may also be dropped if the success or failure condition becomes true.

4.3.3 Active state

Goals in the Active state are in pursuit of the performance of a task, the achievement of a state, or the maintenance of a condition. They may therefore have one or more plans associated. In earlier work, we specified the detail of substates within Active and Suspended states [40], thus providing additional detail on how the plan or plans associated with the goal are managed.[1] Here we restrict our attention to the abstract states of Fig. 2.

4.3.3.1 Maintenance goals. Maintain goals enter the Active state in two ways. First, they enter Active from the Monitoring state when the triggering condition is true, via the *respond* operation. In this transition, the agent posts and activates the preventative or recovery achievement subgoal. A maintain goal posts a recovery goal R if the maintenance condition was violated or a preventative goal P if the maintenance condition is predicted to be violated.

[1] Some minor points of clarification can be added to that workshop paper.

Recovery and preventative goals are always achieve goals, and themselves commence in the Active state.[2] Note that no goal selection is required, since in the semantics of maintenance goals we adopt, R and P are fixed and pre-defined. Further, note that a recovery or maintenance goal, when activated by the agent, will have its own instance of the goal life-cycle. That is, the states and transitions shown in Fig. 2 pertain to the parent maintain goal and again for its R or P subgoal. Detail of how the agent manages these subgoals would go beyond the level of detail of Fig. 2; see [40].

The second way that maintain goals enter the Active state is from the Suspended state via the *re-respond* operation. In this transition, the agent posts and activates the preventative or recovery achievement subgoal.

4.3.3.2 Achievement goals. Achieve goals also enter the Active state in two ways. First, they enter Active from the Pending state via the *activate* operation. Second, achieve goals enter Active from the Suspended state via the *reactivate* operation. Either way, the agent will seek an applicable plan for the goal, e.g., by means-ends reasoning.

In the case that no plan can be found for an achievement goal, there are only two options: to drop the goal, or to suspend it. In principle, it is possible to argue for suspending the goal in this circumstance, as it may be possible that a plan may be found later (or that a plan that is currently inapplicable may become so) due to changes in the environment. However, this assumes that the results of the process to find a plan will vary significantly over time, and may result in a large number of suspended goals. Hence it seems simplest to us to drop an achievement goal when no plan can be found for it. Note that this applies only to achievement goals, as there is always a plan for a maintenance goal, namely to adopt the appropriate achievement goal as a subgoal.

4.3.4 Suspended state

The final of the four basic states, Suspended, contains a goal of any type that is suspended, monitoring its reconsideration condition, awaiting possible resumption. Goals transition to this state when the *suspend* operation is applied to them. The suspended goal may have one or more plans associated. Additional details of their management is again given in [40]. The operational semantics in Sect. 5 does not have suspend and resume methods.

Besides from Pending, a maintain goal g can be suspended from the Active state, or from the Monitoring state. In the latter case, since the goal g comes from Monitoring, it can have no plan or subgoal attached.[3] Should the goal be aborted or its success or failure condition become true, it is dropped; otherwise, the goal remains suspended until the agent decides to resume it—most commonly, because its reconsideration condition becomes true.[4]

In more detail, upon the resumption of a goal, the agent's deliberation has several options. The agent may opt to (1) leave the goal suspended (no change of state), (2) *reconsider* the

[2] In our formal semantics, the goals technically commence in Pending but transition immediately to Active due to their activation conditions.

[3] In the former case, the goal can have a plan or subgoal attached. Therefore a goal g suspended from Active should have suspend methods run for g and its subgoals, if any. This ideal is not reflected in our formal semantics, since it is a substate detail described in [40].

[4] That is, *resume* is a top-level command. Hence, the reconsideration condition is a 'note' from the agent to itself to guide its deliberation over the suspended goal: a sufficient but not necessary condition for when the agent should next look at the goal.

goal (back to Pending state, first aborting the subgoal, if any), (3) *reactivate* the goal (back to Monitoring, again aborting the subgoal, if any), (4) *re-respond* by resuming the subgoal (if one exists; otherwise, activate a new subgoal as above in state Active) and continue to attempt to uphold the maintenance condition (to state Active), or (5) *abort* the goal.

Achieve goals suspended from the Active state or from the Pending state. As with maintain goals, if not aborted prior to resumption, a goal may be resumed when its reconsideration condition becomes true, or when the agent decides to resume it.

For an achievement goal, the agent's options are to (1) leave the goal suspended (no change of state), (2) *reconsider* the goal (back to Pending state, first aborting any plans), (3) *reactivate* the goal by performing resume methods before transitioning to the Active state, or (4) *abort* the goal. Recall that suspend and resume methods, like abort methods, are assumed not to fail.

4.3.5 Aborting state

Goals in this state are goals that have been aborted, but for which the abort methods are in execution. Once the abort method has completed, the goal is dropped. If the goal has a trivial abort method, then it simply transitions immediately to the terminal state.

Note that once the goal is in the Aborting state, the only possible outcome is that the goal is dropped (once the abort methods have completed). Hence it is not possible to suspend—or perform any other operation on—a goal that is in the Aborting state.

5 A formal operational semantics of goal life-cycles

We now describe the formalisation of our approach to goals. We present here the formalisation for the goal transitions of Fig. 2. We formalize in an operational semantics all of the proceeding Sect. 4, except for the few specific points noted in that section.

5.1 Design issues

In order to use Fig. 2 as a specification of a goal deliberation process, we need to determine what information is required for each goal, and how this information is used to make decisions about when the transitions of Fig. 2 should be applied. Some transitions are essentially controlled by conditions, while others depend on an explicit agent decision (or a combination of conditions and a decision), as is made precise in our formal semantics.

Our approach is to follow that of van Riemsdijk et al. [49], which will provide formal definitions of the transitions for each goal. A similar approach has been used by Morandini et al. [25]. From our perspective, this provides a means of specifying the behaviour of the agent without having to consider all implementation details, which would be necessary if we were to use a specific agent programming language such as Jadex or JACK. Put another way, we are interested in the operational semantics of the agent execution, rather than presenting a programming solution in a specific language.

Besides the choice of formal language, two of the key differences in our work from that of van Riemsdijk et al. and Morandini et al.—which enable us to support the full variety of goal types and operations upon goals—are that we have four basic goal states (Pending, Monitoring, Active and Suspended) rather than two, and that not all transitions are possible (for example, goals of accomplishment can never be in the Monitoring state). This allows us to deal with suspended goals in a more detailed and realistic manner, as well as providing

a more natural semantics for maintenance goals. Further, unlike Morandini et al. [25], our semantics deals with plans as well as goals. This means that we can incorporate subgoals into plans, allowing the agent designer a richer and more natural way to specify the system's behaviour.

Our semantics allows for goal transitions to occur both due to mandated semantics (e.g., achievement of the success condition of a goal)—the *triggered by conditions* for which the agent does not make a deliberate decision—as well as due to agent deliberation: the *top-level commands* which are a decision by the agent's deliberation to impose an operation upon a goal. The latter case will be indicated by a change in the agent's beliefs, and specifically the addition of formulae such as *drop(g)*, indicating that the agent has decided to drop goal g. The precise means that an agent uses to come to such decisions is beyond the scope of this article; here we are concerned with the consequences of such a decision. It is an item of future work to determine when it is appropriate to drop (or abort or suspend, or perform any operation on) a particular goal, and hence in what follows we indicate the results of such reasoning by updating the agent's beliefs.

5.2 Overview of CAN rules

Following van Riemsdijk et al. [49] and Morandini et al. [25], we will give an operational semantics for the goal transitions of Fig. 2 in the formal system CAN [15,30,31,52]. This approach has the further benefit that we can study properties of the semantics at an appropriate level of abstraction. For example, we do not need to concern ourselves with the details of the way in which goals are implemented, or how the agent notifies itself that a plan has succeeded (possibly by posting an internal event).

CAN was developed with the intention of expressing the required behaviours, rather than a specific implementation. We choose CAN because it allows the specification of the operational semantics, which could then be implemented in an agent programming language such as JACK, Jadex, etc. In this sense, our use of CAN is intended as an artefact of the detailed design of an agent system, rather than an implementation. In other words, the role of CAN in our work is to specify how the interactions between goals operate, which in our contribution includes plan-level detail.

Using CAN as a basis means that we describe the agent behaviour as formal transitions between agent configurations of the form $\langle B, \mathcal{G} \rangle$, where B is the agent's beliefs, and $\mathcal{G}$ is the set of goals the agent is pursuing. The rules defining these transitions are written in the form:

$$\frac{Condition}{\langle B, \mathcal{G} \rangle \longrightarrow \langle B', \mathcal{G}' \rangle} \tag{1}$$

The behaviour of an agent is determined by the initial state of the agent and a fixed set of transition rules of the above form. These rules (described in detail in the following subsections) may be divided into three groups as follows. We use abbreviated forms of the transition labels of Fig. 2.

1. **Goal transition rules** corresponding to the states and transitions of Fig. 2:
   act(A), act(M), respond, re-respond, drop, abort, abort_start, abort_end, suspend, recon, react(A), react(M)
2. **Planning rules**: ($plan_1$, $plan_2$, $plan\_abort$, $fail$)
3. **Execution rules** (the remaining rules)

We believe that this is the first time that these three aspects have been combined into one semantics. The goal transition rules are essentially derived from Fig. 2, and are most directly

comparable to those of van Riemsdijk et al. [49] and Morandini et al. [25]. These rules are discussed in Sect. 5.3. The planning rules are also based on the framework of van Riemsdijk et al., and are used to determine the relationship between goals and plans. These are discussed in Sect. 5.4 The execution rules are those which deal with the execution of plans, which in this context can only take place if there are no changes in the status of the goals. These are discussed in Sect. 5.5.

It should be noted that an achievement goal is dropped on success, and also on failure. Accordingly, there is no explicit notion of success and failure for such goals in the following subsections; only whether the goal should be dropped or not. For maintenance goals, if the maintenance condition is violated and later restored, it is appropriate to return the goal to the Monitoring state rather than dropping it, although there may be circumstances under which it is appropriate to drop the maintenance goal altogether. Thus the main difference in terms of the formal semantics between achievement goals and maintenance goals is that the latter have a more elaborate sequence of possible transitions.

5.2.1 Assumptions

Our assumptions about the deliberation process represented by these transitions are:

- *All goals are given unique identifiers.* This ensures that goals can explicitly refer to other goals, allowing the agent designer to specify transitions such as one goal being suspended when another specific goal is activated.
- *Any change in any goal's state has preference over any executing plans.* This means that execution can only take place when the set of goal contexts (defined below) is stable, i.e., none of the transitions in Fig. 2 are currently able to take place. This is somewhat conservative, but it allows the agent designer the freedom to specify whatever interaction between goals is desired (such as making all achieve goals inactive whenever any maintain goal becomes active), knowing that any change in any goal's status will result in the status of all goals being reconsidered. This, in turn, may result in a corresponding change in what is being executed.
- *Plans are not necessarily known in advance, but may be generated online.* This means that we do not assume that the agent *necessarily* has a plan library (although this is a perfectly valid option). This also means that we have to explicitly allow for plan generation in our formal definition; we leverage previous techniques [49]. This is done in the rules below by the abstract procedure *mer*, which captures *means-end-reasoning* (see rules $plan_1$ and $plan_2$ below), as introduced by van Riemsdijk et al. This approach may be regarded as not being prescriptive about the way in which plans are associated with goals.
  In many BDI systems, such means-ends-reasoning is accomplised by means of *event-condition-action* rules, i.e., rules of the form $e : c \rightarrow A$, in which when event e is perceived, if condition c is true, then action A is performed. This would mean that when a goal is activated, an event is raised which will lead to the goal being activated. Whilst such rules can be accommodated in CAN, we believe the *mer* approach provides both flexibility and an appropriate level of abstraction, leaving us free to concentrate on transitions between goal states and the consequences of them. For instance, the procedure *mer* may select an applicable plan from a pre-written plan library (as in the earliest agent systems from the late 1980's). Alternatively, *mer* may use real-time planning algorithms to generate plans on-the-fly, or some combination of pre-written plans and plans generated as required. The important point from our perspective is that our approach does not specify or depend on a particular means to determine a plan to be executed for a given goal.

– *No restriction is made on the number of goals that may be active at once.* Different agent applications may have different requirements or preferences for the number of goals that are active at a particular time. For example, *Curiosity* may allow a number of experiments to proceed concurrently whilst in a particular location, but when returning to base, it seems sensible that only the *Return* goal is active. From the perspective of our operational semantics, this means that we need to be able to allow an arbitrary number of goals to be active at any time, but also to potentially allow means to enforce properties such as no other goals being active when a particular goal is active. Hence we provide the agent designer with mechanisms to enforce restrictions if desired, but not to build them into the CAN rules. Accordingly we will have a standard pattern for goal transition rules, which can be tailored by the designer to suit the particular application.
– *Achievement goals may be used as subgoals in plans.* This means that a plan may contain an achievement goal as a step, at which point the goal is executed, with the only difference being that success and failure are treated in the same way as success and failure for actions. We do not allow maintenance goals as subgoals of a plan; to establish the intended semantics and its implications is left for future work. Note however that the agent retains flexibility to manipulate maintenance goals, in that a plan can modify the beliefs relevant to the context of a maintenance goal.

5.2.2 Goal representation and state

In our semantics, each goal $g \in \mathcal{G}$ is described by a corresponding *goal context* tuple $\langle I, G, Rules, State, Plan\rangle$, in which

– I is a unique identifier for the goal
– G is the type, either achieve or maintain
– *Rules* is a set of *condition-action* pairs of the form $\langle c, A\rangle$, where c is a condition drawn from the language ℓ (used also for representing beliefs), and A is one of { activate, reactivate, reconsider, respond, re-respond, suspend, drop, abort }
– *State* is one of {P, M, A, S, B} representing the states Pending, Monitoring, Active, Suspended, and Aborting respectively.
– *Plan* is either: (1) the current plan body (if any) being executed for this goal, (2) the null value, ϵ, indicating the absence of any plan body for this goal (as distinct from the trivially successful or failed plan bodies, nil and $fail$), or (3) the special plan constant $abort$, indicating that an applicable abort method should be found and executed (as detailed in Sect. 5.3.3).

Note that our notation for g from this point on differs from the informal notational convenience used in Sect. 4. The existence of unique identifiers ensures that we can distinguish goals of the same type and ensures that goals can refer to each other. I, G and *Rules* are fixed throughout execution. *State* and *Plan* are dynamic and may change during execution. Hence we need only record the tuple $\langle I, State, Plan\rangle$ in order to specify the status of a goal at any point, as G and *Rules* can be determined from I. However, when describing rules, it is often simpler to use the full form $\langle I, G, Rules, State, Plan\rangle$.

Changes of state. There are four phases that take place in the execution cycle:

1. Updating beliefs and determining transitions for each goal
2. Performing state transitions for each goal

Table 1 Applicable operations by goal type and state

Operation	Goal type	States
Activate	Achieve, Maintain	Pending
Respond	Maintain	Monitoring
Suspend	Achieve, Maintain	Pending, Monitoring, Active
Reconsider	Achieve, Maintain	Suspended
Reactivate	Achieve	Suspended
Reactivate	Maintain	Suspended, Active
Re-respond	Maintain	Suspended
Abort	Achieve, Maintain	Pending, Monitoring, Active, Suspended
Drop	Achieve, Maintain	Pending, Monitoring, Active, Suspended

3. Determining plans for active goals, if necessary
4. Executing plans

It should be noted that each of the goal operations are only applicable in certain specific states. For example, there is no point in performing the activate action on a goal which is in the Active state. For this reason we specify via Table 1 the states and goal types for which each action is applicable. (Note that there are no applicable actions in the Aborting state.) We need to define formal rules only for the combinations listed in the table. We say a goal operation Op is *applicable* in state S on goal type G if there is an entry for Op, S, and G in Table 1; otherwise it is *inapplicable*.

An issue to be addressed is to determine when a change of goal state should occur. As discussed at the beginning of Sect. 4, we do not model the entire deliberation of the agent in this article, but only the changes of goal state that occur as a consequence of the agent's deliberation. We achieve this by the use of the distinguished set of facts in (2) below, in which I is a goal identifier:

$$drop(I), abort(I), suspend(I), reconsider(I), activate(I), reactivate(I) \quad (2)$$

Note that these correspond to the goal operations in Table 1, with the exception of respond and re-respond. The reason that these two operations are exempted is that the decision to activate a maintenance goal is one that is determined purely by the maintenance condition, and not by the decision making of the agent.

The decision to change the state of a particular goal is signified by asserting the corresponding fact into the beliefs of the agent. This means that we focus on the changes of goal state implied by such facts, rather than the process by which the decision to change state has occurred. For example, if the agent decides to suspend a particular goal g, then any subgoals of g should also be suspended. In our framework, this would be signified by the assertion of the form $suspend(g)$ into the agent's beliefs, and the rules for the subgoals of g would be used to determine that the subgoals should also be suspended. Typically, these facts are handed by the rules attached to each goal, as an input to the process to determine what action should be taken for each goal.

Once an operation has been chosen, there is no difference in the agent's behaviour between decisions based on internal conditions (such as the success condition of a goal becoming true) and those based on a top-level decision by the agent (such as dropping a goal because it conflicts with another one). This is very much by design, in that once a decision to drop

(or abort, or suspend, or resume) a goal has been made, the consequences of doing so are the same regardless of the reason for the change.

5.3 Goal transition CAN rules

We first describe the goal transition rules. Formalization of the semantics of state transitions hinges on the appropriate definition of *Rules* for each goal. These definitions follow the same general principles, and can be tailored for individual goals.

General approach. Given an action A in Fig. 2, which under condition c takes goal $g = \langle I, G, Rules, S_1, P_1\rangle$ from state S_1 to S_2, we ensure that there is a rule in R of the form $\langle c, A\rangle$ and that there is a transition to operate on that rule of the general form:

$$\frac{\langle c, A\rangle \in Rules \quad B \models c}{\langle B, \mathcal{G} \cup \{\langle I, G, Rules, S_1, P_1\rangle\}\rangle \rightarrow \langle B, \mathcal{G} \cup \{\langle I, G, Rules, S_2, P_2\rangle\}\rangle} \tag{3}$$

where $g \notin \mathcal{G}$.

We will abbreviate (3) as

$$\frac{\langle c, A\rangle \in Rules \quad B \models c}{\langle I, G, Rules, S_1, P_1\rangle \longrightarrow \langle I, G, Rules, S_2, P_2\rangle} \tag{4}$$

where $A \notin \{\mathsf{drop}, \mathsf{abort}\}$.

In some cases, the agent wants a condition c to be evaluated 'autonomously', i.e., without any further deliberation. In other cases, the agent wants an explicit condition. Thus, we require that all activations conditions for goals contain a formula of the form $activate(I)$, so that it is possible to specify conditions of the form $c = Cond \wedge activate(I)$. This means that for the goal to change state, not only must *Cond* hold, we must also have that the agent has explicitly decided to activate the goal by adding $activate(I)$ to its beliefs. This mechanism further allows us to provide for the possibility that the agent may decide to drop any goal at any time: it can do so by adding $drop(I)$ to its beliefs. The same holds for any of the distinguished facts in (2) above.

In particular, as noted in Sect. 4, it is common to include an activation condition of the form $\langle Cond \wedge \text{activate}(I), \mathsf{activate}\rangle$. Hence, we will denote by $standard(I, Succ, Cond)$, where *Succ* is a set of conditions, the following set of rules:

$$\begin{aligned}
&\{\langle s, \mathsf{drop}\rangle | s \in Succ\} \cup \\
&\{\langle \text{drop}(I), \mathsf{drop}\rangle, \langle \text{abort}(I), \mathsf{abort}\rangle, \langle \text{suspend}(I), \mathsf{suspend}\rangle, \\
&\quad \langle Cond \wedge \text{activate}(I), \mathsf{activate}\rangle\} \cup \\
&\{\langle \text{reactivate}(I), \mathsf{reactivate}\rangle, \langle \text{reconsider}(I), \mathsf{reconsider}\rangle\}
\end{aligned} \tag{5}$$

We now create parametrized rules within this framework for achieve and maintain goals.

5.3.1 Initial state of goals

Each goal g known to the system is initially in the Pending state and thus represented by a goal context $g = \langle I, G, Rules, \mathsf{P}, \epsilon\rangle$ where ϵ denotes the empty plan. Let κ be the pre-condition, S the success condition, and F the failure condition of g.

– For achieve(κ, S, F), $Rules$ is: $standard(I, \{S, F\}, \kappa)$
– For maintain(κ, C, R, P, S, F), $Rules$ is:
$standard(I, \{S, F\}, \kappa) \cup \{\langle\{\neg C, C \wedge \pi(\neg C)\}, \mathsf{respond}\rangle, \langle rerespond(I), \mathsf{re-respond}\rangle\}$

Note the form of the respond rule, in which we have an explicit condition for each of the two cases, i.e., when the maintenance condition is believed to be false ($\neg C$), in which case a reactive goal is activated, or when the maintenance condition is true but is predicted to become false in the future ($C \wedge \pi(\neg C)$). It would be possible, as with any of the rules, to re-write several conditions into one disjunctive condition, which in the case of respond would be $\neg C \vee (C \wedge \pi(\neg C))$. While this would mean that there is exactly one condition for every rule, we believe it is more intuitive and natural to write an implicit disjunction of the conditions, as above.

We will use the notation $G =$ achieve or $G =$ maintain to indicate whether G is an achieve goal or maintain goal respectively.

5.3.2 Activate transition rules

The transition rules for activation are straightforward. achieve goals go to the Active state:

$$\frac{G = \mathsf{achieve} \quad \langle c, \mathsf{activate}\rangle \in Rules \quad B \models c}{\langle I, G, Rules, \mathsf{Pending}, \epsilon\rangle \longrightarrow \langle I, G, Rules, \mathsf{Active}, \epsilon\rangle} \; act(A)$$

By contrast, instead of going directly to the Active state on activation, maintain goals go to the Monitoring state, in which the maintenance condition is being monitored, but no action is being taken yet.

$$\frac{G = \mathsf{maintain} \quad \langle c, \mathsf{activate}\rangle \in Rules \quad B \models c}{\langle I, G, Rules, \mathsf{Pending}, \epsilon\rangle \longrightarrow \langle I, G, Rules, \mathsf{Monitoring}, \epsilon\rangle} \; act(M)$$

Note that when a maintain goal is in the Monitoring state, the maintenance condition is being actively monitored, meaning that if the maintenance condition (i.e., C in maintain(κ, C, R, P, S, F)) is violated (or predicted to be violated), then the maintain goal transitions to the Active state (see the rule for the *respond* transition in Sect. 5.3.5 below). We model this by ensuring that the respond action is triggered when the maintenance goal is in the Monitoring state. This ensures that any maintenance condition which can be expressed as a formula in the logic of the agent's beliefs can be used.

5.3.3 Drop and abort transition rules

The drop action transition simply drops the goal:

$$\frac{\langle c, \mathsf{drop}\rangle \in Rules \quad B \models c}{\langle B, \mathcal{G} \cup \{\langle I, G, Rules, State, \pi\rangle\}\rangle \longrightarrow \langle B, \mathcal{G}\rangle} \; drop$$

Note that this can take place in any of the four basic states.

When an abort transition occurs in the Pending or Monitoring state, there is no need for abort methods, as nothing has been executed. When the abort takes place in the Active or Suspended states, abort methods are needed, as the goal may have plans or subgoals in execution. This means that there are two types of abort transition: one that simply drops the goal, and one that calls the abort methods.

The first abort transition is just like the drop transition above:

$$\frac{\langle c, \mathsf{abort}\rangle \in Rules \quad B \models c \quad State \in \{\mathsf{P}, \mathsf{M}\}}{\langle B, \mathcal{G} \cup \{\langle I, G, Rules, State, \pi\rangle\}\rangle \longrightarrow \langle B, \mathcal{G}\rangle} \; abort$$

The second abort transition involves two stages: first, putting the goal into the Aborting state, and then executing any appropriate abort method.

$$\frac{\langle c, \mathsf{abort}\rangle \in Rules \quad B \models c \quad State \in \{\mathsf{A}, \mathsf{S}\}}{\langle B, \mathcal{G} \cup \{\langle I, G, Rules, State, \pi\rangle\}\rangle \longrightarrow \langle B, \mathcal{G} \cup \{\langle I, G, Rules, \mathsf{Aborting}, abort\rangle\}\rangle} \; abort\_start$$

Once the goal is thus in the Aborting state with the special plan constant *abort* as the plan, an appropriate abort method is found by the same means-end-reasoning process that is used to find appropriate plans for active goals (see the *plan_abort* rule in Sect. 5.4). This means that the agent designer is free to choose whether to use real-time planning to find the abort method, or to allocate pre-defined abort methods (as is done in [36]).

Once the abort method has been executed, the goal is dropped. Since abort methods are assumed to never fail [37], executing an abort method will always terminate with *nil* (see Sect. 5.5 below). The *abort_end* rule is therefore unconditional.

$$\frac{}{\langle B, \mathcal{G} \cup \{\langle I, G, Rules, \mathsf{Aborting}, nil\rangle\}\rangle \longrightarrow \langle B, \mathcal{G}\rangle} \; abort\_end$$

5.3.4 Suspend transition rule

For the suspend action, the main issue to resolve is how to determine which state the goal should be returned to on resumption. We assume that the agent will indicate this by adding either $reactivate(I)$, $reconsider(I)$, or $rerespond(I)$ to its beliefs, as appropriate. Note that these conditions are evaluated as part of the rules for the actions reactivate, reconsider, and rerespond.

$$\frac{\langle c, \mathsf{suspend}\rangle \in Rules \quad B \models c \quad State \in \{\mathsf{P}, \mathsf{A}, \mathsf{M}\}}{\langle I, G, Rules, State, \pi\rangle \longrightarrow \langle I, G, Rules, \mathsf{Suspended}, \pi\rangle} \; suspend$$

5.3.5 Respond, subgoal and re-respond transition rules

The respond and re-respond actions pertain to maintain goals. When a maintenance goal becomes active, it triggers the adoption of a new achievement goal (either the recovery goal or preventative goal) with the intention that when this new goal is achieved, the violation of the maintenance condition (either actual or predicted) will be overcome. Hence for a maintain goal $G_M = \mathsf{maintain}(\kappa, C, R, P, S, F)$ the respond action (which is only available to maintain goals) results not only in the maintenance goal G_M becoming active, but also in the adoption of a new achievement goal, which we will denote as G_S. If the achievement goal G_S succeeds, then we reactivate G_M (i.e., G_M returns to the Monitoring state). We can ensure this by making the success condition S_{SG} of G_S the only condition for the reactivate rule. If G_S fails or is dropped or aborted, we drop G_M. In addition, G_M is dropped if either its success condition S or failure condition F becomes true. These properties are all reflected in the rules for the maintenance goal, and in particular the conditions for the actions reactivate and drop.

As noted previously, there is not a single subgoal G_S which is applicable in all circumstances; in particular, a different subgoal may be pursued depending on whether the maintenance condition was violated, or predicted to be violated in the future. In the former case, we adopt a recovery goal R; in the latter, we adopt a preventative goal P. We express this variation in the respond rule by executing the plan (6) below for G_M, where C is the maintenance condition, and π is a predictive function, i.e., one that can predict that the maintenance condition will be violated in the near future.

$$(\!|\neg C : R, (C \wedge \pi(\neg C)) : P|\!) \tag{6}$$

This plan is essentially a notational convenience, in that the conditions $\neg C$ and $C \wedge \pi(\neg C)$ will be evaluated at the same time as the condition of the respond action (see the rule below). This means that we only need to specify one rule, rather than one for adopting R and a near-identical one for adopting P. Note also that once the appropriate goal is chosen, it will immediately transition from the Pending state to the Active state, as its activation condition is $\neg C$ for R and $C \wedge \pi(\neg C)$ for P.

$$\frac{G = \mathsf{Maintain} \quad \langle c, \mathsf{respond}\rangle \in \mathit{Rules} \quad B \models c}{\langle B, \mathcal{G} \cup \{\langle I_1, G_M, \mathit{Rules}, \mathsf{M}, \epsilon\rangle\}\rangle \longrightarrow \langle B, \mathcal{G} \cup \{\langle I_1, G_M, \mathit{Rules}, \mathsf{A}, (\!|\neg C : R, (C \wedge \pi(\neg C)) : P|\!)\rangle\}\rangle} \; \mathit{respond}$$

where R is the recovery goal of G_M and P is the preventative goal of G_M.

The case for a subgoal is closely related (i.e., when the current plan step is a subgoal), and so we discuss it here, rather than with the other plan execution rules (see Sect. 5.5). The subgoal is executed *sequentially*, i.e., the parent goal waits until the subgoal has completed before moving on. If the subgoal succeeds, the subgoal is dropped, and execution proceeds (just as in the case of any other successful step). If the subgoal fails, or is dropped (other than after it has succeeded) or aborted, then this is treated as a plan failure, i.e., that the step failed and hence (by default) an alternative, if available, should be pursued. Hence the plan for the parent goal is to wait for one of the success or failure conditions to become true, or for the subgoal to be dropped or aborted, and then query whether the subgoal succeeded. The parent goal's step then succeeds only if the success condition of the subgoal is true.

$$\frac{\mathit{stable}}{\langle B, \mathcal{G} \cup \{\langle I_P, G_1, \mathit{Rules}, \mathsf{A}, G_2\rangle\}\rangle \longrightarrow \langle B, \mathcal{G} \cup \{\langle I_P, G_1, \mathit{Rules}, \mathsf{A}, \mathit{SGP}\rangle\} \cup \{\langle I_C, G_2, \mathit{Rules}_1, \mathsf{P}, \epsilon\rangle\}\rangle} \; \mathit{goal}$$

where

- SGP is $S_c \vee F_c \vee \mathit{drop}(I_C) \vee \mathit{abort}(I_C) :?S_c$
- Rules_1 is $\mathit{standard}(I_C, \{S_c, F_c\}, \mathit{true}) \cup \{\langle \mathit{drop}(I_P), \mathsf{drop}\rangle, \langle \mathit{abort}(I_P), \mathsf{abort}\rangle, \langle \mathit{suspend}(I_P)\{\mathsf{suspend}\rangle\}$
  $\cup\{\langle \mathit{reactivate}(I_P), \mathsf{reactivate}, \langle \mathit{reconsider}(I_P), \mathsf{reconsider}\rangle\}$
- stable denotes the condition that there are no applicable actions to change the state of a goal (see Sect. 5.4 for more details).

Note also the presence of the extra rules (i.e.in addition to the standard rules) for drop, abort, suspend, reactivate and reconsider. The presence of these rules means that whenever the parent goal is dropped, aborted, suspended, reactivated or reconsideed, then the same action is applied to the subgoal. This does not imply the reverse, i.e.that if a subgoal undergoes any of these transitions, this does not mean that the parent goal necessarily undergoes the same transition. Hence the behaviour of the subgoal will mirror that of its parent goal, but may also undergo further transitions.

It follows that the subgoal transition will only be applicable when there are no changes of goal state, and it will add a new goal to the goal context. This is precisely the behaviour required by the respond action.

Note also the addition of the rules below to the subgoal beyond those from the *standard* set:

$$\langle drop(I_P), \mathsf{drop}\rangle, \langle abort(I_P), \mathsf{abort}\rangle, \langle suspend(I_P), \mathsf{suspend}\rangle, \\ \langle reactivate(I_P), \mathsf{reactivate}\rangle \langle reconsider(I_P), \mathsf{reconsider}\rangle \quad (7)$$

In particular, note the id I_P which ensures that if the parent goal is dropped, aborted, suspended, reactivated or reconsidered, then the subgoal will also undergo the corresponding transition. The ability to specify rules of this nature is one of the features of our approach, as this mechanism ensures that the subgoal behaves appropriately, simply by ensuring that the appropriate rules are in place.

The rule for re-respond is similar to the rule for respond. The main issue is that there are two goals that need to be made active again, namely the maintain goal and the previously active subgoal (since re-respond can only apply if the maintain goal was in the Active state before being suspended).

$$\frac{G_M = \mathsf{Maintain}\ \langle c, \mathsf{re-respond}\rangle \in Rules_1 \quad B \models c}{\langle B, \mathcal{G} \cup \{\langle I_1, G_M, Rules_1, \mathsf{S}, SGP\rangle, \langle I_2, G_S, Rules_2, \mathsf{S}, \pi\rangle\}\rangle \longrightarrow \langle B, \mathcal{G} \cup \{\langle I_1, G_M, Rules_1, \mathsf{A}, SGP\rangle, \langle I_2, G_S, Rules_2, \mathsf{A}, \epsilon\rangle\}\rangle}\ \textit{re-respond}$$

where we denote by SGP the same expression as in the goal rule, i.e., SGP is $S_c \vee F_c \vee drop(I_2) \vee abort(I_2) :?S_c$. Recall that S_c is the success condition of the subgoal, and this means that executing SGP means waiting for the goal to terminate, and then testing whether or not it succeeded.

5.3.6 Reconsider and reactivate transition rules

Finally for the goal transition rules, we have the pair of rules related to suspended goals.

The rule for reconsider is:

$$\frac{\langle c, \mathsf{reconsider}\rangle \in Rules \quad B \models c}{\langle I, G, Rules, \mathsf{Suspended}, \pi\rangle \longrightarrow \langle I, G, Rules, \mathsf{Pending}, \epsilon\rangle}\ recon$$

The rule for reactivate differs between achieve goals and maintain goals. For the former, we have the rule

$$\frac{G = \mathsf{achieve} \quad \langle c, \mathsf{reactivate}\rangle \in Rules \quad B \models c}{\langle I, G, Rules, \mathsf{Suspended}, \pi\rangle \longrightarrow \langle I, G, Rules, \mathsf{Active}, \epsilon\rangle}\ react(A)$$

whereas for the latter we have the rule

$$\frac{State \in \{\mathsf{A}, \mathsf{S}\} \quad G = \mathsf{maintain} \quad \langle c, \mathsf{reactivate}\rangle \in Rules \quad B \models c}{\langle I, G, Rules, State, \pi\rangle \longrightarrow \langle I, G, Rules, \mathsf{Monitoring}, \epsilon\rangle}\ react(M)$$

5.4 Planning transition CAN rules

Having described the goal transition rules, we next describe the planning transition rules. We denote by $stable(B, \mathcal{G})$ that for all goals g in $\mathcal{G}$ we have that if $\langle c, \mathsf{action}\rangle \in Rules$ and $B \models c$, then action is not applicable (where $Rules$ is the rules for g). In other words, for all goals $g \in \mathcal{G}$, either there is no $\langle c, \mathsf{action}\rangle \in Rules$ such that $B \models c$, or that for all $\langle c, \mathsf{action}\rangle \in Rules$ such that $B \models c$, action is not applicable in the current goal state, as defined in Table 1. For example, the activate action is only applicable in the Pending state. This definition is needed to allow for the possibility that $B \models c$ where $\langle c, \mathsf{activate}\rangle \in Rules$

when g is already in the Active state, and so the action will have no effect. In other words, an action only takes place if its preconditions are believed, and the action is applicable (as defined by Table 1) in the current state of the goal. We will often abuse notation and write just *stable* when the beliefs and goals are clear from the context.

Once all changes in goal state have occurred—and hence the goal states are 'stable'—the next step is to determine whether there are goals for which plans need to be found. We do this by inspecting the final element of the goal context tuple; if this is the empty plan, this indicates that a plan needs to be found. Otherwise, the existing plan is used. When a plan needs to be found, following van Riemsdijk et al. [49] we make use of a *means-end-reasoning* procedure *mer*, i.e., a means of generating a plan for a given goal (as introduced in Sect. 2.3). This means may depend on the goal itself, the agent's beliefs, and the other goals of the agent. The advantage of this mechanism is that it abstracts the rules away from any particular method of associating plans with goals. In particular, this approach will be applicable for pre-existing libraries of plans (a traditional agent-oriented programming technique), or for agents with online planning ability, or a combination of both.

It may not always be possible to find such a plan; in this case, we assume that the *mer* procedure will return the empty plan ϵ. Hence the first two rules below deal with when a non-empty plan is returned ($plan_1$) and when an empty plan is returned ($plan_2$).

It is also possible that the plan found may fail; in this case, having executed the plan, we update the current plan for the goal to ϵ, as a 'signal' that a new plan needs to be found (the *fail* rule below). This effectively signals the *mer* procedure to attempt to find a further plan, i.e., to reconsider the rules $plan_1$ and $plan_2$ at the appropriate time.

Hence the planning transition rules as follows:

$$\frac{stable \quad \Pi = mer(G, B, \mathcal{G} \cup \{\langle I, G, Rules, \mathsf{Active}, \epsilon\rangle\}) \quad \Pi \neq \epsilon}{\langle B, \mathcal{G} \cup \{\langle I, G, Rules, \mathsf{Active}, \epsilon\rangle\}\rangle \longrightarrow \langle B, \mathcal{G} \cup \{\langle I, G, Rules, \mathsf{Active}, \Pi\rangle\}\rangle} \; plan_1$$

$$\frac{stable \quad \Pi = mer(G, B, \mathcal{G} \cup \{\langle I, G, Rules, \mathsf{Active}, \epsilon\rangle\}) \quad \Pi = \epsilon}{\langle B, \mathcal{G} \cup \{\langle I, G, Rules, \mathsf{Active}, \epsilon\rangle\}\rangle \longrightarrow \langle B, \mathcal{G}\rangle} \; plan_2$$

$$\frac{stable \quad \Pi = mer(G, B, \mathcal{G} \cup \{\langle I, G, Rules, \mathsf{Aborting}, abort\rangle\})}{\langle B, \mathcal{G} \cup \{\langle I, G, Rules, \mathsf{Aborting}, abort\rangle\}\rangle \longrightarrow \langle B, \mathcal{G} \cup \{\langle I, G, Rules, \mathsf{Aborting}, \Pi\rangle\}\rangle} \; plan\_abort$$

$$\frac{stable \quad \pi \neq \epsilon \quad \langle B, \mathcal{G} \cup \{\langle I, G, Rules, \mathsf{Active}, \pi\rangle\}\rangle \longrightarrow \langle B', \mathcal{G} \cup \{\langle I, G, Rules, \mathsf{Active}, fail\rangle\}\rangle}{\langle B, \mathcal{G} \cup \{\langle I, G, Rules, \mathsf{Active}, \pi\rangle\}\rangle \longrightarrow \langle B', \mathcal{G} \cup \{\langle I, G, Rules, \mathsf{Active}, \epsilon\rangle\}\rangle} \; fail$$

Note that we assume that *mer* will never return ϵ when called as part of the *plan_abort* rule; if there is no abort method, in this case *mer* will return nil.

Finally, note that the *fail* rule is merely a means of ensuring that the result of plan execution is the empty plan ϵ. As the success and failure conditions for a goal are given separately, there is no need for the execution mechanism to 'signal' the failure (or success) of a plan. In other words, ϵ only signifies that another plan needs to be found if execution is to continue.

5.5 Execution transition CAN rules

Finally, we describe the execution transition rules. We refer to Sect. 2.3 for an introduction to CAN's execution transition rules. Recall that the plan $P_1; P_2$ executes P_1, and then executes P_2 if P_1 succeeds; the plan $P_1 \triangleright P_2$ executes P_1 and then executes P_2 if P_1 fails; the plan $P_1 \| P_2$ executes the subplans P_1 and P_2 concurrently, and fails if either of P_1 and P_2 fails; the plan nil always succeeds; and the plan $fail$ always fails.

The execution rules are based on the standard CAN rules [15], with two extensions. Specifically, the wait construct (i.e., $\phi : P$ where P is not executed unless ϕ is true) and the rule *goal* (see above) which deals with the case when goals (of any type) can occur in plans (and hence as subgoals of another goal). These two minor extensions are introduced here, but they do not change the essential nature of CAN.

We obtain the set of twenty-four rules that follow in this subsection. Note the presence of *stable* in each premise; this ensures that plans only execute when there are no changes of goal state. Note also that plans can be executed when a goal is either in the Active state (regular plans) or the Aborting state (abort methods).

In the rules below, we abbreviate both

$$\langle B, \mathcal{G} \cup \{\langle I, G, R, \textsf{Active}, P_1\rangle\}\rangle \longrightarrow \langle B', \mathcal{G} \cup \{\langle I, G, R, \textsf{Active}, P_2\rangle\}\rangle$$

and

$$\langle B, \mathcal{G} \cup \{\langle I, G, R, \textsf{Aborting}, P_1\rangle\}\rangle \longrightarrow \langle B', \mathcal{G} \cup \{\langle I, G, R, \textsf{Aborting}, P_2\rangle\}\rangle$$

to $\langle B, P_1\rangle \longrightarrow \langle B', P_2\rangle$. In other words, plan execution can take place in both the Active and Aborting states, but there is no change in the goal configuration in either state apart from the beliefs and the plan itself.

We are now ready to present the execution rules, as follows.

$$\frac{\mathit{stable} \quad \langle B, P_1\rangle \longrightarrow \langle B', P'\rangle}{\langle B, P_1 \| P_2\rangle \longrightarrow \langle B', P' \| P_2\rangle} \|_1 \qquad \frac{\mathit{stable} \quad \langle B, P_2\rangle \longrightarrow \langle B', P'\rangle}{\langle B, P_1 \| P_2\rangle \longrightarrow \langle B', P_1 \| P'\rangle} \|_2$$

$$\frac{\mathit{stable} \quad \langle B, P_1\rangle \longrightarrow \langle B', P'\rangle}{\langle B, P_1 \triangleright P_2\rangle \longrightarrow \langle B', P' \triangleright P_2\rangle} \triangleright_1$$

$$\frac{\mathit{stable} \quad \langle B, P_1\rangle \longrightarrow \langle B'', \mathit{fail}\rangle \quad \langle B, P_2\rangle \longrightarrow \langle B', P'\rangle}{\langle B, P_1 \triangleright P_2\rangle \longrightarrow \langle B', P'\rangle} \triangleright_2 \qquad \frac{\mathit{stable} \quad \langle B, P_1\rangle \longrightarrow \langle B', P'\rangle}{\langle B, P_1; P_2\rangle \longrightarrow \langle B', P'; P_2\rangle} ;$$

$$\frac{\mathit{stable}}{\langle B, \mathit{nil} \| P\rangle \longrightarrow \langle B, P\rangle} \mathit{nil}_1 \qquad \frac{\mathit{stable}}{\langle B, P \| \mathit{nil}\rangle \longrightarrow \langle B, P\rangle} \mathit{nil}_2 \qquad \frac{\mathit{stable}}{\langle B, \mathit{nil}; P\rangle \longrightarrow \langle B, P\rangle} \mathit{nil}_3$$

$$\frac{\mathit{stable}}{\langle B, P; \mathit{nil}\rangle \longrightarrow \langle B, P\rangle} \mathit{nil}_4 \qquad \frac{\mathit{stable}}{\langle B, \mathit{nil} \triangleright P\rangle \longrightarrow \langle B, \mathit{nil}\rangle} \mathit{nil}_5 \qquad \frac{\mathit{stable}}{\langle B, \mathit{fail} \| P\rangle \longrightarrow \langle B, \mathit{fail}\rangle} \mathit{fail}_1$$

$$\frac{\mathit{stable}}{\langle B, P \| \mathit{fail}\rangle \longrightarrow \langle B, \mathit{fail}\rangle} \mathit{fail}_2 \qquad \frac{\mathit{stable}}{\langle B, \mathit{fail}; P\rangle \longrightarrow \langle B, \mathit{fail}\rangle} \mathit{fail}_3$$

$$\frac{\mathit{stable}}{\langle B, \mathit{fail} \triangleright P\rangle \longrightarrow \langle B, P\rangle} \mathit{fail}_4$$

We follow CAN's approach to plans, in that if the precondition of an action is true, then we assume that the action succeeds. In other words, the only way for an action to fail is for its precondition to be false. Hence in the rules act_1 and act_2 below, we first test the precondition ($pre(a)$); if this follows from the agent's beliefs, then the action succeeds and the beliefs are updated appropriately; otherwise the action fails. It is possible to allow for more sophisticated processing, such as *sensory actions*, for which it is possible to tell immediately after execution whether it has succeeded or not. Designing rules for such actions is outside the scope of this article; for now, we note that this is not an intrinsic limitation of CAN, but is purely a design decision.

In line with this approach, the remaining execution rules are as follows.

$$\frac{stable \quad B \models pre(a)}{\langle B, a\rangle \longrightarrow \langle B', nil\rangle}\ act_1 \qquad \frac{stable \quad B \not\models pre(a)}{\langle B, a\rangle \longrightarrow \langle B, fail\rangle}\ act_2$$

$$\frac{stable}{\langle B, nil\rangle \longrightarrow \langle B, \epsilon\rangle}\ nil \qquad \frac{stable}{\langle B, +b\rangle \longrightarrow \langle B \cup \{b\}, \epsilon\rangle}\ add \qquad \frac{stable}{\langle B, -b\rangle \longrightarrow \langle B \backslash \{b\}, \epsilon\rangle}\ del$$

$$\frac{stable \quad B \models \phi}{\langle B, ?\phi\rangle \longrightarrow \langle B, \epsilon\rangle}\ query_1 \qquad \frac{stable \quad B \not\models \phi}{\langle B, ?\phi\rangle \longrightarrow \langle B, fail\rangle}\ query_2$$

$$\frac{stable \quad \psi_i : P_i \in \Delta \quad B \models \psi_i}{\langle B, (\!|\Delta|\!)\rangle \longrightarrow \langle B, P_i \triangleright (\!|\Delta \setminus \{\psi_i : P_i\}|\!)\rangle}\ select$$

$$\frac{stable \quad B \models \phi \quad \langle B, P\rangle \longrightarrow \langle B', P'\rangle}{\langle B, \phi : P\rangle \longrightarrow \langle B', P'\rangle}\ wait_1 \qquad \frac{stable \quad B \not\models \phi}{\langle B, \phi : P\rangle \longrightarrow \langle B, \phi : P\rangle}\ wait_2$$

6 Illustrating the operational semantics: *Curiosity* scenario revisited

We now consider how to apply the transitions from Fig. 2 to the earlier *Curiosity* scenario. Our purpose being to illustrate the semantics, we construct the scenario with certain simplifying assumptions which we outline below.

6.1 Basic scenario

Curiosity's initial goals are the achievement goal Exp, which performs various experiments, the maintenance goal $Battery$, which maintains a certain battery level, and $SciOpp$, which investigates science opportunities as and when they arise. We also include a goal $Return$, which will be triggered shortly before dusk to ensure that *Curiosity* returns to its base before the end of the sol (Mars day). Hence *Curiosity* will have the goals shown below:

Name	Goal type and signature
Exp	achieve($dawn, S, dusk \wedge \neg S$)
$Battery$	maintain($true, battery\_level \geq 10\,\%, Charge, Charge, false, false$)
$Charge$	achieve($true, battery\_level \geq 100\,\%, battery\_failure$)
$SciOpp$	achieve($rock\_seen, rock\_analysed, analysis\_failure$)
$Return$	achieve($nearly\_dusk, at\_base, movement\_failure$)

With the exception of $Return$, these goals are shown in Fig. 1. In the figure, Exp is written out as $Experiment$.

For the purposes of the scenario, we will not need to make any more precise the latter three failure conditions. Initially, all four goals are in the Pending state. Exp can commence at dawn, and has a particular set of experiments to perform (denoted by S), which are considered failed if they cannot be completed by dusk. $Battery$ is a maintenance goal which is intended to keep the battery level above 10 % (which is denoted by C in the rules below for brevity). If the charge falls below this level (or is predicted to fall below this level), the subgoal $Charge$ will be triggered. $Charge$ is an achieve goal, which is activated when there is a maintenance goal violation, and succeeds when the battery is sufficiently charged. $SciOpp$ is a science opportunity goal, applicable when a certain kind of rock is seen; it succeeds when the rock

has been analysed. *Return* is an achieve goal, which is applicable when it is nearly dusk, and hence time to return to the base.

Once *Curiosity* returns to its base, this will potentially trigger further actions, such as performing regular maintenance tasks, and scheduling its next communication with Earth. This communication will also include a fresh set of goals for the next sol. Hence in our initial scenario, we will only consider the management of the goals during the Mars day; we demonstrate more intricate goal management in Sect. 6.2.

For the five goals given above, *Battery* (and hence *Charge*) has the highest priority (meaning that any other goals that are active when *Battery* is activated should be suspended), *Return* the next highest, *SciOpp* the next highest, and *Exp* the lowest priority of all.

This means that *Curiosity*'s initial goal context is as follows:

$$\langle i_1, Exp, R_1, \mathsf{P}, \epsilon\rangle, \langle i_2, SciOpp, R_2, \mathsf{P}, \epsilon\rangle, \langle i_3, Battery, R_3, \mathsf{P}, \epsilon\rangle, \langle i_4, Return, R_4, \mathsf{P}, \epsilon\rangle$$

where:
$R_1 = standard(i_1, \{S, dusk \wedge \neg S\}, \{dawn\}) \cup \{\langle\{active(i_4)\}, \mathsf{abort}\rangle, \langle\{active(i_2), active(i_3)\}, \mathsf{suspend}\rangle\}$,
$R_2 = standard(i_2, \{rock\_analysed\}, \{rock\_seen\}) \cup \{\langle\{active(i_3)\}, \mathsf{suspend}\rangle\}$,
$R_3 = standard(i_3, \{\}, \{true\}) \cup \{\langle\{\neg C, C \wedge \pi(\neg C)\}, \mathsf{respond}\rangle, \langle\{re - respond(i_3)\}, \mathsf{re-respond}\}$,
$R_4 = standard(i_4, \{at\_base\}, \{nearly\_dusk\}) \cup \{\langle\{active(i_2), active(i_3), \mathsf{suspend}\}\rangle\}$.
As *Charge* is triggered only when needed, we do not include it in the initial goal context.[5]

Observe that priority amongst the goals is reflected in the rules; for example, as *Battery* has a higher priority than *Exp*, if *Battery* becomes active when *Exp* is active, then *Exp* will be suspended. Naturally, if a different priority is desired by the agent designer, it is a simple matter to re-arrange the rules to reflect this priority. The abort method for *Exp* (which we do not write out in full) has been designed to include recording the information gathered so far.

As noted above, given that the goals and their associated rules do not change during execution, we can represent the initial goal context as:

$$\langle Exp, \mathsf{P}, \epsilon\rangle, \langle SciOpp, \mathsf{P}, \epsilon\rangle, \langle Battery, \mathsf{P}, \epsilon\rangle, \langle Return, \mathsf{P}, \epsilon\rangle$$

(where we have chosen to retain meaningful goal identifiers, rather than ones systematically generated).

For the sake of illustration, we suppose that the plan chosen to achieve *Exp* has four sequential steps, and hence can be represented as $p_1; p_2; p_3; p_4$. More complex plan structures can be accommodated in CAN, but do not add anything essential to this example. Similarly we suppose that the plan for *SciOpp* is $s_1; s_2$ (such as moving to the rock and then analysing it), the plan for *Charge* is the single action *Charge*, and that the plan for *Return* is the single action *return*.

An illustration of the expected sequence of *Curiosity*'s goal states is given in Table 2. The agent's initial beliefs include that it is dawn. Its programming includes a default command to consider and activate goal *Battery* at the start of each sol, and programming to ensure that the agent keeps track of a clock, which updates beliefs such as *dawn*, *dusk*, and *nearly_dusk* appropriately. Accordingly, the agent's initial beliefs include that the goals *Battery* and *Exp* should be activated.

[5] An alternative would be to have it in the Pending state initially, with its activation condition being the maintenance condition.

Table 2 *Curiosity*'s sequence of goal states: each tuple contains the name of the goal, the current state (P—Pending, A—Active, M—Monitoring, S—Suspended, B—Aborting), and the current plan to achieve the goal, respectively

Stage	Experiment	Science	Battery	Return
1	$\langle Exp, \mathsf{P}, \epsilon\rangle$	$\langle SciOpp, \mathsf{P}, \epsilon\rangle$	$\langle Battery, \mathsf{P}, \epsilon\rangle$	$\langle Return, \mathsf{P}, \epsilon\rangle$
2	$\langle Exp, \mathsf{P}, \epsilon\rangle$	$\langle SciOpp, \mathsf{P}, \epsilon\rangle$	$\langle Battery, \mathsf{M}, \epsilon\rangle$	$\langle Return, \mathsf{P}, \epsilon\rangle$
3	$\langle Exp, \mathsf{A}, \epsilon\rangle$	$\langle SciOpp, \mathsf{P}, \epsilon\rangle$	$\langle Battery, \mathsf{M}, \epsilon\rangle$	$\langle Return, \mathsf{P}, \epsilon\rangle$
4	$\langle Exp, \mathsf{A}, p_1; p_2; p_3; p_4\rangle$	$\langle SciOpp, \mathsf{P}, \epsilon\rangle$	$\langle Battery, \mathsf{M}, \epsilon\rangle$	$\langle Return, \mathsf{P}, \epsilon\rangle$
5	$\langle Exp, \mathsf{A}, p_2; p_3; p_4\rangle$	$\langle SciOpp, \mathsf{P}, \epsilon\rangle$	$\langle Battery, \mathsf{M}, \epsilon\rangle$	$\langle Return, \mathsf{P}, \epsilon\rangle$
6	$\langle Exp, \mathsf{A}, p_3; p_4\rangle$	$\langle SciOpp, \mathsf{P}, \epsilon\rangle$	$\langle Battery, \mathsf{M}, \epsilon\rangle$	$\langle Return, \mathsf{P}, \epsilon\rangle$
7	$\langle Exp, \mathsf{A}, p_3; p_4\rangle$	$\langle SciOpp, \mathsf{P}, \epsilon\rangle$	$\langle Battery, \mathsf{A}, MGP\rangle$	$\langle Return, \mathsf{P}, \epsilon\rangle$
8	$\langle Exp, \mathsf{S}, p_3; p_4\rangle$	$\langle SciOpp, \mathsf{P}, \epsilon\rangle$	$\langle Battery, \mathsf{A}, MGP\rangle$	$\langle Return, \mathsf{P}, \epsilon\rangle$
9	$\langle Exp, \mathsf{S}, p_3; p_4\rangle$	$\langle SciOpp, \mathsf{P}, \epsilon\rangle$	$\langle Battery, \mathsf{A}, Charge\rangle$	$\langle Return, \mathsf{P}, \epsilon\rangle$
10	$\langle Exp, \mathsf{S}, p_3; p_4\rangle$	$\langle SciOpp, \mathsf{P}, \epsilon\rangle$	$\langle Battery, \mathsf{A}, SGP\rangle$, $\langle Charge, \mathsf{P}, \epsilon\rangle$	$\langle Return, \mathsf{P}, \epsilon\rangle$
11	$\langle Exp, \mathsf{S}, p_3; p_4\rangle$	$\langle SciOpp, \mathsf{P}, \epsilon\rangle$	$\langle Battery, \mathsf{A}, SGP\rangle$, $\langle Charge, \mathsf{A}, \epsilon\rangle$	$\langle Return, \mathsf{P}, \epsilon\rangle$
12	$\langle Exp, \mathsf{S}, p_3; p_4\rangle$	$\langle SciOpp, \mathsf{P}, \epsilon\rangle$	$\langle Battery, \mathsf{A}, SGP\rangle$, $\langle Charge, \mathsf{A}, charge\rangle$	$\langle Return, \mathsf{P}, \epsilon\rangle$
13	$\langle Exp, \mathsf{S}, p_3; p_4\rangle$	$\langle SciOpp, \mathsf{P}, \epsilon\rangle$	$\langle Battery, \mathsf{A}, SGP\rangle$, $\langle Charge, \mathsf{A}, nil\rangle$	$\langle Return, \mathsf{P}, \epsilon\rangle$
14	$\langle Exp, \mathsf{S}, p_3; p_4\rangle$	$\langle SciOpp, \mathsf{P}, \epsilon\rangle$	$\langle Battery, \mathsf{A}, SGP\rangle$	$\langle Return, \mathsf{P}, \epsilon\rangle$
15	$\langle Exp, \mathsf{S}, p_3; p_4\rangle$	$\langle SciOpp, \mathsf{P}, \epsilon\rangle$	$\langle Battery, \mathsf{M}, \epsilon\rangle$	$\langle Return, \mathsf{P}, \epsilon\rangle$
16	$\langle Exp, \mathsf{A}, \epsilon\rangle$	$\langle SciOpp, \mathsf{P}, \epsilon\rangle$	$\langle Battery, \mathsf{M}, \epsilon\rangle$	$\langle Return, \mathsf{P}, \epsilon\rangle$
17	$\langle Exp, \mathsf{A}, p_3; p_4\rangle$	$\langle SciOpp, \mathsf{P}, \epsilon\rangle$	$\langle Battery, \mathsf{M}, \epsilon\rangle$	$\langle Return, \mathsf{P}, \epsilon\rangle$
18	$\langle Exp, \mathsf{A}, p_4\rangle$	$\langle SciOpp, \mathsf{P}, \epsilon\rangle$	$\langle Battery, \mathsf{M}, \epsilon\rangle$	$\langle Return, \mathsf{P}, \epsilon\rangle$
19	$\langle Exp, \mathsf{A}, p_4\rangle$	$\langle SciOpp, \mathsf{A}, \epsilon\rangle$	$\langle Battery, \mathsf{M}, \epsilon\rangle$	$\langle Return, \mathsf{P}, \epsilon\rangle$
20	$\langle Exp, \mathsf{S}, p_4\rangle$	$\langle SciOpp, \mathsf{A}, \epsilon\rangle$	$\langle Battery, \mathsf{M}, \epsilon\rangle$	$\langle Return, \mathsf{P}, \epsilon\rangle$
21	$\langle Exp, \mathsf{S}, p_4\rangle$	$\langle SciOpp, \mathsf{A}, s_1; s_2\rangle$	$\langle Battery, \mathsf{M}, \epsilon\rangle$	$\langle Return, \mathsf{P}, \epsilon\rangle$
22	$\langle Exp, \mathsf{S}, p_4\rangle$	$\langle SciOpp, \mathsf{A}, s_2\rangle$	$\langle Battery, \mathsf{M}, \epsilon\rangle$	$\langle Return, \mathsf{P}, \epsilon\rangle$
23	$\langle Exp, \mathsf{S}, p_4\rangle$	$\langle SciOpp, \mathsf{A}, nil\rangle$	$\langle Battery, \mathsf{M}, \epsilon\rangle$	$\langle Return, \mathsf{P}, \epsilon\rangle$
24	$\langle Exp, \mathsf{S}, p_4\rangle$	(dropped)	$\langle Battery, \mathsf{M}, \epsilon\rangle$	$\langle Return, \mathsf{A}, \epsilon\rangle$
25	$\langle Exp, \mathsf{B}, save\rangle$	–	$\langle Battery, \mathsf{M}, \epsilon\rangle$	$\langle Return, \mathsf{A}, \epsilon\rangle$
26	$\langle Exp, \mathsf{B}, nil\rangle$	–	$\langle Battery, \mathsf{M}, \epsilon\rangle$	$\langle Return, \mathsf{A}, \epsilon\rangle$
27	(dropped)	–	$\langle Battery, \mathsf{M}, \epsilon\rangle$	$\langle Return, \mathsf{A}, \epsilon\rangle$
28	–	–	$\langle Battery, \mathsf{M}, \epsilon\rangle$	$\langle Return, \mathsf{A}, return\rangle$
29	–	–	$\langle Battery, \mathsf{M}, \epsilon\rangle$	$\langle Return, \mathsf{A}, nil\rangle$
30	–	–	$\langle Battery, \mathsf{M}, \epsilon\rangle$	(dropped)

The absence of a plan is denoted by ϵ—empty. MGP is $\langle\!|\neg C : Charge, (C \wedge \pi(\neg C)) : Charge|\!\rangle$, SGP is $battery\_charged \vee drop(Charge) \vee abort(Charge) : ?battery\_charged$

Initially, *Curiosity*'s goals are all in the Pending state. $Battery$ is then activated, and moves into the Monitoring state, and then Exp is activated and moves into the Active state. As Exp is the only active goal, a plan is generated for it (p_1; p_2; p_3; p_4) and starts executing. During the execution of action p_3, a violation of the battery level is predicted, and so $Battery$ is activated. This triggers the activation of the goal $Charge$, which commences in the Pending state but immediately moves to the Active state, and for which the plan $Charge$

is executed. Note that the rules R_1 and R_2 for the goals Exp and $SciOpp$ respectively ensure that Exp and $SciOpp$ are suspended (if active) when $Charge$ is activated. Once $Charge$ has succeeded (i.e., the battery is fully charged), it is dropped, and the goal $Battery$ returns to the Monitoring state. Exp then resumes execution, but before p_4 is completed, $SciOpp$ is activated, as *Curiosity* has spotted a rock which matches the science opportunity criterion. Exp is suspended, until $SciOpp$ is achieved. *Curiosity* completes the latter goal but, before it can resume Exp, the trigger for $Return$ is activated, as *Curiosity* has determined it has to leave now in order to return to the base by dusk. Hence Exp is aborted (which includes saving internally the partial results that have been obtained during the sol, represented here by the abort method $save$), and the action *return* is executed, which results in *Curiosity* being back at the base for the night.

It can be seen how our approach makes available to the agent the range of goal operations, and provides a formal specification for the goals over their lifecycles. Achievement (e.g., Exp) and maintenance (e.g., $Battery$) goals are accommodated, together with plans.

6.2 Extending the basic scenario

In the above basic scenario, *Curiosity* is programmed to return to its base at the end of every sol. We now consider a more sophisticated programming. When dusk approaches, *Curiosity* can stop its experiments and return to base as before. Alternatively, it can *suspend* its experiments, secure its position, wait until dawn, and resume the experiments. This is possible provided the rover has sufficient health status (e.g., battery level) for the night, and that forecast weather conditions (e.g., dust storms) are permissible for it.

The new goal, $Nighttime$, has signature: achieve$(nearly_dusk, \wedge, calm_weather)$ $dawnsecuring\_failure$. Like $Return$, $Nighttime$ is applicable when the end of the sol is approaching. Unless otherwise instructed, it is up to *Curiosity* to decide which of the two goals to adopt.

We suppose that the plan for $Nighttime$ consists of four steps, $n_1; n_2; n_3; n_4$, corresponding to suspending any current experiments, moving to a stable position, waiting for dawn, and then resuming any experiments. Other parts of *Curiosity*'s programming would accept any new science targets instructed from Earth for the new sol. The new targets might preempt some of previous targets (i.e., those not completed the previous day), since the highest priority science targets would be pursued. Further, since this plan does not include transmitting the partial results of any experiments suspended at dusk, it is important that the suspend and failure methods of $Nighttime$ schedule the communication of any partial results.

The initial goal context (in simplified notation, as before) is:

$$\langle Exp, \mathsf{P}, \epsilon\rangle, \langle SciOpp, \mathsf{P}, \epsilon\rangle, \langle Battery, \mathsf{P}, \epsilon\rangle, \langle Return, \mathsf{P}, \epsilon\rangle, \langle Nighttime, \mathsf{P}, \epsilon\rangle$$

An illustration of the expected sequence of *Curiosity*'s goal states is given in Table 3. We do not show goal $Return$ since $Nighttime$ will be adopted instead. Stages 1–22 progress as before. In stage 23, with dusk approaching, *Curiosity* activates $Nighttime$. It causes both Exp and $SciOpp$ to be suspended. Being a critical health goal, $Battery$ is not suspended. When dawn comes, in stage 27, *Curiosity* resumes both science goals.

We could study further more sophisticated scenarios. For example, a return to base might trigger other behaviour, such as a communication goal back to Earth that uses the base's high-bandwidth capability, and goals for scheduled maintenance tasks. Also, we may consider making $SciOpp$ either a purely reactive maintenance goal, or a 'permanently triggered' achievement goal; the latter case can be specified in our semantics simply by altering the rule that drops the goal when it succeeds to a rule that restores it to the Pending state. Our

Table 3 *Curiosity*'s sequence of goal states in the extended scenario

Stage	Experiment	Science	Battery	Nighttime
1	$\langle Exp, \mathsf{P}, \epsilon \rangle$	$\langle SciOpp, \mathsf{P}, \epsilon \rangle$	$\langle Battery, \mathsf{P}, \epsilon \rangle$	$\langle Nighttime, \mathsf{P}, \epsilon \rangle$
2	$\langle Exp, \mathsf{P}, \epsilon \rangle$	$\langle SciOpp, \mathsf{P}, \epsilon \rangle$	$\langle Battery, \mathsf{M}, \epsilon \rangle$	$\langle Nighttime, \mathsf{P}, \epsilon \rangle$
3	$\langle Exp, \mathsf{A}, \epsilon \rangle$	$\langle SciOpp, \mathsf{P}, \epsilon \rangle$	$\langle Battery, \mathsf{M}, \epsilon \rangle$	$\langle Nighttime, \mathsf{P}, \epsilon \rangle$
4	$\langle Exp, \mathsf{A}, p_1; p_2; p_3; p_4 \rangle$	$\langle SciOpp, \mathsf{P}, \epsilon \rangle$	$\langle Battery, \mathsf{M}, \epsilon \rangle$	$\langle Nighttime, \mathsf{P}, \epsilon \rangle$
5	$\langle Exp, \mathsf{A}, p_2; p_3; p_4 \rangle$	$\langle SciOpp, \mathsf{P}, \epsilon \rangle$	$\langle Battery, \mathsf{M}, \epsilon \rangle$	$\langle Nighttime, \mathsf{P}, \epsilon \rangle$
6	$\langle Exp, \mathsf{A}, p_3; p_4 \rangle$	$\langle SciOpp, \mathsf{P}, \epsilon \rangle$	$\langle Battery, \mathsf{M}, \epsilon \rangle$	$\langle Nighttime, \mathsf{P}, \epsilon \rangle$
7	$\langle Exp, \mathsf{A}, p_3; p_4 \rangle$	$\langle SciOpp, \mathsf{P}, \epsilon \rangle$	$\langle Battery, \mathsf{A}, MGP \rangle$	$\langle Nighttime, \mathsf{P}, \epsilon \rangle$
8	$\langle Exp, \mathsf{S}, p_3; p_4 \rangle$	$\langle SciOpp, \mathsf{P}, \epsilon \rangle$	$\langle Battery, \mathsf{A}, MGP \rangle$	$\langle Nighttime, \mathsf{P}, \epsilon \rangle$
9	$\langle Exp, \mathsf{S}, p_3; p_4 \rangle$	$\langle SciOpp, \mathsf{P}, \epsilon \rangle$	$\langle Battery, \mathsf{A}, Charge \rangle$	$\langle Nighttime, \mathsf{P}, \epsilon \rangle$
10	$\langle Exp, \mathsf{S}, p_3; p_4 \rangle$	$\langle SciOpp, \mathsf{P}, \epsilon \rangle$	$\langle Battery, \mathsf{A}, SGP \rangle$, $\langle Charge, \mathsf{P}, \epsilon \rangle$	$\langle Nighttime, \mathsf{P}, \epsilon \rangle$
11	$\langle Exp, \mathsf{S}, p_3; p_4 \rangle$	$\langle SciOpp, \mathsf{P}, \epsilon \rangle$	$\langle Battery, \mathsf{A}, SGP \rangle$, $\langle Charge, \mathsf{A}, \epsilon \rangle$	$\langle Nighttime, \mathsf{P}, \epsilon \rangle$
12	$\langle Exp, \mathsf{S}, p_3; p_4 \rangle$	$\langle SciOpp, \mathsf{P}, \epsilon \rangle$	$\langle Battery, \mathsf{A}, SGP \rangle$, $\langle Charge, \mathsf{A}, charge \rangle$	$\langle Nighttime, \mathsf{P}, \epsilon \rangle$
13	$\langle Exp, \mathsf{S}, p_3; p_4 \rangle$	$\langle SciOpp, \mathsf{P}, \epsilon \rangle$	$\langle Battery, \mathsf{A}, SGP \rangle$, $\langle Charge, \mathsf{A}, nil \rangle$	$\langle Nighttime, \mathsf{P}, \epsilon \rangle$
14	$\langle Exp, \mathsf{S}, p_3; p_4 \rangle$	$\langle SciOpp, \mathsf{P}, \epsilon \rangle$	$\langle Battery, \mathsf{A}, SGP \rangle$	$\langle Nighttime, \mathsf{P}, \epsilon \rangle$
15	$\langle Exp, \mathsf{S}, p_3; p_4 \rangle$	$\langle SciOpp, \mathsf{P}, \epsilon \rangle$	$\langle Battery, \mathsf{M}, \epsilon \rangle$	$\langle Nighttime, \mathsf{P}, \epsilon \rangle$
16	$\langle Exp, \mathsf{A}, \epsilon \rangle$	$\langle SciOpp, \mathsf{P}, \epsilon \rangle$	$\langle Battery, \mathsf{M}, \epsilon \rangle$	$\langle Nighttime, \mathsf{P}, \epsilon \rangle$
17	$\langle Exp, \mathsf{A}, p_3; p_4 \rangle$	$\langle SciOpp, \mathsf{P}, \epsilon \rangle$	$\langle Battery, \mathsf{M}, \epsilon \rangle$	$\langle Nighttime, \mathsf{P}, \epsilon \rangle$
18	$\langle Exp, \mathsf{A}, p_4 \rangle$	$\langle SciOpp, \mathsf{P}, \epsilon \rangle$	$\langle Battery, \mathsf{M}, \epsilon \rangle$	$\langle Nighttime, \mathsf{P}, \epsilon \rangle$
19	$\langle Exp, \mathsf{A}, p_4 \rangle$	$\langle SciOpp, \mathsf{A}, \epsilon \rangle$	$\langle Battery, \mathsf{M}, \epsilon \rangle$	$\langle Nighttime, \mathsf{P}, \epsilon \rangle$
20	$\langle Exp, \mathsf{S}, p_4 \rangle$	$\langle SciOpp, \mathsf{A}, \epsilon \rangle$	$\langle Battery, \mathsf{M}, \epsilon \rangle$	$\langle Nighttime, \mathsf{P}, \epsilon \rangle$
21	$\langle Exp, \mathsf{S}, p_4 \rangle$	$\langle SciOpp, \mathsf{A}, s_1; s_2 \rangle$	$\langle Battery, \mathsf{M}, \epsilon \rangle$	$\langle Nighttime, \mathsf{P}, \epsilon \rangle$
22	$\langle Exp, \mathsf{S}, p_4 \rangle$	$\langle SciOpp, \mathsf{A}, s_2 \rangle$	$\langle Battery, \mathsf{M}, \epsilon \rangle$	$\langle Nighttime, \mathsf{P}, \epsilon \rangle$
23	$\langle Exp, \mathsf{S}, p_4 \rangle$	$\langle SciOpp, \mathsf{A}, s_2 \rangle$	$\langle Battery, \mathsf{M}, \epsilon \rangle$	$\langle Nighttime, \mathsf{A}, \epsilon \rangle$
24	$\langle Exp, \mathsf{S}, p_4 \rangle$	$\langle SciOpp, \mathsf{S}, s_2 \rangle$	$\langle Battery, \mathsf{M}, \epsilon \rangle$	$\langle Nighttime, \mathsf{A}, n_1 \rangle$
25	$\langle Exp, \mathsf{S}, p_4 \rangle$	$\langle SciOpp, \mathsf{S}, s_2 \rangle$	$\langle Battery, \mathsf{M}, \epsilon \rangle$	$\langle Nighttime, \mathsf{A}, n_2 \rangle$
26	$\langle Exp, \mathsf{S}, p_4 \rangle$	$\langle SciOpp, \mathsf{S}, s_2 \rangle$	$\langle Battery, \mathsf{M}, \epsilon \rangle$	$\langle Nighttime, \mathsf{A}, n_3 \rangle$
27	$\langle Exp, \mathsf{A}, p_4 \rangle$	$\langle SciOpp, \mathsf{A}, s_2 \rangle$	$\langle Battery, \mathsf{M}, \epsilon \rangle$	$\langle Nighttime, \mathsf{A}, n_4 \rangle$
28	$\langle Exp, \mathsf{A}, p_4 \rangle$	$\langle SciOpp, \mathsf{A}, s_2 \rangle$	$\langle Battery, \mathsf{M}, \epsilon \rangle$	$\langle Nighttime, \mathsf{A}, nil \rangle$

semantics ensures that *Curiosity* performs operations correctly on its goals, according to the decisions that the agent's reasoning makes.

6.3 Implementation

We have developed a prototype implementation of the full CAN rules for the four states given in Fig. 2, and we have used it to verify the example described in this section. This implementation, which we refer to as *Orpheus*, consists of around 750 lines of Prolog, and has been tested under Ciao and SWI-Prolog. This has been a useful tool for testing our ideas, and is available from the authors.[6]

[6] http://www.cs.rmit.edu.au/~jah/orpheus.

It should be stressed that Orpheus is not intended as (yet another) agent programming language. Rather, it is a tool to be used to explore the way in which changes of goal state can be managed, and as a means of providing experimental evaluation. In particular, the main focus of the implementation is in managing the states of goals, applying the appropriate rules, and ensuring that the four phases discussed in Sect. 5.2 are followed. Once the CAN execution rules were implemented, it was a simple task to translate the goal state transition rules into executable code in Prolog.

In order to verify the examples in the *Curiosity* scenario, we chose to associate plans with goals by means of some simple Prolog rules, and similarly for the handling of events. However, this choice could easily be changed to more sophisticated mechanisms. The main purpose of the implementation is to study the goal state changes, and not how plans are generated. Hence, it seemed appropriate to keep simple the less important mechanisms.

7 Related work

In this section we survey related work and place our contribution in context.

7.1 Goal types

The distinction between achievement and maintenance goals has early roots in AI literature. For instance, Cohen and Levesque [10] note that an agent adopts an achievement goal to make true a state of the world (that it believes is currently false); whereas an agent adopts a maintenance goals to keep true a condition in the world (that it believes is currently true). The distinction parallels the distinction between liveness and safety properties in the verification literature [12]. An agent's attitude towards a goal, be it a goal of accomplishment or a goal of monitoring, determines what is called a *commitment strategy* (see the discussion in, e.g., [42]), i.e., what kind of conditions (for dropping, aborting, etc) are actually checked or ignored by the agent.

The different types of goals found in the literature and in implemented agent systems are surveyed by Braubach et al. [7]. While there is broad agreement about perform and achieve goals, less attention has been directed towards maintain goals. The reactive and proactive semantics for maintenance goals is explored by Duff et al. [16]. However, they do not consider aborting or suspending goals, and do not give formal semantics for the behaviour of maintenance goals. Other authors give proposals for the semantics of maintenance [3,49].

7.2 Goal states and transitions

Mechanisms for adopting and dropping goals, and generating plans for them, have been variously explored at both the semantic theoretical and implemented system levels; we do not cite here the extensive body of work. In our earlier work, we formalized the mechanisms for the operations of aborting, suspending, and resuming goals [36,37]. However, that work considered only achieve goals. Our analysis is that the literature lacks a state-transition specification for all classes of goals that accounts for the current mechanisms for aborting and suspending. Beyond our scope are recent examples of exploring goal failure and re-planning [30,34].

Bordini and Hübner [5] provide a semantics for the agent system Jason's 'internal actions', including its mechanism for handling plan failure. Inasmuch as they act to modify internal state, these internal actions are akin to the internals of our abstract goal states [40].

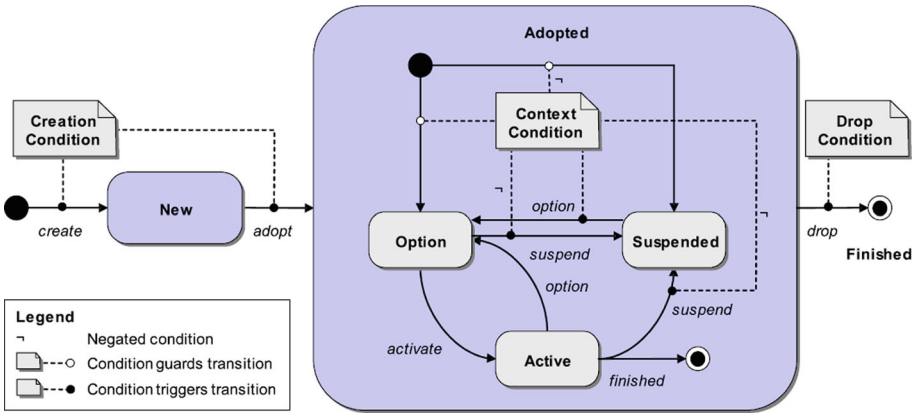

Fig. 3 Abstract goal state transition system of the Jadex framework (from [7])

Braubach et al. [7] build the Jadex agent system [28] on an explicit state-based manipulation of goals. Figure 3 (taken from [7]), illustrates the basic generic goal state transition system that they propose. Goals begin in a New state and when adopted, they move to the Option state (akin to our Pending). If the 'context' condition, that is, the condition which captures the circumstances when the goal is applicable, is true, the deliberation mechanism may initiate the goal processing and the goal transitions to the Active state (akin to our own Active). The deliberation mechanism may also deactivate a goal which moves it back to Option from Active. If the context is not true or becomes false, the goal moves to the Suspended state, from which, it moves to the Option state when the context becomes true again. They further specialise this basic transition system to the specific goal types. Whilst there are similarities in their proposal to ours, there are a number of significant differences:

- Their 'suspend' and 'resume' is based on an applicability condition whilst we propose deliberation-directed suspension and resumption. This is necessary since goals may be suspended due to the agent pursuing higher priority goals, lack of resources, or even user intervention.
- When an achieve goal is suspended, in their framework all associated subgoals and plans are terminated and a process history is maintained for determining which plans to restart when the goal is resumed. They do not consider clean up actions which are often necessary as we have argued [37]. Similarly for when goals are aborted [36]. The details of the Active and Suspend states in the transition system we propose incorporates the mechanisms for aborting, suspending and resuming goals described in [36,37].
- The process to re-establish the maintenance condition of a maintain goal is part of the maintenance goal in their framework. In our approach, we generate a sub-goal of type achieve to re-establish the condition which is treated as a normal achieve goal. This allows for the sub-goal (or the plans/goals below it) to be suspended/resumed or aborted independent of the maintain goal. The abort is of particular importance since aborting a plan to re-establish the maintenance condition may result in a different plan being executed.

The aim of Braubach et al. is to define a principled yet pragmatic foundation for the Jadex system; no attempt is made for a generic formalization with a uniform set of operations on

goals at an abstract representational level. Braubach et al. [6] discuss *long-term* goals, which may be considered as an input for determining when a goal should be dropped, aborted or suspended; here we are concerned with the consequences of such decisions, rather than the reasons that they are made.

van Riemsdijk et al. [46,48] provide semantics for goals, based on default logic, emphasizing that, while the set of an agent's goals need not be consistent, its set of intentions must be. This and similar work is complementary to ours, in that we do not consider the process by which the agent decides whether to adopt a goal and whether to adopt an intention (plan) for it. The same authors [13,14] expand their analysis of declarative goals to perform goals, achieve goals, and maintain goals, providing a logic-based operational semantics.

van Riemsdijk et al. [49] present a generic, abstract, type-neutral goal model consisting of suspend and active states. Their two states can be thought of as "not currently executing a plan" and "currently executing a plan", respectively. Their work, which like ours encompasses achieve, perform, query, and maintain goals, has a less detailed accounting for maintenance goals and for aborting and suspending than our work. Further, we argue that the states of non-execution and suspension should be distinguished, and that goals should be created into the Pending state, and not the Suspend state. Winikoff et al. [51] extend the work of [49] with new types of time-varying goals, such as 'achieve and maintain', sketching a semantics in LTL.

Contrary to van Riemsdijk et al., but agreeing with Braubach et al., it is worth remarking on the value of Pending state as we see it. On the one hand, for achievement goals the state distinguishes between a goal having in-execution plans associated (Active) and not (Pending). On the other hand, for maintenance goals it distinguishes between a goal monitoring the maintenance condition (Monitoring; Active if responding) versus not (Pending). Indeed, we see it as a natural state, in the sense that there are three natural states of a maintenance goal: not monitoring the maintenance condition (Pending), monitoring but with no violation (Monitoring), violation detected (Active). There are some interesting examples along the lines of keeping *Curiosity* out of high winds. When there are no winds, there is nothing for the rover to be concerned about. When there are high winds, the relevant maintenance goal would be in the Monitoring state, and the proactive mechanism would prevent any action taking place that would expose it to high winds.

Morandini et al. [25] use the generic goal model of van Riemsdijk et al. to reduce the semantic gap between design-time goal models and run-time agent implementations. Their operational semantics is focused on providing an account of the relationship between a goal and its subgoals, including success conditions which are not necessarily the same as those of the subgoals. Our work likewise encompasses dynamic achievement of a goal according to logical conditions, enabled by a subgoaling mechanism. Crucially, since we are concerned with execution, our semantics accounts for plans as well as goals. This means that our goal states contain finer distinctions, and in particular the sub-division of the Active and Suspended states. Our work is further distinguished by a richer range of operations that may be applied to a goal (e.g., a richer semantics for suspending a goal and its children; aborting as well as failing), and by the inclusion of proactive maintenance goals.

Khan and Lespérance [23] tackle goal dynamics for prioritized goals through a logical approach. Their focus is to ensure that active goals are consistent with each other and the agent's knowledge. Lorini et al. [24] study in detail the dynamics of goals and plans under changes to the agent's beliefs. Such works that enable an agent to reconsider its goals in the light of belief updates are complementary to our work, and beyond our scope here.

8 Conclusion and future work

Management of goals is central to intelligent agents in the BDI tradition. Through this article we have provided an operational semantics for goal management across the common goal types in the literature. The key contributions of our rule-based operational semantics for goal states and transitions are (1) to encompass both goals of accomplishment and rich goals of monitoring, (2) to provide a specification of abort and suspend for all the common goal types, and (3) to account for plan execution as well as the dynamics of subgoaling. To the best of our knowledge, this is the first time that these three aspects have been combined into one semantics.

By developing the formal operational semantics for our generic framework in the agent language CAN [31], we have not been tied to any particular agent implementation. An agent system implementing the semantics we describe may correctly apply any of the permitted operations to a goal of any type in any state. These operations include principled activation, aborting, suspending, and resuming of goals. We have developed a prototype implementation of the semantics in Prolog.

While the goal life-cycle is relatively simple at a high level (Fig. 2), pragmatic implementations of our framework will want to encapsulate as much of the complexity as possible—for instance, detailed substates for the Active state—from the agent designer (programmer), by encapsulation in the mechanisms of the agent execution framework (compare [5,6]). We have developed our framework with flexibility in mind, so that an agent designer can use our framework in a number of different contexts. For example, the use of the procedure *mer* for means-end reasoning means that our framework can be used with any reasonable method for associating goals and plans. Similar flexibility applies to the means by which the agent makes decisions about whether to drop, suspend, resume, or abort one of its goals.

Our emphasis on goals exploits a natural synergy with goal-oriented software engineering [45] and the modelling of business rules [8,9]. In both cases, the use of goals is fundamental and a comprehensive semantics is important. Our semantics not only clarifies how an agent can manage its goals, based on the decisions that it chooses to make, but it provides a path for correctness verification of agent behaviour, through verification of Orpheus programs (compare with other approaches formal verification of agents logics and systems [12]). Given some agent decision-making behaviour, it would be beneficial in business and other settings to be able to provide guarantees about the agent's operation. This article has provided an operational semantics for the agent's goal operations, which will form an essential element of a formal system in which such proofs can be developed.

We have accounted for the life-cycle of each goal type and have not sought to address overall agent deliberation, plan deliberation, resource management, or plan scheduling. Thus far we have examined the same questions as Braubach et al. [7]; future work is to address the other questions they pose. Likewise, we have not considered failure handling and exceptions. Our work is complementary to works that consider generic or application-specific reasoning about goal interactions, such as Thangarajah and Padgham [41] and Shaw et al. [32], works that consider goal generation, such as da Costa Pereira and Tettamanzi [11], and works that consider goal and plan selection, such as Hindriks et al. [19] and Lorini et al. [24]. We think there is potential in the exploration of such links.

Acknowledgments We are grateful to an anonymous reviewer for this point. We thank Lin Padgham, Sebastian Sardiña, and the participants of the DALT'10 workshop for discussions. We thank the reviewers of the earlier versions of this article for thoughtful and detailed comments which have improved the work. JT acknowledges the support of the Australian Research Council and Agent Oriented Software Pty. Ltd. under Grant LP0453486. NYS acknowledges the support of the University Research Board of the American

University of Beirut, and thanks the Operations group at the Judge Business School and the fellowship at St Edmund's College, Cambridge. The work of DNM and NYS through SRI International was supported in part by the Defense Advanced Research Projects Agency (DARPA) under Contract No. FA8750-07-D-0185/0004. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of DARPA, or the Air Force Research Laboratory.

References

1. Anderson, C. (1990, August). The first rover on Mars—The Soviets did it in 1971. The Planetary Report.
2. Bajracharya, M., Maimone, M. W., & Helmick, D. (2008). Autonomy for Mars rovers: Past, present, and future. *Computer*, *41*(12), 45–50.
3. Baral, C., Eiter, T., Bjaereland, M., & Nakamura, M. (2008). Maintenance goals of agents in a dynamic environment. *Artificial Intelligence*, *172*(12–13), 1429–1469.
4. Bordini, R. H., Fisher, M., Visser, W., & Wooldridge, M. (2004). State-space reduction techniques in agent verification. In *Proceedings of AAMAS'04* (pp. 896–903), New York.
5. Bordini, R.H., Hübner, J. F. (2010). Semantics for the Jason variant of AgentSpeak (plan failure and some internal actions). In *Proceedings of ECAI'10* (pp. 635–640), Lisbon.
6. Braubach, L., & Pokahr, A. (2009). Representing long-term and interest BDI goals. In *Proceedings of 7th international workshop on programming multi-agent systems (ProMAS'09)*, Budapest.
7. Braubach, L., Pokahr, A., Moldt, D., & Lamersdorf, W. (2004). Goal representation for BDI Agent systems. In *Proceedings of 2nd international workshop on programming multi-agent systems (ProMAS'04)* (pp. 9–20), New York.
8. Burmeister, B., Arnold, M., Copaciu, F., & Rimassa, G. (2008). BDI-agents for agile goal-oriented business processes. In *Proceedings of AAMAS'08 (industry track)* (pp. 37–44), Estoril.
9. Chopra, A., Dalpiaz, F., Giorgini, P., & Mylopoulos, J. (2010). Modeling and reasoning about service-oriented applications via goals and commitments. In *Proceedings of 22nd conference on advanced information systems engineering (CAiSE'10)* (pp. 113–128), Hammamet.
10. Cohen, P. R., & Levesque, H. J. (1990). Intention is choice with commitment. *Artificial Intelligence*, *42*, 213–261.
11. da Costa Pereira, C., & Tettamanzi, A. (2010). Belief–goal relationships in possibilistic goal generation. In *Proceedings of ECAI'10* (pp. 641–646), Lisbon.
12. Dastani, M., Hindriks, K. V., Meyer, J. J. (Eds.). (2010). *Specification and verification of multi-agent systems*. New York: Springer.
13. Dastani, M., van Riemsdijk, M. B., & Meyer, J. J. C. (2006). Goal types in agent programming. In *Proceedings of AAMAS'06* (pp. 1285–1287), Hakodate.
14. Dastani, M., van Riemsdijk, M. B., & Meyer, J. J. C. (2006). Goal types in agent programming. In *Proceedings of ECAI'06* (pp. 220–224), Riva del Garda.
15. Sardiña, S., de Silva, L., & Padgham, L. (2006). Hierarchical planning in BDI agent programming languages: A formal approach. In *Proceedings of AAMAS'06* (pp. 1001–1008), Hakodate.
16. Duff, S., Harland, J., & Thangarajah, J. (2006). On proactivity and maintenance goals. In *Proceedings of AAMAS'06* (pp. 1033–1040), Hakodate.
17. Estlin, T., Rabideau, G., Mutz, D., & Chien, S. (2000). Using continuous planning techniques to coordinate multiple rovers. *Electronic Transactions on Artificial Intelligence*, *4*(A), 45–57.
18. Georgeff, M., & Rao, A. (1998). Rational software agents: From theory to practice. In *Agent technology: Foundations, applications, and markets* (Chap. 8, pp. 139–160). New York: Springer.
19. Hindriks, K. V., van der Hoek, W., & van Riemsdij, M. B. (2009). Agent programming with temporally extended goals. In *Proceedings of AAMAS'09* (pp. 137–144), Budapest.
20. Hindriks, K. V., & van Riemsdijk, M. B. (2008). Using temporal logic to integrate goals and qualitative preferences into agent programming. In *Proceedings of 6th international workshop on declarative agent languages and technologies (DALT'08)* (pp. 173–189), Estoril.
21. Jet Propulsion Laboratory: Mars Science Laboratory: Mission science goals. (2012). http://mars.jpl.nasa.gov/msl/mission/science/goals/. Retrieved August 2012.
22. Jet Propulsion Laboratory: Mars Science Laboratory/Curiosity. (2012). Tech. Rep. JPL 400–1491 8/12. National Aeronautics and Space Administration, Pasadena.
23. Khan, S. M., & Lespérance, Y. (2010). A logical framework for prioritized goal change. In *Proceedings of AAMAS'10* (pp. 283–290), Toronto.
24. Lorini, E., van Ditmarsch, H. P., & Lima, T. D. (2010). A logical model of intention and plan dynamics. In *Proceedings of ECAI'10* (pp. 1075–1076), Lisbon.
25. Morandini, M., Penserini, L., & Perini, A. (2009). Operational semantics of goal models in adaptive agents. In *Proceedings of AAMAS'09* (pp. 129–136), Budapest.

26. Myers, K. L., & Morley, D. N. (2001). Human directability of agents. In *Proceedings of first international conference on knowledge capture (K-CAP'01)* (pp. 108–115), Victoria.
27. Padgham, L., & Winikoff, M. (2004). *Developing intelligent agent systems: A practical guide*. New York: Wiley.
28. Pokahr, A., Braubach, L., & Lamersdorf, W. (2005). Jadex: A BDI reasoning engine. In *Multi-agent programming* (pp. 149–174). Heidelberg: Springer.
29. Rao, A. S., & Georgeff, M. P. (1992). An abstract architecture for rational agents. In *Proceedings of KR'92* (pp. 439–449), Cambridge.
30. Sardiña, S., & Padgham, L. (2007). Goals in the context of BDI plan failure and planning. In *Proceedings of AAMAS'07* (pp. 16–23), Honolulu.
31. Sardiña, S., & Padgham, L. (2011). A BDI agent programming language with failure handling, declarative goals, and planning. *Journal of Autonomous Agents and Multi-Agent Systems*, *23*(1), 18–70.
32. Shaw, P. H., Farwer, B., & Bordini, R. H. (2008). Theoretical and experimental results on the goal-plan tree problem. In *Proceedings of AAMAS'08* (pp. 1379–1382), Estoril.
33. Siebra, C., Tate, A., & Lino, N. Q. (2004). Planning and representation of joint human-agent space missions via constraint-based models. In *Proceedings of international workshop on planning and scheduling in space (IWPSS'04)*.
34. de Silva, L., Sardiña, S., & Padgham, L. (2009). First principles planning in BDI systems. In *Proceedings of AAMAS'09* (pp. 1105–1112), Budapest.
35. Singh, D., Sardiña, S., & Padgham, L. (2010). Extending BDI plan selection to incorporate learning from experience. *Robotics and Autonomous Systems*, *58*(9), 1067–1075.
36. Thangarajah, J., Harland, J., Morley, D., & Yorke-Smith, N. (2007). Aborting tasks in BDI agents. In *Proceedings of AAMAS'07* (pp. 8–15), Honolulu.
37. Thangarajah, J., Harland, J., Morley, D., & Yorke-Smith, N. (2008). Suspending and resuming tasks in BDI agents. In *Proceedings of AAMAS'08* (pp. 405–412), Estoril.
38. Thangarajah, J., Harland, J., Morley, D., & Yorke-Smith, N. (2010). On the life-cycle of BDI agent goals. In *Proceedings of ECAI'10* (pp. 1031–1032), Lisbon.
39. Thangarajah, J., Harland, J., Morley, D., & Yorke-Smith, N. (2010). Operational behaviour for executing, suspending and aborting goals in BDI agent systems. In *Proceedings of 8th international workshop on declarative agent languages and technologies (DALT'10)* (pp. 1–17), Toronto.
40. Thangarajah, J., Harland, J., Morley, D., & Yorke-Smith, N. (2011). Operational behaviour for executing, suspending, and aborting goals in BDI agent systems. In A. Omicini, S. Sardina, & W. Vasconcelos (Eds.), *Declarative agent languages and technologies VIII*. Lecture notes in computer science (Vol. 6619, pp. 1–21). Berlin: Springer.
41. Thangarajah, J., & Padgham, L. (2011). Computationally effective reasoning about goal interactions. *Journal of Automated Reasoning*, *47*(1), 17–56.
42. Thangarajah, J., Padgham, L., & Harland, J. (2002). Representation and reasoning for goals in BDI agents. In *Proceedings of twenty-fifth Australasian computer science conference (ACSC'02)* (pp. 259–265), Melbourne.
43. Thangarajah, J., Padgham, L., & Winikoff, M. (2003). Detecting and avoiding interference between goals in intelligent agents. In *Proceedings of IJCAI'03* (pp. 721–726), Acapulco.
44. Thangarajah, J., Padgham, L., & Winikoff, M. (2003). Detecting and exploiting positive goal interaction in intelligent agents. In *Proceedings of AAMAS'03* (pp. 401–408), Melbourne.
45. van Lamsweerde, A. (2001). Goal-oriented requirements engineering: A guided tour. In *Proceedings of 5th IEEE international symposium on requirements engineering (RE'01)* (pp. 249–263), Toronto.
46. van Riemsdijk, M. B., Dastani, M., & Meyer, J. J. C. (2005). Semantics of declarative goals in agent programming. In *Proceedings of AAMAS'05* (pp. 133–140), Utrecht.
47. van Riemsdijk, M. B., Dastani, M., Meyer, J. J. C. (2005). Subgoal semantics in agent programming. In *Proceedings of 12th Portuguese conference on artificial intelligence (EPIA'05)* (pp. 548–559), Covilhã.
48. van Riemsdijk, M. B., Dastani, M., & Meyer, J. J. C. (2009). Goals in conflict: Semantic foundations of goals in agent programming. *Journal of Autonomous Agents and Multi-Agent Systems*, *18*(3), 471–500.
49. van Riemsdijk, M. B., Dastani, M., & Winikoff, M. (2008). Goals in agent systems: A unifying framework. In *Proceedings of AAMAS'08* (pp. 713–720), Estoril.
50. Washington, R., Golden, K., Bresina, J., Smith, D. E., Anderson, C., & Smith, T. (1999). Autonomous rovers for Mars exploration. In *Proceedings of IEEE aerospace conference* (pp. 237–251), Aspen.
51. Winikoff, M., Dastani, M., & van Riemsdijk, M. B. (2010). A unified interaction-aware goal framework. In *Proceedings of ECAI'10* (pp. 1033–1034), Lisbon.
52. Winikoff, M., Padgham, L., Harland, J., & Thangarajah, J. (2002). Declarative and procedural goals in intelligent agent systems. In *Proceedings of KR'02* (pp. 470–481). Toulouse.
53. Wooldridge, M. (2002). *An introduction to multiagent systems*. Chichester: Wiley.