Quantifying the Progress of Goals in Intelligent Agents

Harland, James; Thangarajah, John; Yorke-Smith, Neil

DOI
10.1504/IJAOSE.2022.122640

Publication date
2022

Document Version
Final published version

Published in
International journal of agent-oriented software engineering

Quantifying the progress of goals in intelligent agents

James Harland and John Thangarajah

RMIT University,
Melbourne, 3000, Australia
Email: james.harland@rmit.edu.au
Email: john.thangarajah@rmit.edu.au

Neil Yorke-Smith*

Delft University of Technology,
2600 GA Delft, The Netherlands
Email: n.yorke-smith@tudelft.nl
and
American University of Beirut,
Beirut 1107 2020, Lebanon
*Corresponding author

Abstract: Deliberation over goals is a fundamental feature of intelligent agent systems. In this article we provide pragmatic but principled mechanisms for quantifying the level of completeness of goals in a belief-desire-intention (BDI) agent. Our approach leverages previous work on resource and effects summarisation which we extend by accommodating both dynamic resource summaries and goal effects, while also allowing a non-binary quantification of goal completeness. We treat both goals of accomplishment (achievement goals) and goals of monitoring (maintenance goals). We reconcile such practical computation of progress estimates of goals of both types with an earlier theoretical perspective on BDI goal completeness, and thus extend the theoretical framework to include maintenance goals. Our computational mechanisms have been implemented in the abstract agent language CAN. We also provide a detailed example in an autonomous rover domain.

Keywords: agent-based systems; maintenance goals; belief-desire-intention; BDI; goal reasoning.

Reference to this paper should be made as follows: Harland, J., Thangarajah, J. and Yorke-Smith, N. (2022) 'Quantifying the progress of goals in intelligent agents', *Int. J. Agent-Oriented Software Engineering*, Vol. 7, No. 2, pp.108–151.

Biographical notes: James Harland is a Professor in Computational Logic at RMIT University. He is known internationally for his work on intelligent agent systems, automated reasoning, logic programming, Turing machines and computer science education. His work centres on the relationship between computation and logical reasoning, particularly in the areas of mathematical logic and proof theory.

John Thangarajah is the Research Director for the Centre for AI Research & Innovation at RMIT University, Melbourne, Australia. His interests are in autonomous multi agent systems (how do we build and construct intelligent systems), agent reasoning (how can programs behave in smart ways),

intelligent conversation systems (how can systems interact intelligently with humans), agent testing (providing assurance that the systems work) and human machine teaming (how can humans and machine work collaboratively).

Neil Yorke-Smith is an Associate Professor of Socio-Technical Algorithmics in the Faculty of Electrical Engineering, Mathematics and Computer Science at the Delft University of Technology (TU Delft). His research focuses on intelligent decision making in complex socio-technical situations, with a particular interest in agent-based methodologies and automated planning and scheduling. The author of over 100 scholarly publications, he is a senior member of the Association for the Advancement of Artificial Intelligence (AAAI) and of the Association for Computing Machinery (ACM).

This paper is a revised and expanded version of a paper entitled 'Quantifying the completeness of goals in BDI agents' presented at Proc. of ECAI'14, Prague, Czech Republic, August 2014; 'Towards quantifying the completeness of BDI goals' presented at Proc. of AAMAS'14, Paris, France, May 2014; 'Estimating the progress of maintenance goals' presented at Proc. of AAMAS'15, Istanbul, Turkey, May 2015.

1 Introduction

Intelligent agent technology has become a popular means for developing applications that exhibit autonomous behaviour: for instance, unmanned vehicles (Wilson et al., 2018; Jha et al., 2018), electronic trading agents (Collins et al., 2009) and tactical simulation systems (Evertsz et al., 2015). Among the many agent architectures used to develop such autonomous systems, the belief-desire-intention (BDI) model of agency (Georgeff and Rao, 1998) is mature and has influenced many agent programming languages such as JACK (Winikoff, 2005), SPARK (Morley and Myers, 2004), GOAL (Hindriks, 2009) and GORITE (Rönnquist, 2007).

Goals are an essential concept in BDI agent systems. Autonomous agents, such as a Mars rover robot, are designed to work in dynamic environments. Hence it is crucial for an agent to deliberate over its goals and manage them appropriately (Thangarajah and Padgham, 2011; Harland et al., 2017; Wilson et al., 2018). The two main types of goals are *achievement* and *maintenance* goals. The most common type, achievement goals, are adopted by the agent to achieve a particular state (such as, for the Mars rover, performing some particular science experiment), and then dropped once this state has been achieved. The other type of goal, increasingly recognised for its importance, is maintenance goals (Duff et al., 2014). A maintenance goal has a particular state of the world that the agent seeks to maintain, i.e., the state must be true, and kept this way indefinitely. For example, the Mars rover would be well advised to ensure that wherever it travels, it always maintains sufficient battery charge for the journey back to its base. Hence, the agent's goal is to monitor the *maintain condition*, to ensure if possible that it never becomes false, and to act to restore it, if it is so violated.

Typically in BDI systems, goal accomplishment is discrete: a goal is either complete (usually, a plan for it has succeeded), or it is incomplete (whether execution of a plan or plans for it has begun or not) (Georgeff and Rao, 1998; van Riemsdijk and Yorke-Smith, 2010). Hence, when deliberating about its goals – such as the decision about which goal

to focus on next – an agent is limited to a coarse binary approximation of goal completeness. If the agent were able to compute a finer-grained approximation of the progress of its goals, it could make more nuanced and potentially more suitable decisions. For example, when resolving goal conflicts (Thangarajah and Padgham, 2011), the agent may choose to continue with one achievement goal that is more complete than another. When reasoning about deadlines, such as ensuring all activities are complete by the end of the day, it is also useful to be able to estimate how close the current goals are to completion, so that the usefulness of a minor extension of time can be evaluated. Similarly, an important opportunity (or threat) may arise before a goal is completed, and hence it will be useful to know whether or not a minor delay is appropriate, or it is better to suspend all current activities in favour of this new opportunity.

While the notion of partially-complete goals has been defined in Zhou et al. (2008) and van Riemsdijk and Yorke-Smith (2010), reasoning frameworks to date have largely left unanswered *how* to compute the level of completeness of a goal in a realistic and principled manner. Moreover, maintenance goals have not been considered. In our preliminary work we presented an approach to computing partial completeness estimates for an achievement goal (Thangarajah et al., 2014b, 2014a), and presented some ideas for what it means to estimate the completeness of a maintenance goal (Thangarajah et al., 2015).

To address this shortfall in the literature, the aim of this article is to provide a principled and general approach that can be used computationally to quantify measures of completeness for both achievement and maintenance goals. Hence we provide a foundation for subsequent reasoning by the agent using this information, i.e., the agent's (intention) deliberation mechanisms. This article thus enables automated reasoning for strategic planning, goal prioritisation, resource allocation and plan selection, among other deliberative tasks.

We make three main contributions. The first is to present detailed mechanisms to compute completeness measures of achievement goals. These mechanisms are based on goal resources and effects. The second contribution is to establish what it means for maintenance goals to be 'complete' and to present mechanisms for efficiently computing completeness measures for maintenance goals. The methods we introduce for resource-based estimation of the completeness of achievement goals are also suitable to predict potential violations of resource-related maintenance goals. This makes it easier to specify more sophisticated interactions between the two types of goals. For example, if it is known that continuing to pursue a given achievement goal will cause a maintenance goal to be violated, then it seems rational to suspend or abort the achievement goal rather than knowingly allowing the maintenance goal become false. In a similar manner, given a concurrent set of achievement goals being pursued, we may be able to predict that concurrent success of all such goals will falsify a maintenance goal, but yet identify a subset of these achievement goals which will not. It is also possible to use completion estimates as a means of determining priorities between goals, such as favouring those which are closest to completion, or in fact doing the opposite in order to satisfy fairness constraints. The third contribution is to reconcile this practical computation of progress estimates – in both goal types, achievement and maintenance – with the theoretical perspective on BDI goal completeness of van Riemsdijk and Yorke-Smith (2010), and to thus implicitly extend the theoretical framework to include maintenance goals. At the same time we instantiate that framework to a computationally-feasible approach.

Our contribution is foundational and addresses the goals of a single agent considered separately. That is, we do not address interaction effects between goals, incomplete information about the future, and suspension. As with most BDI agent approaches, we assume that the agent's beliefs are generally true, but not guaranteed to be so, and that goals, plans and actions generally succeed, but are also not guaranteed to do so. In other words, our agents are generally optimistic, but prepared for (occasional) failures. We discuss future extensions at the end of the article. Our work is intended to provide a snapshot of the agent's current activities as a means of assisting in such deliberations, and in particular whether to continue its current activities, or to abort or suspend them.

The article is structured as follows. Section 2 reviews the literature. Section 3 provides necessary background for understanding our contribution. Section 4 presents the autonomous rover scenario. Section 5 proposes means to quantify the completeness of achievement and of maintenance goals. Section 6 illustrates these mechanisms on the scenario. Section 7 discusses from a theoretical perspective. Section 8 summarises the article and presents future directions.

2 Related work

An agent reasoning about its goals is a long-standing area of agent design and engineering, and one which is seeing a resurgence of research (Roberts et al., 2018; Aha, 2018). In this section we survey the notion of goal completeness and progress, paying attention to the key previous work on which we build. We include related fields such as AI planning.

We start with conceptual foundations of goals. Whereas goals in agent programming languages are not customarily defined to allow for partial completeness, Holton, from a philosophical perspective, argues for the existence of 'partial intentions', a concept spanning both desires and goals (Holton, 2008). Holton's partial intentions have a sense similar to Bratman's (1978) (although seeming to allow multiple live plans simultaneously); our focus is on partial completeness of goals in a practical agent system. Towards the latter, Haddawy and Hanks (1992) made an early contribution, defining a function from propositions to the real number, which represents the degree of satisfaction of a goal.

While partial completeness is less common in the AI literature, goals have commonly been associated with a utility, priority or preference in the agents literature (e.g., Huang and Bell, 1997; Hindriks et al., 2008; Khan and Lespérance, 2010) and in the automated planning literature (e.g., Do et al., 2007; Hsu et al., 2007). The purpose is usually for a form of intention selection (see, e.g., Visser et al., 2016): which goals to prioritise or pursue, or which plan/action to select and execute.

Vukovic and Robinson (2005) adopt Haddawy and Hanks' definition of the degree of goal satisfaction, for the purpose of context-aware goal transformation. In the setting of robust web services, these authors reformulate failed goals using a temporal planner. Their focus revolves around context and an ontology for it rather than agent autonomy.

Letier and van Lamsweerde (2004) are instead interested in the setting of requirements engineering. These authors determine the probabilistic partial goal satisfaction of alternative system designs. Our work shares the use of application-specific measures and the upwards and downwards propagation of them.

Thangarajah et al. (2007) explore multiple criteria that an agent may include in its goal deliberation, including utility, preference, deadline, resource considerations, goal interactions, effort to date, and likelihood of success. Although they describe a dynamic constraintbased reasoning mechanism, these authors also do not explicitly consider reasoning with partially-complete goals.

Based on their earlier work (Thangarajah et al., 2002, 2003), Thangarajah and Padgham (2011) study goal interactions, both positive (synergy) and negative (conflicts). Their work considers action effects as simple Boolean predicates. These authors define the *Goal-Plan Tree* (GPT) structure of alternating layers of goal and plan-nodes, and use this structure to inform deliberation such as goal adoption and plan selection. The reasoning centres around the use of resource and effect summaries annotated on GPT nodes and dynamically updated as execution proceeds. We will use GPTs in this article.

Building on Thangarajah et al., Morley et al. (2006) further develop reasoning in a BDI agent over GPT structures. They provide an algorithm for an agent to update resource estimates on GPTs accommodating resource bound information, parameterised goals, and rich plan constructs. Dynamic resource estimation will be leveraged in our contribution. Specifically, we use dynamic updating of GPT resource summaries.

Subsequently, Jiang et al. (2014) reason with preferences over GPT structures. Yao et al. (2016a, 2016c) focus on the executability of BDI intentions formulated in GPTs. Again unlike our work, none of these authors, nor Morley et al. (2006), explicitly consider reasoning with measures of goal completeness.

Different to the GPT line of work, Zhou and Chen (2004) adopt instead a logical approach. In contrast to others, these authors do seek a notion of partial completeness. They define semantics for partial implication of desirable propositions (Zhou and Chen, 2004). Zhou et al. (2008) investigate partial goal satisfaction on the basis of this logical semantics, viewing a goal as being completed when a (possibly disjunctive) proposition is achieved according to the logic. These authors focus on application of different notions of partial implication to goal modification in the context of belief change. We share a similar motivation, but our objective is a quantitative, grounded representation of partial satisfaction integrated into a reasoning framework.

This brings us to the two most related works. van Riemsdijk and Yorke-Smith (2010) formalised the concept of a partially-complete goal for a BDI-like agent. These authors captured partial satisfaction of an achievement goal using a progress metric, and a minimum value that the goal must attain for the agent to consider it completely satisfied. The authors described agent reasoning using such a representation, but did not provide any detailed computational mechanisms. We provide a mechanism to efficiently compute domain-specific measures of completeness, which is necessary if van Riemsdijk and Yorke-Smith (2010, 2012) framework's is to be useful in practice. We will discuss the relationship between the work of van Riemsdijk and Yorke-Smith (2010, 2012) and our work in more detail in Section 7.3.

Thangarajah et al. (2014b, 2014a) presented an approach to computing partial completeness estimates for an achievement goal, and some ideas for what it means to estimate the completeness of a maintenance goal (Thangarajah et al., 2007). We build on this preliminary work in this article. We provide a detailed computational approach for maintenance goals and unify the reasoning across the two types of goals.

Lastly, we review related reasoning questions. First, intention selection has been extensively studied as an important topic for intelligent agents (compare Yao et al., 2016a). Notably, van der Hoek et al. (2007) develop a logical analysis of BDI intention

revision, which can be linked with a notion of partial goal satisfaction. The focus of these authors is more theoretical than computational, in that an analysis is provided but no implementation methods.

Second, as van Riemsdijk and Yorke-Smith (2010) point out, there is a body of work on reasoning with partial *plans*, for instance in plan formation or negotiation (e.g., Lesser et al., 2004; Grosz and Hunsberger, 2006; Kamar et al., 2009), as well as in the automated planning literature (e.g., Smith, 2004). For example, in the area of multiagent planning and negotiation, Kamar et al. (2009) investigate helpful assistance of teammates in pursuit of a plan that could be partially complete, and Kamali et al. (2007) investigate information timing.

In the context of hierarchical task network (HTN) planning, Clement et al. (2007), based on their own earlier work, develop summarisations of propositional and metric resource conditions and effects [of which Thangarajah and Padgham (2011) can be seen as a special case] of a partial temporal HTN plan, and, like Thangarajah et al. (2003), use these to determine potential and definite interactions between abstract tasks. Their work admits resource bound information and emphasises facilitating the HTN planning process. Although accommodating interleaved local planning, multiagent coordination, and concurrent execution, their work is not in the context of BDI-style agents and does not target measures of goal completeness.

Third, resources and agents' executablity of tasks are formalised in so-called *resource logics*, which allow expression of properties such as "a coalition of agents A has a strategy (a choice of actions) requiring no more than *b* resources, such that whatever the actions by the agents outside the coalition, any evolution of the system generated by the strategy satisfies some temporal property" (Alechina et al., 2017). Alechina et al. (2017) present a decidable fragment of resource agent logic. Again, however, the logic is binary with regard to outcomes.

Fourth, time can be seen as an important special case of a resource. There is a body of work on reasoning about time in agent programming languages. For instance, the soft real-time agent architecture (Horling et al., 2006), AgentSpeak (XL) (Bordini et al., 2002) and AgentSpeak (RT) (Vikhorev et al., 2011). The compatibility of our contribution and that body of work is deferred to future research.

Fifth, there is a broad literature on probabilistic and fuzzy or soft computing. One can see measures about fuzzy goals (Shen et al., 2004; Katarzyniak and Popek, 2013), as being degrees of completion of goals. Our interest is in the logical tradition of BDI agents. An interesting question for future study is how a fuzzy degree of completeness relates to our work.

3 Background

This section provides the necessary background about BDI agent systems, goals, and GPTs.

3.1 BDI agents and goal types

The BDI architecture (Georgeff and Rao, 1998) specifies that an agent has beliefs about the world (denoted $\mathcal{B}$), goals that it wishes to accomplish ('desires', but see below), and goals that it has adopted together with plans to achieve them ('intentions'). BDI agents

are characterised by having a library of (parameterised) plans. An agent chooses the most appropriate set of plans and how to execute them in order to achieve a set of goals – or more generally to maximise some objective function. The steps of a plan may contain (sub-)goals which are in turn achieved by other plans.

By design, BDI agents in general have different ways (plans) of accomplishing a particular goal, and these may use different resources and bring about different effects. Moreover, plans may fail and unexpected events may occur. The deliberation on which way to achieve the goal (i.e., plan selection) is made dynamically during execution depending on the context the agent is in, and hence is not known in advance.

For discussion on BDI architectures and their extensions, we refer to the literature (e.g., Broersen et al., 2001; Braubach et al., 2004; Myers and Yorke-Smith, 2005; Herzig et al., 2017).

It should be noted that the BDI approach assumes that the agent generally does not have *knowledge* of the world (i.e., justified true information that will not change), but rather *beliefs*, which may be fallible and subject to change. The agent is also assumed to be rational, meaning that its beliefs are based on evidence, as much as possible, and that the agent does not believe obviously untrue statements such as 1 + 1 = 5 or that gravity does not apply. This means that we assume that the information available to the agent is generally reliable but not infallible, and that the agent's beliefs are generally true.

We will assume that the agent has an appropriate method for storing and updating its beliefs (which we denote here by $\mathcal{B}$) and for inferring whether given statements follow from its beliefs or not (which will denote by $\mathcal{B} \models G$ as appropriate).

As originally formulated, the BDI architecture focused on goals of accomplishment. Subsequently, the importance of goals of maintenance was identified (Braubach et al., 2004; Duff et al., 2006; Dastani et al., 2006). We next describe these two types of goals. It will be necessary to understand the states that a maintenance goal can hold, in order to define notions of completeness for it. In order to see the contrast between achievement and maintenance goals, we also depict the states of the former.

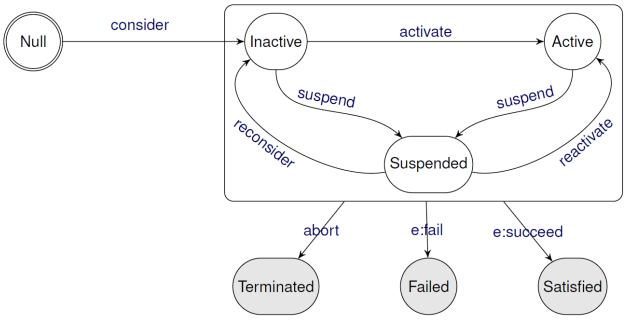
3.1.1 Achievement goals

An agent adopts an achievement goal in order to accomplish some change in the world, such as the successful completion of a science experiment. We denote an achievement goal G as achieve(k, S, F), where k is the goal's context, S is its success condition, which we take as a conjunction of effects, and F is its failure condition.

The goal's context is a pre-condition for the goal to be considered by the agent: while the context is not believed by the agent to be true, the goal is not applicable. The success condition describes the state of the world that must be true in order for the goal to be accomplished (Sardiña and Padgham, 2011). We write $S(G)$ for the success condition of a goal G. Should the agent believe that the failure condition becomes true (before it believes that the success condition becomes true), then the instance of the goal has failed (Harland et al., 2014).

Figure 1 depicts a state-based life-cycle for an achievement goal. Note the labels denote events (e:fail, e:succeed) and actions (all others). The actions are performed by the agent whereas the events are things the agent comes to believe (i.e., the agent now believes success or failure condition is true). Null means that the goal has not (yet) been created.

Figure 1 Life-cycle of an achievement goal



Source: Adapted from Harland et al. (2014, 2017)

3.1.2 Maintenance goals

Our approach to maintenance recognises the inherent fallibility of agents in maintaining appropriate conditions in their environment. A condition to be maintained is not ensured to hold continuously, but, every time it fails to hold, the agent *reactively* attempts to resurrect it. Further, the agent may act *proactively* and attempt to pre-empt failure if it can anticipate the falsehood of the maintenance condition (Duff et al., 2006). In either case, the agent is attempting to achieve a state of the world that is not currently true, which is exactly the context in which it adopts an achievement goal. Hence maintenance goals are generally associated with one or more achievement goals, which can be triggered either reactively or proactively in order to restore or maintain the maintenance condition. For example, this may involve recharging a battery when its charge level falls below a certain amount (reactive) or reducing activity in order to preserve battery charge (proactive).

The literature on maintenance goals and related concepts is extensive. Some early conceptions are the passive and active preserve goals of the PRS system (Ingrand et al., 1996); later works develop temporally extended goals (Hindriks et al., 2009), as well as both reactive and proactive goals (Darimont et al., 1997; Braubach et al., 2004; Duff et al., 2006; Dastani et al., 2006; Kaminka et al., 2007; Hindriks and van Riemsdijk, 2008; Baral et al., 2008). We refer to Duff et al. (2006) for a survey.

While we will follow Duff et al.'s (2006) concept of maintenance goal, we point out that our interest in this article is on the logical goals of BDI agents, as stated in Section 2. For instance[1], one notion of maintenance goal is a continuous goal achieved to some degree, e.g., 'maintain a distance of 1 metre from the wall'. This goal is maintained to some degree if the agent stays between [0.9, 1.1]m of the wall, and to greater and lesser degree otherwise. This kind of 'fuzzy' or 'soft computing' agents (Katarzyniak and Popek, 2013) are not in our remit.

Let M = maintain(k, m, S, F) be a maintenance goal for condition m. We specify the semantics of M to be: provided context k is true, the agent will maintain the truth of condition m until it believes that either the success condition S or failure condition F become true. In other words, 'once k becomes true, maintain m until S or F'.

In adopting the term 'success condition', we follow terminology in the literature. It should be clear that the meaning of 'success' condition differs for a maintenance goal than for an achievement goal, although in notation we write $S(\cdot)$ for either type.

Specifically, $\mathcal{B} \models S$ does not mean the goal has 'succeeded' in the sense of having accomplished an 'objective' but rather that the goal has stopped being relevant in the sense that the purpose of the goal has elapsed (Section 5.3.1).

The agent might maintain M reactively or proactively. The progress estimates computed in this article are applicable for either of these two ways of reasoning that the agent might use. In reactive reasoning, $\mathcal{B} \models \neg m$ means that the agent believes that m is false: thus, the agent adopts a recovery achievement goal R. If the agent has a look ahead mechanism π (provided by the agent designer) (Duff et al., 2006), then it can also perform proactive reasoning.[2] Let $\pi(m)$ be a predicate that returns true or false according to the (current) prediction of the future truth-hood of m. Then, $\mathcal{B} \models m \wedge \pi(\neg m)$ means that the agent believes that m will become false unless it acts appropriately: thus, the agent adopts a preventive achievement goal P. One way to envisage this is that the look ahead mechanism is a process of checking the agent's current plans for violations of any maintenance goals. However, the precise details of the look ahead mechanism does and how it works, and its limitations, are not relevant to the focus of this article. For our purpose, we only need that the agent has some such mechanism. In general, the look ahead will not be fully reliable and will have limited time horizon. For a discussion we refer to van Riemsdijk and Yorke-Smith (2012), Duff et al. (2014) and Barkan and Kaminka (2019).

Duff et al. (2006, 2014) include the reactive repair goal R and proactive preventative goal P in the syntax for M, i.e., they specify in the definition of a maintenance goal the repair and preventative goals. Following Duff et al. (2014), we do not prescribe how the agent determines relevant plans for its goals; this may be done by the use a traditional plan library, or by real-time planning, or a combination both (de Silva et al., 2009), and so these plans are not included in the definition of the maintenance goal. Harland et al. (2014) clarify the semantics of Duff et al. (2014) by insisting that both R and P be achievement goals. We follow this semantics, which is consistent with that of Duff et al. (2014).

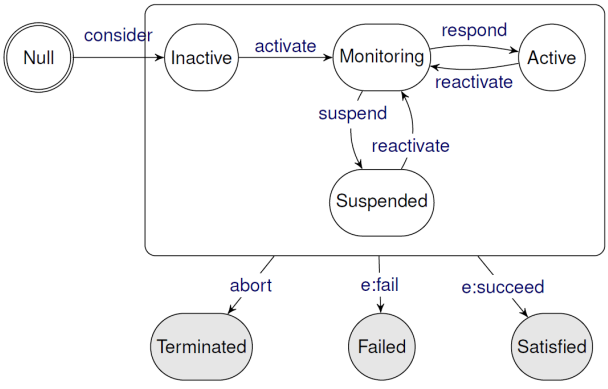
In more detail, the proactive repair and maintenance goals are respectively R = achieve($\neg m$, m, F_R) and P = achieve($m \wedge \pi(\neg m)$, $\pi(m)$, F_P), where F_R and F_P are the failure conditions for R and P respectively, specified by the agent designer.

Figure 2 depicts a state-based life-cycle for a maintenance goal M (Duff et al., 2006; Harland et al., 2014). It is important to specify the goal's semantics precisely, because this affects the notions of completeness in Section 5. In the figure, the event e:fail means that the agent believes that F has become true; the event e:success means that the agent believes that S has become true.

The state Monitoring distinguishes maintenance from achievement goals. Once the agent considers the goal, M starts in the Inactive state; upon its activation, M transitions to the Monitoring state. This difference means that we can allow for processes such as maintaining a lower maximum speed in windy conditions, rather than requiring that all such goals immediately enter the Monitoring state. Here, the agent monitors the (predicted) truth status of condition m. Should the agent believe that m is not true or will not remain true in the future, it transitions the maintenance goal M to Active, where the agent creates and adopts a recovery goal R or a preventive goal P, as explained above. M remains Active until m is restored or is no longer predicted to become false, whereupon M returns to Monitoring. Should P or R fail, the agent retries them, as we assume that success of either of these goals is sufficient to restore m, and that both P and R will

typically succeed (we do not discuss other potentially useful behaviours in this situation.) If the agent believes that it *cannot* prevent $\neg m$ or restore m, it has the option to fail M. There are at least two ways of engineering this: the failure condition F could be specified incorporate this requirement, or the failure condition could include the failure of P or R, i.e., the agent tries once to prevent $\neg m$ or restore m, and fails M if it cannot. Should the agent choose to transition M to the Suspended state, this means that the agent no longer monitors m and it aborts any active achievement goals P or R generated by M (Harland et al., 2014). The simplest approach regarding suspension and reactivation is for the agent to recompute its progress estimates for M when the goal is reactivated.

Figure 2 Life-cycle of a maintenance goal



Source: Adapted from Duff et al. (2006)

3.2 Goals and the GPT

A BDI agent will generally have several goals that it is currently pursuing, which may include both achievement goals and maintenance goals. The latter will typically be in the Monitoring state, i.e., not active, but with the relevant maintenance conditions being monitored, so that appropriate action may be taken if any maintenance condition is violated or predicted to be violated.

Traditionally, a BDI agent program consists of a set of pre-defined plans that are used to achieve or maintain the agent's goals. Each plan consists of steps which are either basic actions or sub-goals. Each sub-goal is in turn achieved by some other plan. This relationship is naturally represented as a tree structure called a *GPT* of the kind seen in Figure 4. Note that even if plans are generated as needed, rather than selected from a pre-existing library, there will still be a GPT relating goals to the plans used to achieve them.

Formally (Yao et al., 2016a; Logan et al., 2017), the root of a GPT is a top-level goal (goal-node), and its children are the plans that can be used to achieve the goal (plan-nodes). Usually there are several alternative plans to achieve a goal: hence, the child plan-nodes are viewed as 'OR' nodes. By contrast, plan execution involves performing all the steps in the plan: hence, the children of a plan-node are viewed as 'AND' nodes. As in Yao et al. (2016c) and Yao and Logan (2016), we consider GPTs in which plans may contain primitive actions in addition to sub-goals. However, we will

take a plan as our basic unit of reasoning, following Thangarajah and Padgham (2011). Figure 3 gives the formal syntax of GPTs.

Figure 3 BNF Syntax of GPTs with actions

$$
\begin{array}{rcl}
\langle \text{GoalType} \rangle & ::= & \langle \text{GoalTypeName} \rangle\ \langle \text{Precondition} \rangle \\
 & & \langle \text{In-condition} \rangle\ \langle \text{Postcondition} \rangle \\
 & & \langle \text{Plans} \rangle \\
\langle \text{GoalTypeName} \rangle & ::= & \langle \text{Label} \rangle \\
\langle \text{Plans} \rangle & ::= & \langle \text{PlanTypeName} \rangle\ (,\ \langle \text{PlanTypeName} \rangle)^* \\
 & & \\
\langle \text{PlanType} \rangle & ::= & \langle \text{PlanTypeName} \rangle\ \langle \text{Precondition} \rangle \\
 & & \langle \text{In-condition} \rangle\ \langle \text{Postcondition} \rangle \\
 & & \langle \text{PlanBody} \rangle \\
\langle \text{PlanTypeName} \rangle & ::= & \langle \text{Label} \rangle \\
\langle \text{PlanBody} \rangle & ::= & \langle \text{ExecutionStep} \rangle\ (;\ \langle \text{ExecutionStep} \rangle)^* \\
\langle \text{ExecutionStep} \rangle & ::= & \langle \text{ActionTypeName} \rangle \mid \langle \text{GoalTypeName} \rangle \\
 & & \mid (\langle \text{ExecutionStep} \rangle \parallel \langle \text{ExecutionStep} \rangle) \\
 & & \\
\langle \text{ActionType} \rangle & ::= & \langle \text{ActionTypeName} \rangle\ \langle \text{Precondition} \rangle \\
 & & \langle \text{In-condition} \rangle\ \langle \text{Postcondition} \rangle \\
\langle \text{ActionTypeName} \rangle & ::= & \langle \text{Label} \rangle \\
 & & \\
\langle \text{Precondition} \rangle & ::= & \epsilon \mid \langle \text{Condition} \rangle\ (,\ \langle \text{Condition} \rangle)^* \\
\langle \text{In-condition} \rangle & ::= & \epsilon \mid \langle \text{Condition} \rangle\ (,\ \langle \text{Condition} \rangle)^* \\
\langle \text{Postcondition} \rangle & ::= & \epsilon \mid \langle \text{Condition} \rangle\ (,\ \langle \text{Condition} \rangle)^* \\
\langle \text{Condition} \rangle & ::= & \langle \text{Statement} \rangle \mid \textit{NOT}\ \langle \text{Statement} \rangle \\
\langle \text{Statement} \rangle & ::= & \textit{string} \mid \langle \text{Variable} \rangle = \langle \text{Value} \rangle \\
\langle \text{Label} \rangle & ::= & \textit{unique string} \\
\langle \text{Variable} \rangle & ::= & \textit{unique string} \\
\langle \text{Value} \rangle & ::= & \textit{string} \\
 & & \\
\langle \text{GoalInstance} \rangle & ::= & \langle \text{InstanceName} \rangle\ \langle \text{GoalType} \rangle \\
\langle \text{PlanInstance} \rangle & ::= & \langle \text{InstanceName} \rangle\ \langle \text{PlanType} \rangle \\
\langle \text{ActionInstance} \rangle & ::= & \langle \text{InstanceName} \rangle\ \langle \text{ActionType} \rangle \\
\langle \text{InstanceName} \rangle & ::= & \langle \text{Label} \rangle
\end{array}
$$

Source: From Yao et al. (2016a)

In order to facilitate reasoning over goals, we follow the methodology of Clement et al. (2007) and particularly Thangarajah and Padgham (2011) and Thangarajah et al. (2003), we require the goals and plans be annotated with certain information about the resource requirements and effects attained, we generate a GPT structure with annotations, and use it in our reasoning algorithms.

In the next section we present a simplified version of the Mars rover scenario, and then in the first part of Section 5 we explain the annotations on goals and plans, before proceeding to describe our computational mechanisms that use them.

4 Example introduced

We illustrate our approach on a Mars rover scenario. In this section we describe a simplified version of the scenario so that we can illustrate Section 5 using it. An autonomous rover has these resources: battery charge (energy), spectroscope utilisations (drill bits), internal memory capacity (for images), and time. The spectroscope involves drilling a small sample from a target (e.g., a rock), and the rover's drill bit has a limited lifetime; hence the spectroscope is a consumable, discrete resource. Energy is a

renewable resource, since the rover can charge its battery from the sun. Memory is a renewable discrete resource. Time (seconds) is a renewable but perishable resource.

To begin with, we will ignore the resource energy and the associated maintenance goals that the rover adopts to ensure it does not exhaust its battery. We return to these in Section 6.

Each sol (Martian day), the rover leaves its base to explore a given region; today, that is region *red*1. The rover's top-level goal is: ExploreRegion(*red*1) = achieve($\top$, TargetList(*canyon*) $\wedge$ At(*canyon*) $\wedge$ Measured(*rock*1) $\wedge$ Measured(*target*), $\bot$). The rover believes that region *red*1 has a rock of interest, *rock*1, which it is currently near, and an area of interest, *canyon*, within region *red*1 but some distance away from *rock*1, which it has been instructed to survey.

The GPT is shown in Figure 4. There are various plans and goals below the top-level goal, as shown. Following the plan *traverseAndStudy* for ExploreRegion(*red*1), the rover will perform an Experiment(*rock*1) on *rock*1, Traverse(*rock*1, *canyon*) (i.e., move) from *rock*1 to *canyon*, and then Survey(canyon).

For an Experiment goal, the rover can choose from two possible plans, one using its spectroscope and the other its thermal imaging device. In both cases, the rover moves close to the target object, positions its device arm, performs the measurement and saves any data. For the Survey goal, the rover has a single plan, which is to first IdentifyTargets, which may be fulfilled by a plan that uses the panoramic camera and then selects one target, and second Experiment on the selected target object. Later we will elaborate the scenario with the more realistic goal of iterating through a list of targets for the survey. Note for better presentation we omit the repeated sub-tree rooted at Experiment(*target*) in the right-hand branch of the GPT, as it is identical to the sub-tree rooted at Experiment(*rock*) apart from the argument.

Figure 4, the GPT for the Mars rover scenario, has the effects and the resources estimated to be required for each leaf plan-node. We assume these estimations are specified by the rover's designer, e.g., based on past experiences. The remaining annotations (except the success conditions) are computed from the leaf nodes and goals as we will describe in Section 5.1. Note that each goal has a pre-specified set of success conditions annotated to it, in additional to the effects.

Lastly, in order to state an appropriate success condition for goals such as Experiment(*rock*1), we use a predicate Measured(*X*), which is true if either SpectralProfile(*X*) or ThermalProfile(*X*) is true. This is easily implemented by an appropriate rule in the agent's beliefs.

5 Quantifying completeness

We can identify a number of factors which may contribute towards assessing the completeness of a goal: resources, deadlines, number of actions/plans complete, time elapsed, effects realised, etc. In this work we propose the use of two factors to determine a quantifiable measure of completeness of a goal: *resource consumption* and, for achievement goals, the *effects* of achieving the goal.

First, we use resource consumption to provide a measure of the level of *effort* the agent has dedicated towards satisfying or repairing the goal. As discussed in Section 2, there has been previous work on representing resource requirements and continuously refining them as the agent executes its goals. We build on this existing work to provide a

quantifiable measure of completeness with respect to effort. Resource consumption cannot directly measure progress: an agent might have expended considerable resources, but made little progress towards a goal. Further while nonetheless resource consumption is a natural and useful quantity to exploit, there are complicating factors especially in realistic settings. For instance, a plan for a goal might fail, necessitating additional resources to retry it or try another plan; and goals and more generally agents can interact, such as the Mars rover needing to wait for another robot, and consuming battery power while waiting. We discuss these points further in Section 8.

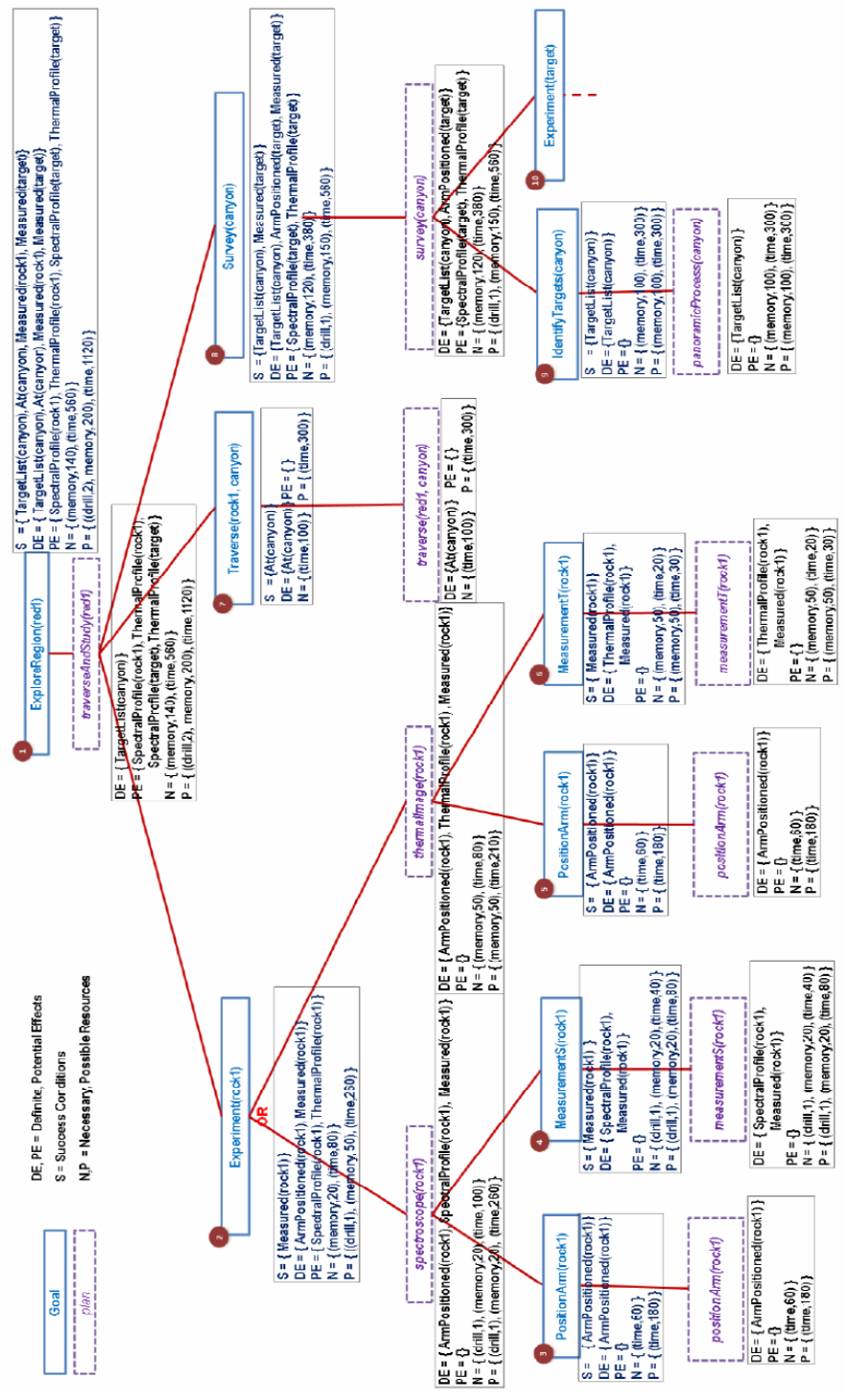
Second, the *effects* of an achievement goal capture its desired outcome, generally in terms of conditions that should be true when the goal execution is complete (Sardiña and Padgham, 2011; Thangarajah and Padgham, 2011). For example, the effect of a goal of the Mars rover to perform a measurement on a rock is that the rock is measured. We use the effects of the goal to provide a measure of the level of goal *accomplishment*, since the purpose of an achievement goal is indeed to bring about its intended effects. Goal effects more directly measure progress than resource consumption. As with resources, we build on and extend existing work on representing and reasoning about the effects of goals and plans (Thangarajah et al., 2003). In that prior work, effects are represented as Boolean predicates, such as *Measured* in the rover example (Figure 4). However, there may be instances where the conditions may be satisfied to a certain degree, such as 80% of the measurement is completed. We extend the prior work in the literature to allow for this representation.

Besides these two factors representing effort and accomplishment, we highlight two other factors that might seem amenable to be used as a measure of completeness: the number of actions performed by the agent and the time taken. To reason with the number of actions seems tempting, in that it is relatively simple to identify them. However, not all actions are the same; for example, moving the rover from one location to another is one action, as is turning on a specific light. This means that resource consumption tends to be a more useful measure, in that different actions may involve vastly different uses of resources.

On the other hand, time can be measured with respect to the pace of goal execution. However, to reason about the time required to execute a particular goal, an explicit representation of the time taken to execute each action or an entire plan is needed. For instance, Yao et al. (2016b) perform deadline-aware reasoning about intentions in a BDI system. If this is the case, then it is possible to consider time as a type of resource and use the same computational mechanisms we describe for resources, as we will illustrate. To ensure tractable computation, however, we do not consider dedicated temporal reasoning or projections (compare van Riemsdijk and Yorke-Smith, 2010; Yao et al., 2016b).

We first consider achievement goals, focussing on quantifying completeness in terms of resource consumption and the effects of achieving the goal, and providing a computational mechanism to do so. We then turn to maintenance goals, defining what completeness means and showing how parts of our computational approach are useful again for these new notions of completeness. As a preliminary step, we specify the annotations on goals and plans that provide the basis for the computation.

Figure 4 GPT in the mars rover scenario, achievement goals only (see online version for colours)



Note: Resources and effects are shown annotated on nodes.

5.1 GPT annotations

5.1.1 Resources (Morley et al., 2006)

The resources consumed when a goal is executed or maintained by an agent depend on the plans that are used to achieve or maintain the goal. As such, resource requirements are not annotated on goals, but only at the plan level. Each plan will have ascribed the resources necessary for the plan to complete execution, in line with the precedent in Clement et al. (2007), Thangarajah and Padgham (2011) and Morley et al. (2006). Note that this means that the GPT must be ground, i.e., all details known for it, at the time of execution. These resource annotations do not include the resources required for executing the sub-goals of the plan, if any. This declarative specification can be made by the agent designer, or in some domains learned from past execution traces.

Definition 1: A set of resources $\mathcal{R}$ is a set of key-value pairs $\{(r_1, \alpha_1), \ldots, (r_n, \alpha_n)\}$ where r_i is the unique *resource name* and $\alpha_i \in \mathbb{N}$ is its corresponding value. □

Definition 2 (Thangarajah and Padgham, 2011): The *resource requirement* of a plan p, denoted $p^{\mathcal{R}}$ is the set of total resource requirements of all the actions within p, in order to successfully execute p (assuming that all actions succeed). □

For example, in Figure 4, plan *measurementT* has resource requirement {(memory, 50), (time, 20)}.

$p^{\mathcal{R}}$ contains all the resource names and, for each, the aggregate total value. For example, if p had one action which used 50 units of memory and another action which used a further 25 units of memory, then in $p^{\mathcal{R}}$ is (memory, 75). Note that $p^{\mathcal{R}}$ does not contain the resource requirements of the sub-goals within the plan, but only of its actions.

The above definition of resources takes the values of resources to be discrete. Our methodology applies if $\alpha_i \in \mathbb{R}$. In fact, we can accommodate lower and upper bound range estimates, thanks to the computation of summary information described below (compare Morley et al., 2006; Clement et al., 2007). Hence, with this extension, resource annotations on plans can be single values (when the resource usage can be estimated precisely by the designer) or ranges (when they cannot be estimated precisely).

We consider resources of two types: *consumable* and *reusable* (or renewable). The former are those that are no longer available following use (e.g., drill bits) and the latter are those that can be reused following usage (e.g., memory[3]).

We will need a means to aggregate resource values. We use straightforward addition, since the domains of all resources are commensurate. Suitable aggregation operators can be defined for ranges (Clement et al., 2007).

Definition 3 (Thangarajah and Padgham, 2011): The *resource set aggregation operator* ($\oplus$) is defined as: $\mathcal{R}_1 \oplus \mathcal{R}_2 = \{(r, \mathcal{R}_1(r) + \mathcal{R}_2(r)) \mid r \in (dom(\mathcal{R}_1) \cup dom(\mathcal{R}_2))\}$ where r is a resource type and $\mathcal{R}_1(r)$ provides the value of r in the relational set $\mathcal{R}_1 \subseteq \mathcal{R}$. □

Note that we assume $\mathcal{R}_i(r) = 0$ if $r \notin \mathcal{R}_i$. So for example {(*drill*, 1), (*memory*, 20)} $\oplus$ {(*memory*, 40), (*time*, 50)} = {(*drill*, 1), (*memory*, 60), (*time*, 50)}.

While straightforward, our definition can suffer from interaction effects (when, e.g., two actions can be achieved with a single resource) and weighting of different resource types' values. Below we introduce weights for the resources.

5.1.2 Effects

Recall from Section 3.1.1 that we define the success condition of an achievement goal to be a set of effects, $S(G)$, where the conjunction of the effects must hold for the goal to be complete. A plan p will have attached to it the effects attained by the actions of that plan (Thangarajah et al., 2003) excluding the effects of any sub-goals that are executed by other plans.

Previous work about effects reasoning defined the effects of a goal (or plan) as simple predicates that are either true or false. As we have seen, this neglects effects which are not discrete but fulfilled continuously to a certain degree. For example, a goal like MeasurementS in Figure 4 involves performing a spectral profile on a target. A variant of this goal with quantified effects might be considered 80% complete once it has analysed 80% of the target's sample, which would result in an effect of (*Measured*, 80).

Let ε be the set of all effect-types relevant to the agent system.

Definition 4: An *effect* is a key-value pair (e, α) where $e \in \varepsilon$, the effect-type, is a unique identifying label and $\alpha \in \mathbb{R}$ is the degree to which the effect has to be attained for it to be achieved. For discrete effects, $\alpha \in \{0, 100\}$; for continuous effects, $0 \leq \alpha \leq 100$. □

For example, consider the effect-quantified variant of the goal MeasurementS which drills a hole and then performs a spectral profile on the drilled sample. At 80% completion of the analysis, its effects can be represented as: {(*Hole-Drilled*, 100), (*Measured*, 80)}.

As with resources, we will need a means to aggregate effect values:

Definition 5 (Thangarajah and Padgham, 2011): The *effects set aggregation operator* ($\oplus$) is defined as: $E_1 \oplus E_2 = \{(e, E_1(e) + E_2(e)) \mid e \in (dom(E_1) \cup dom(E_2))\}$.

5.2 Quantifying completeness of achievement goals

In this section we introduce two measures of completeness for achievement goals in BDI agents, based respectively on resources and effects. Our computational mechanisms reason over dynamically-updated summary information on the GPT, derived from the resource and effect annotations described above.

5.2.1 Summary information

In order to determine the level of completion of an achievement goal G at the current time t, with respect to resources or effects, it is necessary to determine:

1 the resources consumed and effects attained thus far in executing G

2 the resources required and effect that should be attained in order for the goal to complete from t.

The former step can be computed accurately, by monitoring the resource consumption and checking the current state of the world for effects achieved. There might be a cost of this checking, such as time or some resource consumption.

The latter step, of determining resources and effects needed to complete the goal, is more complex. Recall from Section 3.1 that the nature of BDI agents means that there can be multiple possible plans for accomplishing a particular goal, using different

resources and bringing about different effects, that plans may fail and unexpected events may occur, and that plan selection is dynamic during execution (although, as above, for a rational agent, we do not expect failure to be generally more common than success). Consequently we cannot always say a priori precisely what resources will be needed to accomplish a given goal.

Further, although no matter which way a goal is pursued, its effects ought to be attained, the way in which the goal is achieved may result in further effects. Some of these (side-)effects may be necessary no matter which way the goal is achieved; others not. For example, the goal Experiment(*rock*1) results in the effect *ArmPositioned*(*rock*1) no matter which plan is followed, while the effects *SpectralProfile*(*rock*1) and *ThermalProfile*(*rock*1) depend on the choice of plan used.

The second step above therefore requires some form of look-ahead for both resources and effects. It suffices for us to adopt and extend the efficient look-ahead mechanism of Thangarajah and Padgham (2011) and Thangarajah et al. (2003) which uses summary information to compute a lower and upper-bound of future resource usage and effects attained.

5.2.1.1 Resource summaries

Previous work (Thangarajah and Padgham, 2011) used the notion of summary information to (dynamically) estimate the *necessary* (lower-bound) and *possible* (upper-bound) resource requirements of a goal. Necessary resources are those that are used no matter which way the agent chooses to achieve the goal, while possible resources are those that may be needed in the worst case. This worst case is the largest resource consumption of the known plans, assuming that all the current plans succeed. This may seem simplistic; however, the alternative is to consider some kind of failure modelling, which is both not a feature of previous work and beyond the scope of the present paper. Furthermore, if a failure results in a change to an agent's plans, the resource estimates can be easily recalculated once the revised plans are known.

We adopt the algorithms for computing and updating resource summaries introduced in Thangarajah and Padgham (2011) and Morley et al. (2006) for our purpose. We do not detail all the same algorithms here since it is not the contribution of this work and is not necessary to understand the approach we present, but refer to the cited references for the details. The most relevant algorithm is updating GPT summaries dynamically, given in pseudocode as Algorithm 1.

Definition 6: The dynamically-updated *resource summary* of a goal G at time t is:

$$RS^t(G) = \left\langle N_R^t(G), P_R^t(G) \right\rangle \tag{1}$$

where $N_R^t(G)$ is the set of necessary resources and $P_R^t(G)$ the set of possible resources required to execute the goal from current time t. □

For example, the goal Survey(*canyon*) in Figure 4 has necessary resources $N_R^t(\text{Survey}(canyon)) = \{(\text{memory}, 20), (\text{time}, 380)\}$, and possible resources $P_R^t(\text{Survey}(canyon))$ $= \{(\text{drill}, 20), (\text{memory}, 380), (\text{time}, 580)\}$.

Note that both parts of the resource summary can change: the resource summary is dynamically-updated as execution proceeds, and the actual use of resources becomes known (rather than estimated). Note also that the necessary resources are the minimum

required, whereas the possible resources are the maximum required. This means that $N_R^t \leq P_R^t$.

In Algorithm 1, for any node of the GPT, summary(*r*) for resource $r \in \mathcal{R}$ returns the pair (necessary, possible) of resource *r*, indexed by a count of the number of child nodes using the resource.

We briefly explain the algorithm. Procedure `delete` simply deletes a node from the GPT. Procedure `update` updates the resource summary of a node: both the necessary and possible resources. The next three procedures use the concept of a node being 'dirty', i.e., its resource summary is in need of an update check, or not. Procedure `propagateDirty` pushes the dirty flag from parent node to its children. Procedure `recompute` performs the actual computation of how the resources at a node have changed, clearing the dirty flag when done. Procedure `decrementCount` records that a unit of a resource type is finished, and initiates upward propagation of this information and of dirty status as appropriate. For further details, see (Thangarajah and Padgham, 2011); a more advanced algorithm is given by Morley et al. (2006).

5.2.1.2 Effect summaries (Thangarajah and Padgham, 2011)

The effect summaries of a goal are defined in terms of *definite* and *potential* effects: definite effects are those that are brought about no matter which way the goal is achieved, while potential effects are those that may potentially be brought about depending on the way the goal is achieved.

Thangarajah et al. (2003) similar to their work on resource summaries, presented a set of algorithms for deriving effect summaries at compile time and updating them dynamically at run-time. The form of the algorithm is similar to Algorithm 1.

Definition 7: The dynamically-updated *effects summary* of an achievement goal *G* is:

$$ES^t(G) = \left\langle D_E^t(G), P_E^t(G) \right\rangle \tag{2}$$

where $D_E^t(G)$ is the set of definite effects and $P_E^t(G)$ is the set of potential effects that will be brought about by pursuing the goal *G* at the current time *t*. □

For example, the goal Survey(*canyon*) has definite effects $D_E^t(\text{Survey}(canyon))$ $= \{\text{TargetList(canyon)}, \text{ArmPositioned(target)}, \text{Measured(target)}\}$, and potential effects $P_E^t(\text{Survey}(canyon)) = \{\text{SpectralProfile(target)}, \text{ThermalProfile(target)}\}$. Note that $D_E^t(G)$ and $P_E^t(G)$ are exclusive, and that the success condition of the goal is a subset of the definite effects, i.e., $S(G) \subseteq D_E^t(G)$.

5.2.1.3 Resource and effect weights

Our initial work (Thangarajah et al., 2014a) treated the different resources, e.g., battery charge, drill bits, memory, etc. to be of equal importance, and likewise the different effect-types, placing the emphasis on domain independence. In this article we allow a domain-dependent weight $\lambda_r \in \mathbb{R}^+$ for each resource *r* and a weight $\mu_e \in \mathbb{R}^+$ for each effect-type *e*, with default values 1. In this way we can model both the domain-independent uniform resources and effects, and also allow customised domain-specific weights.

The use of weights provides the agent designer with the flexibility to emphasise some resources more than others in the calculation of completeness. The precise meaning of these weights is therefore outside the scope of this paper; our task here is to ensure that the calculations respect these weights.

5.2.2 Resources as a measure of completeness

The aim of our resource analysis is to provide an agent with a quantified measure of effort with respect to the amount of resources consumed thus far in executing a goal, in the context of the total resource requirements for achieving the goal. Hence we require the agent to keep track of the total resources consumed in executing each goal. Note that this is the resources for a single goal G: other goals might be using the same resources; their usage is not ascribed to G.

Definition 8: Let $R^t(G)$ be the set of resources consumed thus far up to current time t solely by the execution of G. □

We write $(R^t(G))\ (r)$ for the value of resource r in $R^t(G)$ at time t, i.e., the value α_r (see Definition 1).

Note that $R^t(G)$ is monotonically increasing with t. It is natural to expect that a corresponding decreased in $N^t(G)$ over time, in that as G gets closer to completion, then the resources required to complete it should decrease. While this is generally true, it may be that $N^t(G)$ does in fact increase; ultimately this is an estimate, and hence may be incorrect.

5.2.2.1 Lower-bound resource consumption analysis

We use the necessary and possible resource summaries to provide a lower and upper-bound resource consumption analysis, respectively.

The intuition of the lower-bound resource consumption analysis is: for every resource that has been used by the current time t or is *necessary* in the future, calculate the percentage of the value of that resource that has been consumed at time t. Aggregate the percentage values to attain a single normalised value. Note that resources used so far for goal G are added to resources still needed to achieve G.

Definition 9: The *lower-bound resource consumption analysis* of a goal G at the current time t is:

$$CR^t_{lb}(G) = \frac{\sum_{r \in dom\left(R^t(G) \cup N^t_R(G)\right)} \lambda_r \dfrac{\left(R^t(G)\right)(r)}{\left(R^t(G) \oplus N^t_R(G)\right)(r)}}{\sum_{r \in dom\left(R^t(G) \cup N^t_R(G)\right)} \lambda_r} \tag{3}$$

where *dom* denotes the domain of the resource types set, i.e., the set of key values (Definition 1), and $\oplus$ is the resource set aggregation operator (Definition 3). □

Note that the plausible intuition that CR^t_{lb} is non-decreasing does not hold in general, as will be seen in execution traces in the Mars rover scenario. The intuitive reason is that the possible resources for one choice of plan may be less than the necessary resources for another choice; see Table 1 for an example of this.

Algorithm 1 Dynamic update of GPT resource summaries

Procedure `delete`(*node*):
  update(node, $\langle \emptyset, \emptyset \rangle$)
  if node ≠ root **then** // remove the node from the parent's child list
  node.parent.children := node.parent.children \ node
Procedure `update`(*node*, *newSummary*):
  if newSummary = node.summary **then** // no changes needed
  return
  foreach type ∈ $\mathcal{R}$ **do**
  (n_{old}, p_{old})$_{count}$:= node.summary(type)
  (n_{new}, p_{new}) := newSummary(type)
  if ($n_{old} \neq n_{new} \vee p_{old} \neq p_{new}$) ∧ node ≠ root **then**
  if $n_{new} = p_{new} = 0$ **then** // resource no longer needed
  `decrementCount`(*node.parent*, *type*)
  else if node ≠ root **then**
  node.parent.dirty[type] := ⊤
  `propagateDirty`(*node.parent*, *type*)
  node.summary := newSummary
Procedure `propagateDirty`(*node*, *type*):
  if node ≠ root ∧ !node.parent.dirty[type] **then**
  node.parent.dirty[type] := ⊤
  `propagateDirty`(*node.parent*, *type*)
Procedure `recompute`(*node*):
  if ¬∃*type* ∈ $\mathcal{R}$ such that node.dirty[type] = ⊤ **then**
  return // no dirty nodes
  foreach *c* ∈ node.children **do**
  `recompute`(*c*) // if there are child nodes recompute them first
  if node is a GoalNode **then**
  node.summary := $\biguplus_{c \in \text{node.children}}$ *c.summary*
  if node is a PlanNode *p* **then**
  node.summary := node.plan.p$^{\mathcal{R}}$ ⊕ node.child.summary
  if node is a ParallelOperator node **then**
  node.summary := $\oplus_{c \oplus \text{node.children}}$ *c.summary*
  if node is a SequenceOperator node **then**
  node.summary := $\otimes_{c \in \text{node.children}}$ *c.summary*
  foreach t such that node.dirty[type] = ⊤ **do**
  node.dirty[type] := ⊥ // updated hence not dirty any longer

```
Procedure decrementCount(node, type):
    (n, p)count := node.summary(type)
    node.summary(type) := (n, p)count–1
    if count – 1 = 0 then                          // resource no longer in use
    node.summary(type) := (0, 0)0
    if node ≠ root then
    decrementCount(node.parent, type)
    else
    node.dirty[type] := ⊤
    if node ≠ root then
    propagateDirty(node.parent, type)
```

Source: Thangarajah and Padgham (2011)

5.2.2.2 Upper-bound resource consumption analysis

The computation is the same as for the lower bound, except instead of the necessary resource summary we use the *possible* resource summary.

Definition 10: The *upper-bound resource consumption analysis* of a goal G at the current time t is:

$$CR^t_{ub}(G) = \frac{\sum_{r \in dom\left(R^t(G) \cup P^t_R(G)\right)} \lambda_r \frac{\left(R^t(G)\right)(r)}{\left(R^t(G) \oplus P^t_R(G)\right)(r)}}{\sum_{r \in dom\left(R^t(G) \cup N^t_R(G)\right)} \lambda_r} \tag{4}$$

Note that as the necessary resources $(N^t_R(G))$ will never exceed the possible resources $(P^t_R(G))$, it will always be the case that $CR^t_{lb}(G)$ will be at least as large as $CR^t_{ub}(G)$. This may seem counter-intuitive; it is important to keep in mind that this is an estimate of the completion of the goal, and hence if it completes using only the necessary resources, then the latter value may be less than 100%, as it may turn out that not all of the possible resources are required. For example, if $R^t(G)$ is 80, $N^t_R(G)$ is 20 and $P^t_R(G)$ is 70, then $CR^t_{lb}(G) = 80/100$ and $CR^t_{ub}(G) = 80/150$.

For example, consider the goal ExploreRegion(red1) at the point where Experiment(rock1) has completed but neither Traverse(rock1, canyon) nor Survey(canyon) has started. Let R^t(Experiment(*rock*1)) = {(drill,1), (memory,40), (time,150)}.

Then we have the following:

- N^t_R(ExploreRegion(*red*1)) = {(drill, 0), (memory, 120), (time, 480)}
- P^t_R(ExploreRegion(*red*1)) = {(drill, 1), (memory, 150), (time, 860)}

and further:

- $CR^t_{lb} = (1/(1+0) + 40/(40+120) + 150/(150+480))/3 = 49.6\%$
- $CR^t_{ub} = (1/(1+1) + 40/(40+150) + 150/(150+860))/3 = 28.6\%.$

A point to note is that there are some occasions when we may have $R^t(G)(r) = N^t_D(G)(r) = P^t_D(G)(r) = 0$ for some resource *r* and goal *G*. In such cases we will not consider *r* in the calculation of CR^t_{lb} and CR^t_{ub}. So if we had R^t(Experiment(*rock*1)) = {(drill,0), (memory,40), (time,150)} and P^t_R(ExploreRegion(*red*1)) = {(drill, 0), (memory, 150), (time, 860)} in the above example, we would calculate as below. This is because the drill resource cannot in any way contribute to the completeness of the goal ExploreRegion(red1), as there is never any change in the usage of this resource.

- $CR^t_{lb} = (40/(40+120) + 150/(150+480))/2 = 24.4\%$
- $CR^t_{ub} = (40/(40+150) + 150/(150+860))/2 = 18.0\%.$

Note that in the case where $R^t(G)(r) = N^t_D(G)(r) = 0$ but $P^t_D(G)(r) > 0$ we will ignore any term involving the first two quantities in the calculation of CR^t_{lb}, but otherwise proceed as above. This is because when there are no resources that have been consumed, there is no relevant measure of completeness that we can compute. Accordingly, if we have R^t(Experiment(*rock*1)) = {(drill, 0), (memory, 40), (time, 150)} then we calculate CR^t_{lb} as (40 / (40 + 120) + 150 / (150 + 480)) / 3 = 16.3%.

5.2.3 Effects as a measure of completeness

We now turn from resources, a measure of effort, to effects, a measure of accomplishment. As we have discussed, the effects of a goal can be thought of as the state of the world that the agent wants to achieve in order to accomplish the goal. For instance, in the example at the end of Section 5.1.2, the rover's goal to survey the area may have the effects of *area-surveyed* and *target-selected*. The percentage of these effects currently achieved gives a quantifiable measure of accomplishment. The issue at hand is not how to express effects (e.g., the language used for *S*(*G*)) but how to quantify goal completeness. For simplicity, we assume that all effects are expressed as atomic formulae. We propose two computational approaches: the first based on the success condition of the goal and the second on the effect summaries of the goal.

5.2.3.1 Completeness based on the success condition

One way of determining the level of completeness of an achievement goal *G*, with respect to accomplishment, is to determine the percentage of effects in the success condition *S*(*G*) achieved at the current point in time.

In order to compute this measure the agent needs to know the current value of a given effect, and to know the initial values of the effects in the success condition of the goal.

Definition 11: Let $B^t(e)$ be a function that evaluates the current value α of the effect $e \in \varepsilon$ as known by the agent at the current time *t*. □

Unlike the success condition or effect summaries, where the value of the effect is *what needs to be accomplished* in the future, the value of the effect determined by $B^t(e)$ is the *current* value of the effect e as estimated by the agent.

Definition 12: The initial set of effects for a goal G is $B^i(G) = \{(e, \alpha^i) | e \in \varepsilon\}$, where α^i is the value of e when the execution of G begins. □

We compute the level of completeness with respect to $S(G)$ by calculating the percentage of the value of each effect in $S(G)$ currently achieved by the agent relative to the initial value when the goal execution began and the value to be achieved. For an effect $e \in dom(S(G))$, we write $S(G)(e)$ to denote the value of e in $S(G)$, and similarly for $B^i(G)$.

Definition 13: The level of completion of a goal G at the current time t with respect to the effects in the success condition is:

$$CE_S^t(G) = \frac{\sum_{e \in dom(S(G))} \mu_e \frac{B^t(e) - B^i(G)(e)}{S(G)(e) - B^i(G)(e)}}{\sum_{e \in dom(S(G))} \mu_e} \tag{5}$$

For example, the goal Survey(*canyon*) in Figure 4 has S(Survey(*canyon*)) = {TargetList(canyon), Measured(target)}. If IdentifyTargets(*canyon*) has completed but Experiment(*target*) has not commenced, we have:

- $CE_S^t(\text{Survey}(canyon)) = ((1-0)/(1-0) + (0-0)/(1-0))/2 = 50\%.$

Note that if desired, the agent designer could use weights in order to emphasise some goals more than others. For example, if the completion of the goal IdentifyTargets(*canyon*) is more important than the goal Experiment(*target*), due to identified targets being able to be investigated later, the designer may wish for the completion of Experiment(*target*) to weigh more heavily on the completion of Survey(*canyon*) than Experiment(*target*).

A limitation of this approach is if $S(G)$ is can change. For example, in one formulation of the goals of the example, IdentifyTargets returns a target list of a priori unknown length, and Experiment must be run on each item of the list. In this sense, the progress based on the success condition depends on the interaction between the two goals. We return to this point later. Note that the above definition is nonetheless well defined, because it is defined in terms of the initial set of effects and their current values.

5.2.3.2 Completeness based on the effect summaries

The above computation $CE_S^t(G)$ does not take into consideration effects other than those in the success condition of the goal G, even for those goals where some (side-)effects are necessary in order to achieve the goal's effects. We include these effects as part of the quantification of completeness and use effect summaries to present a lower-bound using the definite effect summary, and an upper-bound using the combined definite and potential effect summaries (since they are exclusive).

Note that goal side-effects were also included in the resource summary approach of Section 5.2.2. For effects, these are not relevant, as effects are either achieved or not. Hence there is no consideration of side-effects in Definition 13.

We adopt the techniques developed by Thangarajah et al. (2003) for deriving and updating the effect summaries, but generalise their formulae to operate on a set of effects that are composed of key-value pairs and not simple predicates. This generalisation changes the way in which the sets of effects are added (⊕) and merged (⊕). We gave the redefined ⊕ operator as Definition 5 but omit the ⊕ operator as this work does not use it.

Definition 14: The *lower-bound effect accomplishment analysis* of a goal G at the current time t is:

$$CE^t_{lb}(G) = \frac{\sum_{e \in dom\left(D^t_E(G)\right)} \mu_e \frac{B^t(e) - B^i(G)(e)}{D^t_E(G)(e) - B^i(G)(e)}}{\sum_{e \in dom} \left(D^t_E(G)\right) \mu_e} \tag{6}$$

Definition 15: The *upper-bound effect accomplishment analysis* of goal G at the current time t is:

$$CE^t_{ub}(G) = \frac{\sum_{e \in dom\left(D^t_E(G) \oplus P^t_E(G)\right)} \mu_e \frac{B^t(e) - B^i(G)(e)}{\left(D^t_E(G) \oplus P^t_E(G)\right)(e) - B^i(G)(e)}}{\sum_{e \in dom} \left(D^t_E(G) \oplus P^t_E(G)\right) \mu_e} \tag{7}$$

where ⊕ is the effects set aggregation operator. □

For example, consider the goal Survey(*canyon*) in Figure 4 which has D^t_E (Survey(*canyon*)) = {TargetList(canyon), ArmPositioned(target), Measured(target)}, and P^t_E(Survey(*canyon*)) = {SpectralProfile(target), ThermalProfile(target)}. If the sub-goal IdentifyTargets(*canyon*) has completed but Experiment(*target*) has not commenced, and none of the effects were true at the start of the goal execution, then we have:

- $CE^t_{lb}(\text{Survey}(canyon)) = ((1-0)/(1-0) + 2\times(0-0)/(1-0))/3 = 33\%$
- $CE^t_{ub}(\text{Survey}(canyon)) = ((1-0)/(1-0) + 4\times(0-0)/(1-0))/5 = 20\%.$

Note that $CE^t_{ub}(G)$ will typically be less than 100%, as it will be unusual for all potential effects to be achieved. This value should thus be read in conjunction with the definite effects measure to provide an estimate of completeness, rather than as specific measurements of a precise quantity.

These two mechanisms, based respectively on resources and effects, are founded on the efficient computation of summary information for achievement goals. They provide us with, for resources, lower and upper bound analyses, and for effects, a success-based analysis and lower and upper bound effect-based analyses. Note that the success-based analysis (5) and the effect-based analysis bounds (6) and (7), are complementary: it is for the agent to decide how to use them. For example, for Survey(canyon), we have seen that the latter three analyses are respectively 50%, 33% and 20%.

We will illustrate these mechanisms on the scenario in Section 6. We next turn to maintenance goals, which present a conceptually more difficult task.

5.3 Quantifying completeness of maintenance goals

By its nature, a maintenance goal is intended to persist in order to uphold its maintain condition – until the goal is no longer relevant. In this sense, a maintenance goal is never 'complete' in the way an achievement goal can be. In this subsection we introduce two notions of 'completeness' for maintenance goals in BDI agents. The intuition is that 'completeness' is in terms of the agent's actions to maintain the goal, whether proactively or reactively.

First, *permanently complete* (PC) refers to a goal no longer being relevant, and hence the agent no longer needing to uphold the goal's maintain condition (which means no further action is ever taken with regards to this goal). By contrast, *recovery complete* (RC) refers to the agent's progress in restoring the maintain condition after it has been violated; in other words, the achievement goal used to store the maintain condition has completed successfully.

There are several reasons why it might be useful for an agent to have estimate of how close a maintenance goal M is to completing its relevancy, and, while the goal is still relevant but the maintain condition m is violated, of how close it is to completing the restoration of the maintain condition. The first reason is to make an estimate of how much longer M is still relevant, such as for how much longer a Mars rover will need to conserve its battery life. If this period is relatively short, it may be appropriate to drop this maintenance goal. The second reason is to consider the constraints on the agent's behaviour which may be required in order to maintain the maintain condition. For example, if the battery can be recharged fully in a few minutes, then this allows a greater range of behaviour than if it will take two hours, or knowing that a certain amount of memory must be kept in reserve. The third reason is to use both these estimates to inform its deliberation and planning about other goals, such as in anticipating resource conflicts (Thangarajah et al., 2003). We will see that the methods introduce above for resource-based estimation of the completeness of achievement goals can also be used to predict potential violations of resource-related maintenance goals. Finally, to use the estimates to help prioritise M in the case of goal conflict, such as because of a lack of resources.

5.3.1 Defining progress for maintenance goals

5.3.1.1 'Completion' of a maintenance goal

Consider a maintenance goal M = maintain(k, m, S, F). Granted that, by its nature, M is intended to persist in order to uphold its maintain condition, what does it mean for such a goal to be 'complete'? Recalling Figure 2, we consider the two relevant goal states in turn.

First, when the goal is in state Monitoring. In contrast to achievement goals, maintenance goals are meant to persist. In this sense, they are never complete in themselves; the agent is not attempting to achieve the success condition S.

However, if M's success condition becomes true, then the purpose of the goal has expired, and so the agent need not keep maintaining M. We call this situation *PC*. A goal that is PC is no longer relevant: it is 'complete' in the sense that the purpose of the goal is complete.

For example, consider the goal

M_1 = maintain(¬at-base > 30%, ¬priority-science-targets-exist,
¬battery-functioning).

With this goal, agent `rover` is not attempting to achieve ¬priority-science-targets-exist. However, if it is true that ¬priority-science-targets-exist, then the purpose of goal M_1 has expired, and M_1 is PC. This condition may be based on a list of currently known targets, and one can then measure progress towards this condition becoming true by tracking progress through this list. Note that it may also be appropriate to have the success condition as at-base, so that the rover will actively seek to maintain its battery charge whenever it is not at the base (not just when there are no more science targets). Progress towards the success of at-base may be measured by proximity to the base, or by the progress of a goal to make this condition true. If the failure condition becomes true, likewise the agent need not keep maintaining *M*. For example, if it is true that ¬battery-functioning, then `rover` need not keep maintaining battery > 30%. This situation is a failure case, and we do not class it as 'completeness' in any sense. Likewise, the goal's transition to state Terminated we also do not class as 'complete'. Note that if a goal fails or otherwise terminates, any quantification of its completeness becomes irrelevant.

Second, we consider when a maintenance goal is in state Active. Here, if the maintain condition *m* becomes false (or is predicted to become false), the agent acts to restore it (respectively, to prevent its violation). If it succeeds in doing so, we call this situation *RC*. RC is the analogy of completeness for achievement goals.

This notion of 'completeness' is decidedly non-monotonic: it can occur many times during execution. In contrast, once a maintenance goal is permanently complete, it has transitioned to state Satisfied and remains there.

5.3.1.2 'Progress' of a maintenance goal

The next question is: what does 'progress' mean for a maintenance goal? That is, what does it mean for *M* to be 'more complete' at time *t'* than at time $t < t''$? Again we consider the two relevant goal states in turn.

First, we consider when the goal is in state Monitoring. Again, in contrast to achievement goals, the agent is not trying to progress *M* towards the accomplishment of *S*. That said, for permanently complete, one could characterise progress by a measure of how much *S*(*M*) is complete. The less informative alternative is to just say $\mathcal{B} \models \neg S$ means that the agent believes *M* is 0% PC, and $\mathcal{B} \models S$ means that it believes *M* is 100% PC.

Second, we consider when the goal is in state Active. When *M* becomes recovery incomplete, the agent *is* then attempting to achieve either *P* or *R* in order to restore *m* or proactively prevent its violation. As such, one could characterise progress in terms of how close *M* is to being RC again by the completeness of *P* or *R*. When *m* becomes violated, then the agent believes the goal is 0% RC. As the agent progresses towards restoring *m*, it believes *M* is increasing (not necessarily monotonically) in percentage RC, and on successful completion of *P* or *R*, then *x* believes *M* is 100% RC again.

5.3.2 Computing estimates

Having indicated two notions of completeness for maintenance goals, we now specify metrics to measure progress in each case. We leverage the computational mechanisms for achievement goal progress of Section 5.2 in order to compute estimates of the metrics. This makes describing the computation quite simple, although, as for achievement goals, the underlying mechanisms are sophisticated. Again we consider the two goal states in turn.

5.3.2.1 When the goal is in state 'Monitoring'

By definition, when a maintenance goal is in state Maintain, then m is true (and not predicted to become false, in the proactive case). Hence, neither R nor P sub-goals are active, and M is (100%) RC. Given that $S(G)$ is not true, the goal is not (100%) permanently complete. For the reasons given earlier, the agent may wish to estimate how close the goal is to being PC.

If we characterise a metric of permanent completeness as how much the success condition S is complete, then we can directly leverage the computation mechanism for completeness based on the set (i.e., conjunction) of effects in the success condition (Section 5.2.3).[4] That is, the percentage PC of M at time t, denoted $PC^t(M)$, is estimated based on estimating the percentage of effects in $S(M)$ that are true at time t:

$$PC^t(M) = CE^t_S(M) \tag{8}$$

5.3.2.2 When the goal is in state 'Active'

Given that a maintenance goal is not (100%) PC, then whenever the goal is in state Active, it is not (100%) RC. Since the recovery goal R or preventative goal P are achievement goals, we can leverage the computation mechanism for completeness for R or P, in one of two ways. First, completeness according to the resource consumption in restoring m (respectively preventing its violation). Second, completeness according to the accomplishment of the effects in restoring (respectively preventing violation) m, based on R's (respectively P's) success condition or on effect summaries for the goal (i.e., R or P).

Note that the recovery/preventative sub-goals of M are restricted such that to themselves have no maintenance sub-goals (Harland et al., 2014). That is, plans to accomplish R or P can contain only actions and achieve sub-goals.

5.3.2.3 Estimate based on resource consumption of R/P

In this case we have lower and upper-bounds as follows. Let G be the achievement goal R or P as appropriate.

$$RC^t_{r-lb}(M) = CR^t_{lb}(G) \tag{9}$$

$$RC^t_{r-ub}(M) = CR^t_{ub}(G) \tag{10}$$

5.3.2.4 Estimate based on effects of R/P

We again have lower and upper-bounds, as follows.

$$RC^{t}_{e-lb}(M) = CE^{t}_{lb}(G) \tag{11}$$

$$RC^{t}_{e-ub}(M) = CE^{t}_{ub}(G) \tag{12}$$

6 Example

We now show a detailed execution of the Mars rover example from Section 4. We begin with the simplified scenario that excludes maintenance goals, iteration and the resource energy, and then study the full scenario. The purpose of this section is to demonstrate the proposed mechanisms on a complex example.

6.1 Simplified scenario: achievement goals only

Recall the scenario as described in Section 4. The rover's top-level goal is A_0 = ExploreRegion(*red*1), whose plan P_1 = *traverseAndStudy* consists of three sequential achieve sub-goals: A_1 = Experiment(*rock*1), A_2 = Traverse(*rock*1, *canyon*), and A_3 = Survey(*canyon*), as seen in Figure 4. Here our interest is in the calculations about the completeness of each goal at every point in the execution process, and what insight that may give us. For ease of illustration, we set weights λ_r and μ_e uniformly to be 1.

Note that to execute Experiment(*rock*1), the agent has to position the robot arm appropriately (PositionArt(*rock*1)), and then choose which measurement method to use (spectral or thermal). For illustration suppose the thermal method is used, which means that the agent will execute PositionArm(*rock*1) followed by MeasurementT(*rock*1). For the later goal Experiment(*target*), we will suppose the spectral method is used, and so PositionArm(*target*) and MeasurementS(*target*) are executed.

Table 1 Resource and effect summaries in the simplified scenario

No.	*Goal*	*N*	*P*	*DE*	*PE*
3, 5	PA(*r*1)	0, 0, 60	0,0,180	AP(*r*1)	-
11, 13	PA(*t*)	0, 0, 60	0,0,180	AP(*t*)	-
4	MS(*r*1)	1, 20, 40	1, 20, 80	SP(*r*1)	-
12	MS(*t*)	1, 20, 40	1, 20, 80	SP(*t*)	-
6	MT(*r*1)	0, 50, 20	0, 50, 30	TP(*r*1)	-
14	MT(*t*)	0, 50, 20	0, 50, 30	TP(*t*)	-
7	T(*r*1, *c*)	0, 0, 100	0, 0, 300	At(*c*)	-
9	IT(*c*)	0, 100, 300	0, 100, 300	TL(*c*)	-
2	E(*r*1)	0, 20, 80	1, 50, 260	AP(*r*1), M(*r*1)	SP(*r*1), TP(*r*1)
10	E(*t*)	0, 20, 80	1, 50, 260	AP(*t*), M(*t*)	SP(*t*), TP(*t*)
8	S(*c*)	0, 120, 380	1, 150, 560	TL(*c*), AP(*t*), M(*t*)	SP(*t*), TP(*t*)
1	ER(*red*1)	0, 140, 560	2, 200, 1,120	TL(*c*), At(*c*), M(*t*), M(*r*1)	SP(*r*1), TP(*r*1), SP(*t*), TP(*t*)

As an initial step we calculate the resource and effects summaries from the GPT in Figure 4. This is straightforward, and the results of this process are in Table 1. *Column no.* refers to the node numbering in Figure 4, column *N* is the necessary resources (respectively: drill count, memory, time), column *P* is the possible resources, column *DE* is the definite effects, and column *PE* is the potential effects.[5]

6.1.1 Execution

With the GPT resource and effect summaries computed, we can now proceed with execution. Table 2 shows the important stages in one execution of the goals in the scenario. For each goal, column *CE* shows the current effects which are true, including success conditions, and both definite and potential effects. The notation in this column is: *S*; *DE*; *PE*. We do not repeat effects in *DE* if they are in *S*. We now describe the trace of this execution in detail.

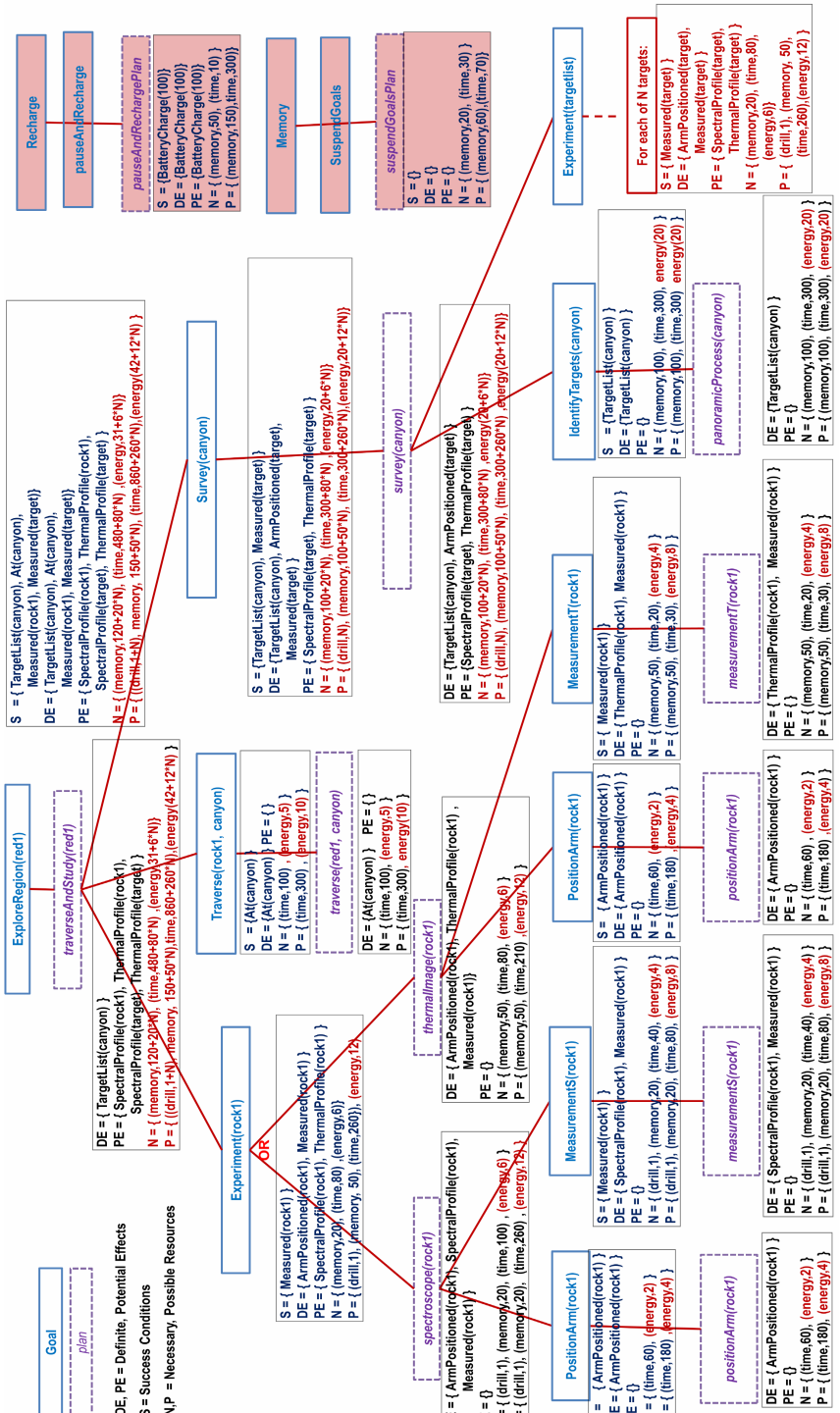
The completeness of the goalsPositionArm(_), MeasurementS(_), MeasurementT(_), Traverse(*rock*1, *canyon*) and IdentifyTargets(_) is straightforward, as for each goal a single action is sufficient for their entire execution. For the others (Explore(*red*1), Experiment(_), Survey(*canyon*)) we simply aggregate the consumed resources and effects of the completed sub-goals, and then recursively aggregate the estimates for completion of the incomplete sub-goals. This means we traverse the GPT, but we only need to consider the children of incomplete sub-goals. For example, if MeasurementS(*target*) is the only incomplete sub-goal of ExploreRegion(*red*1), then we aggregate the resources and effects used by the completed goals Experiment(*rock*1), Traverse(*rock*1, *canyon*) and Survey(*canyon*), and calculate the completion estimates for MeasurementS(*target*) in order to find the completion estimate for ExploreRegion(*red*1). Similarly if the only sub-goal of ExploreRegion(*red*1) that is complete is PositionArm(*rock*1), then the consumed resources and effects for ExploreRegion(*red*1) are just those for PositionArm(*rock*1), and the calculation of the remaining resources and effects is the aggregation of those for all the other sub-goals.

To compute the completion estimates, note that initially, all R^t values are 0, and the N_R^t and P_R^t values are as ascribed a priori on the GPT nodes. When a goal is completed, the N_R^t and P_R^t estimates are both 0, and hence the CR_{ub}^t and CR_{lb}^t measures will be 100%. The possible resources will always be equal to or greater than the definite resources, so CR_{ub}^t will never exceed CR_{lb}^t. As we discuss in Section 7.2, during the execution of a goal, R_u^t will increase, and N_R^t and P_R^t will (generally) decrease. Once a goal is complete, its R_u^t value may also be used to calculate the completeness values of subsequent goals. For example, once Experiment(*rock*1) has completed, knowledge of the resources it has used will be needed for the completeness calculations for Traverse(*rock*1, *canyon*) and Survey(*canyon*).

Table 2 Example execution in the simplified Mars rover scenario

No.	*Goals*	R^t	N_R^t	P_R^t	CR_{lb}^t	CR_{ub}^t	*CE*	CE_s^t	CE_{lb}^t	CE_{ub}^t
0	Explore(*red*1)	0, 0, 0	0, 140, 560	2, 200, 1,120	0%	0%	-	0%	0%	0%
1	Explore(*red*1)	0, 0, 0	0, 170, 560	1, 200, 1,070	0%	0%	-	0%	0%	0%
	Experiment(*rock*1)	0, 0, 0	0, 50, 80	0, 50, 210	0%	0%	-	0%	0%	0%
2	Explore(*red*1)	0, 0, 120	0, 170, 500	1, 200, 890	9.7%	4.0%	-	0%	0%	0%
	Experiment(*rock*1)	0, 0, 120	0, 50, 20	0, 50, 30	43%	40%	-; AP(r1); -	0%	50%	25%
	PositionArm(*rock*1)	0, 0, 120	0, 0, 0	0, 0, 0	100%	100%	-; AP(r1); -	100%	100%	-
3	Explore(*red*1)	0, 50, 145	0, 170, 480	1, 200, 860	26%	13%	M(r1); TP(r1); -	25%	25%	25%
	Experiment(*rock*1)	0, 50, 145	0, 0, 0	0, 0, 0	100%	100%	M(r1); AP(r1); TP(r1)	100%	100%	75%
	MeasurementT(*rock*1)	0, 50, 25	0, 0, 0	0, 0, 0	100%	100%	M(r1); TP(r1); -	100%	100%	-
4	Explore(*red*1)	0, 50, 345	0, 120, 380	1, 150, 560	38.5%	21%	M(r1), At(c); -; TP(r1)	50%	50%	37.5%
	Traverse(*rock*1, *canyon*)	0, 0, 200	0, 0, 0	0, 0, 0	100%	100%	At(c); -; -	100%	100%	-
5	Explore(*red*1)	0, 150, 645	0, 20, 80	1, 50, 260	88.6%	48.8%	M(r1), At(c), TL(c); -; TP(r1)	75%	75%	50%
	Survey(*canyon*)	0, 100, 300	0, 20, 80	1, 50, 260	81%	40%	TL(c); -; -	50%	33%	20%
	IdentifyTargets(*canyon*)	0, 100, 300	0, 0, 0	0, 0, 0	100%	100%	TL(c); -; -	100%	100%	-
6	Explore(*red*1)	0, 150, 645	1, 20, 100	1, 20, 260	58.3%	53.2%	M(r1), At(c), TL(c); -; TP(r1)	75%	75%	50%
	Survey(*canyon*)	0, 100, 300	1, 20, 100	1, 20, 260	52.8%	45.6%	TL(c); -; -	50%	33%	20%
	Experiment(*target*)	0, 0, 0	1, 20, 100	1, 20, 260	0%	0%	-	0%	0%	0%
7	Explore(*red*1)	0, 150, 765	1, 20, 40	1, 20, 80	61.1%	59.6%	M(r1), At(c), TL(c); -; TP(r1)	75%	75%	50%
	Survey(*canyon*)	0, 100, 420	1, 20, 40	1, 20, 80	58.2%	55.8%	TL(c); AP(t); -	50%	67%	40%
	Experiment(*target*)	0, 0, 120	1, 20, 40	1, 20, 80	25%	20%	-; AP(t); -	0%	50%	25%
	PositionArm(*target*)	0, 0, 120	0, 0, 0	0, 0, 0	100%	100%	AP(t); -; -	100%	100%	-
8	Explore(*red*1)	1, 170, 825	0, 0, 0	0, 0, 0	100%	100%	M(r1), At(c), TL(c), M(t); -; TP(r1), SP(t)	100%	100%	75%
	Survey(*canyon*)	1, 120, 480	0, 0, 0	0, 0, 0	100%	100%	TL(c), M(t); AP(t); SP(t)	100%	100%	80%
	Experiment(*target*)	1, 20, 180	0, 0, 0	0, 0, 0	100%	100%	M(t); AP(t); SP(t)	100%	100%	75%
	MeasurementS(*target*)	1, 20, 60	0, 0, 0	0, 0, 0	100%	100%	SP(t); M(t); -	100%	100%	-

Figure 5 GPT in the Mars rover scenario including maintenance goals (see online version for colours)



In the execution of ExploreRegion(*red*1), R_u^t is initially 0, and the N_R^t and P_R^t estimates are as in stage 0. Experiment(*rock*1) is then adopted and executed. The N_R^t and P_R^t estimates for Experiment(*rock*1) are refined in stage 1, as the rover has decided that the thermal profile will be used. After the goal PositionArm(*rock*1), the estimates for the completion of Experiment(*rock*1) are updated. Once MeasurementT(*rock*1) is completed (stage 2), Experiment(*rock*1) is also complete (stage 3). The next goal is Traverse(*rock*1, *canyon*), which completes (stage 4) using 200 units of time. Survey(*canyon*) is next, which leads to the execution of the goal IdentifyTargets(*canyon*), for which the definite and possible resource estimates are the same. The next goal is Experiment(*target*), which unlike the instance for Experiment(*rock*1), chooses the spectral option, and hence results in the execution of the goals PositionArm(*target*) (stage 7) and MeasurementS(*target*) (stage 8), which completes the execution.

6.2 Full scenario: achievement and maintenance goals

The simplified scenario neglects the resource energy, and omits the rover's maintenance of its battery charge. Until now we also assumed the *canyon* area has only a single science target of interest. Figure 5 shows the full scenario. Differences from Figure 4 are in highlighted in red text. Note the two maintenance goals depicted in the top-right of the figure.

Consider the rover's top-level maintenance goal for this sol:

M_1 = maintain(¬at-base, battery >,30% priority-targets-exist(*rock*1)
∨priority-targets-exist(*canyon*), ¬battery-functioning)

M_1 has context condition k of ¬at-base; maintain condition m of battery > 30% – the rover must keep its battery charge at least 30% of maximum, which is sufficient to get back to its base; success condition S of ¬priority-science-targets-exist – this means the goal is relevant as long as its context is true and the rover believes that high-priority science targets exist in its current target area(s); and failure condition that the battery is non-functional. In addition, the rover has goals such as R = PauseAndRecharge – the rover stops movement and science and waits until solar cells have recharged battery sufficiently – and P = ConserveEnergy – for which the rover might move more slowly, or deprioritise some science targets. For ease of illustration, we will focus on the PauseAndRecharge goal and a reactive semantics for M_1.

6.2.1 Execution

For the region of interest *red*1, the rover knows from previous surveys that the region contains *rock*1 and *canyon*, but it does not how many science targets there will be in the latter. The success condition of M_1 is therefore priority-targets-exist(*rock*1) ∨ priority-targets-exist(*canyon*). At the start of mission execution, all the effects in $S(M_1)$ are false, so M_1 is 0% permanently complete.

Part-way through execution of plan P_1 = *traverseAndStudy* for achievement goal A_0 this sol, just after positionArm(*rock*1) succeeds, the maintain condition m is violated. The rover therefore adopts the achieve goal R to repair m. M_1 is 0% RC at this point. R leads to the suspension of P_1 and so A_1 and its sub-goals, which are resumed once R succeeds and M_1 is 100% RC again.

Later, once Experiment(*rock*1) succeeds, the first part of M_1's success condition is true, and the rover computes that M_1 is now 50% permanently complete. This is clearly an estimate, since there could be more science targets in priority-targets-exist(*canyon*) than in priority-targets-exist(*rock*1), or it could take more effort to study one target in the latter than in the former. That is, the rover has only an estimate of how much progress has been made on M_1 and it must keep maintaining M_1 until $S(M_1)$ is fully true.

Given the estimation that M_1 is 50% complete, the rover notes that it already has had to recharge once. Hence, based on its estimate, the rover may decide to choose a plan for Transverse(*rock*1, *canyon*) that involves travelling at lower speed than normal, because it predicts that it has limited battery charge left before it will likely to pause and charge again.

6.2.2 Multiple targets in the canyon

We extend the latter part of the scenario as previously presented by having the rover select not one single target in the canyon (from the found target list) and perform Experiment(*target*), but, perhaps more realistically, to work through all targets $target_i$ in the target list and perform Experiment($target_i$) on each one. This means the bottom-right node of the GPT becomes an iterative goal 'for each $target_i$ in *targetList*, do Experiment($target_i$)'.[6]

This sol, the rover finds four priority science targets in the canyon through goal IdentifyTargets(*canyon*). Hence, $S(M_1)$ can be expanded to priority-targets-exist(*rock*1) $\wedge$ priority-targets-exist(*target*1) $\wedge \ldots \wedge$ priority-targets-exist(*target*4). Note that such refinement of terms in the success condition means that the relevance (PC) estimates of M_1 are non-monotonic. Before any of the canyon targets are investigated, the completeness of M_1 has fallen from 50% to 20% (i.e., *rock*1 is done but four canyon targets remain). As the rover proceeds with Experiment($target_i$), the progress estimate for M_1 can be further updated.

6.2.3 Battery recharging

In the full scenario we must consider the resources required to perform the battery recharging process. In particular, suppose that a certain amount of memory is required by the recharge process, and so it is necessary that a certain amount of memory be available whenever M_1 is Active. One possibility would be to ensure that the first step of the recovery goal PauseAndRecharge includes pausing all other goals, in order to ensure that the memory is available. Another possibility, which enables more rational and proactive resource management, is to introduce a second maintenance goal M_2, whose task is to ensure that sufficient memory is available (indicated by memory-high). This means we have:

$$M_2 = \text{maintain}(\text{battery} < 50\%, \text{memory-high}, \text{at-base}, \neg\text{battery-functioning})$$

The idea is for M_2 to maintain a certain amount of free memory, which will be required when M_1 becomes active. Since M_2 needs only monitor memory levels when the rover is nearing the time for the battery to be recharged, M_2 only enters the Monitoring state when the battery level drops below 50%, at which point it begins monitoring the memory level. Initially the goal is RC and not PC. If the memory falls too low (indicated by memory-high becoming false), then M_2 is no longer RC; it becomes Active. The recovery

goal for M_2 suspends the Explore(*red*1) goal, which frees up the memory to allow the battery to be recharged. As a result, the battery may be recharged even before the level drops below 30%, since waiting until this happens will only lower the level of available memory even further. Once the battery is recharged, M_1 returns to the Monitoring state, M_2 returns to the Inactive state, and ExploreRegion(*red*1) is resumed. We have not included the details of this variation on the scenario, which is similar in many respects to what is shown above.

6.2.4 Discussion

It should be noted that the estimates used for measuring the completeness of achievement goals are similar to the considerations needed for the activation of resource-based maintenance goals, such as M_1 and M_2. In particular, as this scenario shows, we can use the estimates for the resources needed to complete, e.g., Survey(*canyon*), to also estimate whether the battery will drop below the crucial level of 30% during the execution of this goal, i.e., whether M_1 will become non-RC, and hence whether the battery should be proactively recharged during the execution of the goal. This could be achieved by incorporating such resource estimations into the procedure π supplied by the agent designer for maintenance goals.

We could also use similar methods to prioritise some goals over others (i.e., those which will use less energy) when the battery level is getting low, or to allow a temporary violation of a maintain condition. The latter may occur when the goal Survey(*canyon*) is nearing completion, and only requires sub-goal MeasurementS(*target*4) to be performed. If, for instance, the agent estimates that the battery level will fall to no less than 27% while this is done, the agent may decide to continue with MeasurementS(*target*4) before recharging. There are a number of similar variations that we could consider similarly; the point to note is that estimating the completeness of achievement goals (via CR^t_{lb} and CR^t_{ub}) and the progress of maintenance goals have a close correspondence. Thia requires a mechanism for priotising certain goals over others, which is outside the scope of this paper.

7 Discussion

This section discusses implementation of our reasoning mechanisms, monotonicity and completeness, and reconciles our computational approach with a previous theoretical perspective.

7.1 Implementation

We have implemented our computational approach in the abstract agent language CAN (Winikoff et al., 2002; Sardiña and Padgham, 2011), and have used it to experiment on the above scenario. It is worth noting that the state-based techniques of Harland et al. (2014) simplify this process, in that it is relatively simple to specify rules such as one that requires ExploreRegion(*red*1) to be suspended when either M_1 or M_2 are Active, or that the activation of M_2 requires proactive activation of M_1 (i.e., even if the maintain condition of M_1 is not yet violated).

The implementation consists of around 2,000 lines of Prolog, and is available from the authors' website at: https://titan.csit.rmit.edu.au/~e24991/orpheus/. It has been tested under Ciao and SWI-Prolog. Execution time on the scenario is negligible, compared to the time without the extra code for the completeness calculations. Once the CAN execution rules were implemented, it was a simple task to translate the rules into executable code in Prolog. CAN is an established formalism for specifying and reasoning about processes in agent systems based on the BDI framework. This allows us the necessary precision to reason about particular points in the execution of an agent's plans without having to commit ourselves to a specific programming language. This particular implementation is not intended as a substitute for such languages, but as a means of experiment with our techniques and checking that the mechanisms introduced perform in the intended manner.

While our computational mechanisms are straightforward to implement, they give significant insight into the behaviour of the goals, as demonstrated in the Mars rover scenario. In particular, the relative values of the various completeness measures together with the consumed resources and effects achieved provide information for the agent's deliberation.

Recall that our emphasis is not on raw computational efficiency, but on finding principled, tractable ways to quantify completion estimates for goals. Hence we do not perform efficiency comparisons with other methods: indeed there are none to directly compare with, as previous works (Thangarajah and Padgham, 2011; Morley et al., 2006), etc. do not account for partial completeness, while van Riemsdijk and Yorke-Smith (2010) do not provide computational mechanisms.

Regarding implementation in a deployed agent programming language, a suitable candidate is for example GOAL (Hindriks, 2009). van Riemsdijk and Yorke-Smith (2010) discuss (but do not undertake) what is necessary to implement reasoning with partial satisfaction (of achievement goals) in GOAL. They point out that modifying the language to include the possibility to reason about partial goal satisfaction will likely involve providing a new notion of goal, analogous to the proposed by van Riemsdijk and Yorke-Smith (2010). The completeness metrics such as CR_{lb}^t and CE_S^t can be computed over the agent's belief base; annotations of the program text analogous to the annotations of the GPT may be useful. Then, one can investigate how deliberation such as action selection change according to this new notion of goal and the new information available from the completeness estimates, i.e., whether the existing mechanism in GOAL can in essence be used or whether other mechanisms are required.

7.2 Monotonicity and completeness

Observe the non-monotonic change between stages 5 and 6 of the detailed trace of Table 2. The N_R^t estimates for ExploreRegion(*red*1) and Survey(*canyon*) increase, and the P_R^t estimates decrease, due to the plan selection process for Experiment(*rock*1). Before it is known which of the two choices will be made, the N_R^t and P_R^t estimates are based on the minimum and maximum values across both choices. When a choice of a particular branch is made (stage 6), the estimates can be made more precise, as these now only need to take into account the particular branch chosen. This shows that the plausible intuition that the CR_{lb}^t is non-decreasing does not hold in general.[7]

It should also be noted that we do not dynamically update the effect estimates, which contrasts with our treatment of the resource estimates. In particular, in the transition from stage 5 to stage 6, as discussed above, it is known what branch of the GPT will be used; the resource estimates are updated accordingly but the effect estimates remain the same. The reason for this is that in general we do not know at what point the definite and possible effects become true. In the above scenario, it is clear that the effects SpectralProfile(*rock*1) and ThermalProfile(*rock*1) are not only mutually exclusive, but only become true at the very end of the execution of the relevant plan. However in general we do not know such specific information, i.e., whether possible effects are exclusive or not, and how closely their success is related to the overall success of the goal. This means that when a plan fails, it could potentially have made some of the definite and possible effects true despite its failure. While it may be interesting to explore the use of dynamically-updated effect summaries in future work, we have not done so here as we have been unable to see how to do this in a domain-independent way.

A related issue is the way in which success conditions, definite effects, and potential effects are specified. In the above, effects are assumed to be just atoms and success conditions are sets of atoms. This makes it straightforward to measure discrete effects by simply counting formulae. It is arguably more natural to use more complex formulae for success conditions, such as SpectralProfile(*rock*1) $\vee$ ThermalProfile(*rock*1) as one of the success conditions for Experiment(*rock*1), rather than using the predicate Measured as above. This may allow for potentially richer methods of measuring effects than the simple measure used here.

7.3 Theoretical perspective

In this subsection, we reconcile our computational approach, denoted 'Hetal', with the theoretical framework of van Riemsdijk and Yorke-Smith (2010) ('vRYS'). Our motivation is to ask whether our development of the former can provide the computational mechanism missing in the latter. In doing so, we instantiate vRYS's theoretical framework to a computationally feasible approach.

The scope of vRYS is greater than ours in this article, in that it studies how an agent can use goal completeness information, for instance in 'goal adaptation'. In Hetal we ask instead how to compute goal completeness estimates. The two lines of work have in common the GPT as the basic reasoning structure. Both place attributes on (leaf) nodes which specify factors such as utility and resource usage. Both perform aggregation from leaf to higher-level nodes. vRYS explicitly include aggregation functions for each factor, and default aggregation functions in AND and OR cases, if no specific function is specified. vRYS consider goals only but mention the 'means' by which a goal can be achieved; Hetal explicitly include plans also – which is necessary to obtain a computational mechanism. vRYS consider only achievement goals.

The key question is what defines (full) completeness of a goal G. vRYS suppose each goal has a progress metric, denoted as a set A with a partial order $\leq$ (usually total w.r.t. $a_{\min}$), and further a minimum value, $a_{\min} \in A$, the *completion value*, that should be reached in order to consider the goal to have been completely satisfied. By contrast, Hetal hold the classical view that completeness is defined by the success condition.

Since vRYS do not use the logical conjunction of effects in $S(G)$ to define completeness, they require each goal to have a *progress appraisal function* ψ from $\mathcal{S}$, the set of states, to A. In addition, they posit an accompanying upper bound function,

which we will denote ψ_U, that takes into account the means m that 'will be used for pursuing the goal': ψ_U 'yields (an estimation of) the maximum value in A reachable' from a state $s \in \mathcal{S}''$ with means m. They also mention but do not develop a lower bound function which we will denote ψ_L. vRYS recognise that all three functions may be estimates, which concurs with the principles of Hetal's mechanism.

Hetal explicitly compute lower and upper-bounds. These bounds can be seen as equivalent to vRYS's ψ (or ψ_L) and ψ_U for the progress metric. While vRYS consider only a single such 'metric', Hetal compute multiple metrics – multiple resources and multiple effects. However, vRYS note "Besides the metric chosen as the progress metric, the agent (or designer) might have interest in others: e.g., progress may be defined in terms of tracks searched, but time taken could be an additional relevant factor in the team's decisions" (van Riemsdijk and Yorke-Smith, 2010).

From this analysis we conclude that the general philosophy of the two lines of work seem compatible. The computation mechanisms of Hetal for lower and upper-bounds can be seen as computing the ψ_L and ψ_U bounds of the progress metric of vRYS. That is, if one metric of those computed by Hetal is designated as the progress metrics for a goal in vRYS's framework, then Hetal provide the computational mechanism that is lacking in vRYS's framework.

In more detail, consider an achievement or maintenance goal G being pursued by an agent using a plan that in vRYS's notation is the meansm.[8] First, in the case of (the usage of) a resource r being selected as the progress metric, we have: $\psi_L(G) = CR^t_{lb}(G)$[9], $\psi(G) = R^t(G)$, and $\psi_U(G) = CR^t_{ub}(G)$. Second, in the case of success-condition effects, we have: $\psi_L(G) = CE^t_{lb}(G), \psi(G) = CE^t_S(G)$, and $\psi_U(G) = CE^t_{ub}(G)$. Note that for maintenance goals, the resource-based metric and the effects-summary-based metric apply to its recovery completeness, while the success-condition-based metric applies to both its recovery and its permanent completeness.

An important point of difference between the two lines of work is the key question of what constitutes completeness of an achievement goal. vRYS give the agent designer the ability to define completeness through the choice of which metric (which could be one of those of Hetal) constitutes the designated progress metric for a goal, together with the designated $a_{\min}$ value. Hetal, by contrast, give this ability through what the agent designer specifies in $S(G)$. Hence, to reconcile the framework of vRYS with the classical view, $S(G)$ should be set to be the condition that corresponds to $\psi(G) \geq a_{\min}$.

Let us make two final remarks. First, if a different metric than resources or effects is selected as the progress metric for a goal, then the metrics computed by Hetal can be useful in agent deliberation, although not defining completeness – as vRYS remark (van Riemsdijk and Yorke-Smith, 2010).

Second, in line with Hetal's computation of multiple metrics, an interesting extension of the vRYS framework would be to have multiple progress metrics, i.e., to choose more than one metric to be the progress metric for a goal. The question then is how to define completeness (since they do not use $S(G)$), the simplest approach being $\psi_i(G) \geq a^i_{\min}$ for all progress metrics i.

8 Conclusions

The practice of BDI agent systems is that goal accomplishment is discrete. This article contributes a principled mechanism for computing completeness of top-level goals of a BDI-style agent in order to inform the agent's deliberation. The overall aim in our work is to provide an approach that is principled and generic and that can be used computationally to quantify a measure of completeness for both goals of accomplishment (achievement goals) and goals of monitoring (maintenance goals). By enabling an agent to compute a finer-grained approximation of the level of completeness of its goals, we endow agents to make more nuanced and potentially more suitable decisions. For example, when reasoning to resolve goal conflicts (Thangarajah and Padgham, 2011), the agent may choose to continue with the goal that is more complete than the other.

As a further, specific example, in the Mars rover scenario, given an imminent deadline such as the approach of dusk, it may be reasonable to terminate the execution of IdentifyTargets(*canyon*) once a sufficient fraction of the canyon has been surveyed, or if a sufficient number of targets has been found. This would allow the top-level goal (ExploreRegion(*red*1)) to be completed before the deadline, despite not having fully explored the canyon for all possible targets. The information computed at run-time by our approach provides a quantifiable basis for such decisions.

First, our approach leverages previous work on resource and effects summarisation but we go beyond by accommodating both dynamic resource summaries and goal effects, while also allowing a non-binary quantification of goal completeness. We can accommodate BDI agents with generative planning capabilities (Sardiña and Padgham, 2011), since we compute dynamically from the GPT.

Second, we examine what it means for a maintenance goal to be 'complete', and compute estimates of the notions of permanently complete and RC, leverage recent work on computing completeness of achievement goals. Throughout, our implementation, in the abstract agent language CAN, enjoys low computational overhead.

Third, we examine our work on practical computation of progress estimates in the light of an earlier theoretical framework on BDI goal completeness. We showed that computation of lower and upper-bounds of Thangarajah et al. (2015) can be seen as computing the bounds on the progress metric of van Riemsdijk and Yorke-Smith (2010), depending on the choice of progress metric. As a by product of this reconciliation, we implicitly extended the reach of the theoretical framework to include maintenance goals. We also instantiated the theoretical framework to a computationally-feasible approach.

8.1 Assumptions and limitations

The assumptions and limitations of this article are summarised as follows:

1 We assume that the agent's beliefs are generally true. However the agent does not have knowledge of the world, only beliefs.

2 We assume that the agent is rational and optimistic, but that its goals do not always succeed.

3 For a maintenance goal, we assume that the success of either a preventive or recovery goal is sufficient to restore the maintenance condition, and that these goals will typically succeed.

4 We assume that effect and resource summaries on leaf nodes are provided by the agent designer.

5 We assume that the value of a resource is zero if the resource is not in a resource subset.

6 We assume that all goal effects are expressed as atomic formulae.

7 We assume that effects are atoms and that success conditions are sets of atoms.

8.2 Future work

This article leads to a number of extensions and future directions, which we now briefly describe.

First, some potential aspects for further work relate to the potentially non-monotonic nature of effects. Despite having made one or more effects become true, these effects could be undone by either interactions between plans (Thangarajah et al., 2003), by another agent, or by interactions with the environment, such as wind moving rocks around after the agent has positioned its robot arm, or an identified target being moved from its initial location. This means the calculations above would need to take into account the need to re-establish effects which had been previously made true. Further, additional resources beyond those estimated previously might be needed for the plan to succeed.

Second, a further observation that may be made about the example involving the wind moving the rocks is the need for preconditions to be maintained. In other words, there is often an implicit connection between goals, such as PositionArm(*rock*1) and MeasurementS(*rock*1) in Figure 4, in that we assume that PositionArm(*rock*1) remains true during the execution of MeasurementS(*rock*1). This suggests that an interesting line of further work would be to consider a more complex plan that involved maintaining any necessary pre-requisites for the duration of the plan; the cost of re-establishing these, as discussed above, would then become the cost of maintaining the relevant pre-requisite conditions.

Third, as noted above, the traditional notion of accomplishment for achievement goals are discrete, i.e., it is either achieved or not. In some circumstances, it may be appropriate to consider a 'less binary' notion of success. For example, for M_1 in the rover scenario above, consider when the battery level drops to 29% and so the rover recharges. However, as this will interrupt the achievement of other goals, it may be that it is not strictly necessary to recharge the battery in full to complete the day's schedule. If it is known that completing the schedule will require only say 90% of the battery charge, it may be better to cease recharging when this level is achieved, and spend the time saved on science opportunities. This will involve considering the goal 'complete enough', rather than strictly complete.

Fourth, another line of further work is suggested by the discussion in Section 6.2 on iterative goals, and in particular how they may apply to the issue of finding and analysing science targets. The notion of an iterative goal, either triggered by a rule of some sort or as an extension of the notion of a maintenance goal, is something that we intend to explore further. Some related issues concern the relationships between the various components of a maintenance goal. For example, it seems natural to consider maintenance goals in which the success condition is the negation of the context condition,

so that there is a direct and intuitive connection between what triggers activity and what ceases it. It also seems natural to consider maintenance goals in which the success condition of the recovery goal implies the maintenance condition, so that there is a formal reflection of the intuitive goal that the achievement of the recovery goal will in fact restore the desired state. For M_1 and M_2 above, it is clear that restoring the battery level to 100% will ensure any specified level of minimum charge is reached; in other examples, it may not be so obvious.

Fifth, one can consider the resource costs for failed plans for achievement goals (including repair and preventative goals). For example, if the Mars rover attempts a spectroscopic analysis, but finds that it fails, it may still consume energy, drill bits, memory and time in doing so. This means that we need to adjust the calculations for the definite and potential resource estimates for completing the goal to take the resources used in failed sub-goals into account.

Sixth, for maintenance goals, there is the possibility of leveraging look ahead predictions in order to refine progress estimates, based the likelihood or number of predicted future maintain condition violations. As discussed in Section 5.3, reasoning about predicted resource use may be useful for resolving conflicts and improving resource utilisation.

Seventh, as noted in Section 2, there are two bodies of literature for which the connections with our work are interesting to explore: the works on reasoning about time in agent programming languages, and the works on the degree of completeness of fuzzy goals.

Lastly, a further conclusion of our work is that measuring the progress of an agent that has maintenance goals can lead to 'higher-level' considerations of progress. For example, potentially a maintenance goal could be active for the entire existence of the agent, which in the Mars rover scenario could be a mission of several months. Given that the rover's activity is restricted to daytime hours, the agent's activity over the course of the mission involves a regular cycle of returning to its base each night. It would seem natural to integrate a regular update of the overall progress of its mission, based largely on techniques similar to those discussed here, to allow it to compare the current sol's progress to previous sols, as well as to estimate its overall progress. For example, at the end of a given sol, the rover may compare the time spent in reactively charging the battery that sol to previous ones, as well as proactive recharging, in order to determine whether its battery use could be better managed. This may also include a comparison of the completion estimates with their actual values, again to potentially improve the accuracy of its predictions and hence of its goal management.

Acknowledgements

We thank the reviewers for their comments which helped to improve this article. We thank the reviewers at AAMAS'14, ECAI'14 and AAMAS'15, where fragments of this work appeared. JT acknowledges the ARC Discovery Grant number DP1094627. NYS acknowledges the AUB University Research Board Grant number 102853. NYS thanks the Operations group at the Cambridge Judge Business School and the fellowship at St Edmund's College, Cambridge. This work was partially supported by TAILOR, a project funded by EU Horizon 2020 research and innovation program under Grant number 952215.

References

Aha, D.W. (2018) 'Goal reasoning: foundations, emerging applications, and prospects', *AI Magazine*, Vol. 39, No. 2, pp.3–24.

Alechina, N., Bulling, N., Logan, B. and Nguyen, H.N. (2017) 'The virtues of idleness: a decidable fragment of resource agent logic', *Artificial Intelligence*, Vol. 245, pp.56–85, https://doi.org/10.1016/j.artint.2016.12.005.

Baral, C., Eiter, T., Bjaereland, M. and Nakamura, M. (2008) 'Maintenance goals of agents in a dynamic environment: formulation and policy construction', *Artificial Intelligence*, Vol. 172, Nos. 12–13, pp.1429–1469.

Barkan, M. and Kaminka, G.A. (2019) 'Towards predictive execution monitoring in BDI recipes', in *Proceedings of 18th International Conference on Autonomous Agents and MultiAgent Systems, (AAMAS'19)*, pp.1808–1810.

Bordini, R.H., Bazzan, A.L.C., de Oliveira Jannone, R., Basso, D.M., Vicari, R.M. and Lesser, V.R. (2002) 'AgentSpeak(XL): efficient intention selection in BDI agents via decision-theoretic task scheduling', in *Proc. of AAMAS'02*, pp.1294–1302.

Bratman, M.E. (1987) *Intention, Plans and Practical Reason*, University of Chicago Press, Chicago, IL.

Braubach, L., Pokahr, A., Moldt, D. and Lamersdorf, W. (2004) 'Goal representation for BDI agent systems', in *Proc. of ProMAS'04, LNCS*, No. 3346, pp.44–65.

Broersen, J.M., Dastani, M., Hulstijn, J., Huang, Z. and van der Torre, L.W.N. (2001) 'The BOID architecture: conflicts between beliefs, obligations, intentions and desires', in *Proc. of Agents'01*, pp.9–16.

Clement, B.J., Durfee, E.H. and Barrett, A.C. (2007) 'Abstract reasoning for planning and coordination', *J. Artificial Intelligence Research*, Vol. 28, pp.453–515, https://doi.org/10.1613/jair.2158.

Collins, J., Ketter, W. and Gini, M.L. (2009) 'Flexible decision control in an autonomous trading agent', *Electronic Commerce Research and Applications*, Vol. 8, No. 2, pp.91–105.

Darimont, R., Delor, E., Massonet, P. and van Lamsweerde, A. (1997) 'iGRAIL/KAOS: an environment for goal-driven requirements engineering', in *Proc. of 19th International Conference on Software Engineering (ICSE'97)*, pp.612–613.

Dastani, M., van Riemsdijk, M.B. and Meyer, J-J.C. (2006) 'Goal types in agent programming', in *Proc. of ECAI'06*, pp.220–224.

de Silva, L., Sardina, S. and Padgham, L. (2009) 'First principles planning in BDI systems', in *Proc. of AAMAS'09*, pp.1105–1112.

Duff, S., Harland, J. and Thangarajah, J. (2006) 'On proactivity and maintenance goals', in *Proc. of AAMAS'06*, pp.1033–1040.

Duff, S., Thangarajah, J. and Harland, J. (2014) 'Maintenance goals in intelligent agents', *Computational Intelligence*, Vol. 30, No. 1, pp.71–114.

Evertsz, R., Thangarajah, J., Yadav, N. and Ly, T. (2015) 'A framework for modelling tactical decision-making in autonomous systems', *Journal of Systems and Software*, Vol. 110, pp.222–238.

Georgeff, M. and Rao, A. (1998) 'Rational software agents: from theory to practice', in *Agent Technology: Foundations, Applications, and Markets*, Chapter 8, pp.139–160, Springer, New York.

Grosz, B.J. and Hunsberger, L. (2006) 'The dynamics of intention in collaborative activity', *Cognitive Systems Research*, Vol. 7, Nos. 2–3, pp.259–272.

Haddawy, P. and Hanks, S. (1992) 'Representations for decision theoretic planning: utility functions for deadline goals', in *Proc. of KR'92*, pp.71–82.

Harland, J., Morley, D.N., Thangarajah, J. and Yorke-Smith, N. (2014) 'An operational semantics for the goal life-cycle in BDI agents', *Autonomous Agents and Multi-Agent Systems*, Vol. 28, No. 4, pp.682–719.

Harland, J., Morley, D.N., Thangarajah, J. and Yorke-Smith, N. (2017) 'Aborting, suspending, and resuming goals and plans in BDI agents', *Autonomous Agents and Multi-Agent Systems*, Vol. 31, No. 2, pp.288–331.

Herzig, A., Lorini, E., Perrussel, L. and Xiao, Z. (2017) 'BDI logics for BDI architectures: old problems, new perspectives', *KI*, Vol. 31, No. 1, pp.73–83.

Hindriks, K.V. (2009) 'Programming rational agents in GOAL', in Fallah-Seghrouchni, A.E., Dix, J., Dastani, M. and Bordini, R.H. (Eds.): *Multi-Agent Programming, Languages, Tools and Applications*, pp.119–157, Springer, Cham, Switzerland.

Hindriks, K.V. and van Riemsdijk, M.B. (2008) 'Satisfying maintenance goals', in *Proc. of DALT'07, LNCS*, Vol. 4897, pp.86–103.

Hindriks, K.V., Jonker, C. and Pasman, W. (2008) 'Exploring heuristic action selection in agent programming', in *Proc. of ProMAS'08, LNCS*, Vol. 5442, pp.24–39.

Hindriks, K.V., van der Hoek, W. and van Riemsdijk, M.B. (2009) 'Agent programming with temporally extended goals', in *Proc. of AAMAS'09*, pp.137–144.

Holton, R. (2008) 'Partial belief, partial intention', *Mind*, Vol. 117, No. 465, pp.27–58.

Horling, B., Lesser, V.R., Vincent, R. and Wagner, T. (2006) 'The soft real-time agent control architecture', *Autonomous Agents and Multi-Agent Systems*, Vol. 12, No. 1, pp.35–91.

Hsu, C., Wah, B.W., Huang, R. and Chen, Y. (2007) 'Constraint partitioning for solving planning problems with trajectory constraints and goal preferences', in *Proc. of IJCAI'07*, pp.1924–1929.

Huang, Z. and Bell, J. (1997) 'Dynamic goal hierarchies', in *Proc. of the 1997 AAAI Spring Symp. on Qualitative Preferences in Deliberation and Practical Reasoning*, pp.9–17.

Ingrand, F.F., Chatila, R., Alami, R. and Robert, F. (1996) 'PRS: a high level supervision and control language for autonomous mobile robots', in *Proc. of ICRA'96*, pp.43–49.

Jha, S., Raman, V., Sadigh, D. and Seshia, S.A. (2018) 'Safe autonomy under perception uncertainty using chance-constrained temporal logic', *J. Automated Reasoning*, Vol. 60, No. 1, pp.43–62.

Jiang, J., Thangarajah, J., Aldewereld, H. and Dignum, V. (2014) 'Reasoning with agent preferences in normative multi-agent systems', in *Proc. of AAMAS'14*, pp.1373–1374.

Kamali, K., Fan, X. and Yen, J. (2007) 'Towards a theory for multiparty proactive communication in agent teams', *Intl. J. of Cooperative Information Systems*, Vol. 16, No. 2, pp.271–298.

Kamar, E., Gal, Y. and Grosz, B.J. (2009) 'Incorporating helpful behavior into collaborative planning', in *Proc. of AAMAS'09*, pp.875–882.

Kaminka, G.A., Yakir, A., Erusalimchik, D. and Cohen-Nov, N. (2007) 'Towards collaborative task and team maintenance', in *Proc. of AAMAS'07*, pp.1–8.

Katarzyniak, R. and Popek, G. (2013) 'Integration of modal and fuzzy methods of knowledge representation in artificial agents', *International Journal of Software Engineering and Knowledge Engineering*, Vol. 23, No. 1, pp.13–30.

Khan, S.M. and Lespérance, Y. (2010) 'A logical framework for prioritized goal change', in *Proc. of AAMAS'10*, pp.283–290.

Lesser, V.R., Decker, K., Wagner, T., Carver, N., Garvey, A., Horling, B., Neiman, D.E., Podorozhny, R.M., Prasad, M.V.N., Raja, A., Vincent, R., Xuan, P. and Zhang, X. (2004) 'Evolution of the GPGP/TÆMS domain-independent coordination framework', *Autonomous Agents and Multi-Agent Systems*, Vol. 9, Nos. 1–2, pp.87–143.

Letier, E. and van Lamsweerde, A. (2004) 'Reasoning about partial goal satisfaction for requirements and design engineering', in *Proc. of SIGSOFT'04*, pp.53–62.

Logan, B., Thangarajah, J. and Yorke-Smith, N. (2017) 'Progressing intention progression: a call for a Goal-Plan Tree contest', in *Proc. of AAMAS'17*, pp.768–772.

Morley, D. and Myers, K. (2004) 'The SPARK agent framework', in *Proc. of AAMAS'04*.

Morley, D., Myers, K.L. and Yorke-Smith, N. (2006) 'Continuous refinement of agent resource estimates', in *Proc. of AAMAS'06*, pp.858–865.

Myers, K.L. and Yorke-Smith, N. (2005) 'A cognitive framework for delegation to an assistive user agent', in *Proc. of AAAI 2005 Fall Symposium on Mixed-Initiative Problem-Solving Assistants*, pp.94–99.

Roberts, M., Borrajo, D., Cox, M. and Yorke-Smith, N. (2018) 'Special issue on goal reasoning', *AI Communications*, Vol. 31, No. 2, pp.115–116.

Rönnquist, R. (2007) 'The goal oriented teams (GORITE) framework', in *Proc. of ProMAS'07, LNCS*, Vol. 4908, pp.27–41.

Sardiña, S. and Padgham, L. (2011) 'A BDI agent programming language with failure handling, declarative goals, and planning', *Autonomous Agents and Multi-Agent Systems*, Vol. 23, No. 1, pp.18–70.

Shen, S., O'Hare, G.M.P. and Collier, R.W. (2004) 'Decision-making of BDI agents, a fuzzy approach', in *Proceedings of 1st International Conference on Computer and Information Technology (CIT'04)*, pp.1022–1027.

Smith, D.E. (2004) 'Choosing objectives in over-subscription planning', in *Proc. of ICAPS'04*, pp.393–401.

Thangarajah, J. and Padgham, L. (2011) 'Computationally effective reasoning about goal interactions', *J. Automated Reasoning*, Vol. 47, No. 1, pp.17–56.

Thangarajah, J., Harland, J. and Yorke-Smith, N. (2007) 'A soft COP model for goal deliberation in a BDI agent', in *Proc. of CP'07Workshop on Constraint Modelling and Reformulation (ModRef'07)*, pp.61–75.

Thangarajah, J., Harland, J. and Yorke-Smith, N. (2015) 'Estimating the progress of maintenance goals', in *Proc. of AAMAS'15*, pp.1645–1646.

Thangarajah, J., Harland, J., Morley, D.N. and Yorke-Smith, N. (2014a) 'Quantifying the completeness of goals in BDI agents', in *Proc. of ECAI'14*, pp.879–884.

Thangarajah, J., Harland, J., Morley, D.N. and Yorke-Smith, N. (2014b) 'Towards quantifying the completeness of BDI goals', in *Proc. of AAMAS'14*, pp.1369–1370.

Thangarajah, J., Padgham, L. and Winikoff, M. (2003) 'Detecting and avoiding interference between goals in intelligent agents', in *Proc. of IJCAI'03*, pp.721–726.

Thangarajah, J., Winikoff, M., Padgham, L. and Fischer, K. (2002) 'Avoiding resource conflicts in intelligent agents', in *Proc. of ECAI-02*, pp.18–22.

van der Hoek, W., Jamroga, W. and Wooldridge, M. (2007) 'Towards a theory of intention revision', *Synthese*, Vol. 155, No. 2, pp.265–290.

van Riemsdijk, M.B. and Yorke-Smith, N. (2010) 'Towards reasoning with partial goal satisfaction in intelligent agents', in *Proc. of ProMAS'10, LNCS*, Vol. 6599, pp.41–59.

van Riemsdijk, M.B. and Yorke-Smith, N. (2012) 'Towards reasoning with partial goal satisfaction in intelligent agents', in *Programming Multi-Agent Systems VIII, LNCS*, Springer, New York, Vol. 6599, pp.41–59.

Vikhorev, K., Alechina, N. and Logan, B. (2011) 'Agent programming with priorities and deadlines'. in *Proc. of AAMAS'11*, pp.397–404.

Visser, S., Thangarajah, J., Harland, J. and Dignum, F. (2016) 'Preference-based reasoning in BDI agent systems', *Autonomous Agents and Multi-Agent Systems*, Vol. 30, No. 2, pp.291–330.

Vukovic, M. and Robinson, P. (2005) 'GoalMorph: partial goal satisfaction for flexible service composition', *International Journal of Web Services Practices*, Vol. 1, Nos. 1–2, pp.40–56.

Wilson, M.A., McMahon, J., Wolek, A., Aha, D.W. and Houston, B.H. (2018) 'Goal reasoning for autonomous underwater vehicles: responding to unexpected agents', *AI Communications*, Vol. 31, No. 2, pp.151–166.

Winikoff, M. (2005) 'JACK intelligent agents: an industrial strength platform', in *Multi-Agent Programming*, pp.175–193, Springer, New York.

Winikoff, M., Padgham, L., Harland, J. and Thangarajah, J. (2002) 'Declarative and procedural goals in intelligent agent systems', in *International Conference on Principles of Knowledge Representation and Reasoning*, Morgan Kaufman, San Francisco, California, USA.

Yao, Y. and Logan, B. (2016) 'Action-level intention selection for BDI agents', in *Proc. of AAMAS'16*, pp.1227–1236.

Yao, Y., de Silva, L. and Logan, B. (2016a) 'Reasoning about the executability of Goal-Plan Trees', in *Proc. of EMAS'16, LNCS*, Vol. 10093, pp.176–191.

Yao, Y., Logan, B. and Thangarajah, J. (2016b) 'Intention selection with deadlines', in *Proc. of ECAI'16*, pp.1700–1701.

Yao, Y., Logan, B. and Thangarajah, J. (2016c) 'Robust execution of BDI agent programs by exploiting synergies between intentions', in *Proc. of AAAI'16*, pp.2558–2565.

Zhou, Y. and Chen, X. (2004) 'Partial implication semantics for desirable propositions', in *Proc. of KR'04*, pp.606–612.

Zhou, Y., van der Torre, L. and Zhang, Y. (2008) 'Partial goal satisfaction and goal change: weak and strong partial implication, logical properties, complexity', in *Proc. of AAMAS'08*, pp.413–420.

Notes

1 We thank an anonymous reviewer for this point.

2 One could also include the look ahead function π in the goal syntax, whereupon the look ahead function could differ by goal.

3 Although note that in the timescale of the scenario of Section 4, memory is not reused.

4 Since x is not trying to achieve S, completeness based on resource usage is irrelevant, as x is not intentionally expending resources to accomplish S. Likewise, completeness based on effect summaries is irrelevant.

5 We abbreviate *rock*1 to *r*1, *target* to *t*, *canyon* to *c*, ArmPositioned(_) to AP(_), SpectralProfile(_) to SP(_), etc.

6 While there is no explicit loop construct in the CAN (Sardiña and Padgham, 2011) language used in our implementation, described below, such a construct is easily be simulated by a particular combination of a goal and plan. Specifically, to execute a plan such as 'while C do P', we adopt the CAN goal *loopP* = *achieve*(C, $\neg C$, $\neg C$), with the corresponding $P; ?\neg C \triangleright fail$. The goal *loopP* is activated when C is true, and is dropped when $\neg C$ is true. The plan for *loopP* will perform P and then pose the query $?\neg C$. If this succeeds, $\neg C$ is true, the plan for *loopP* terminates, and *loopP* succeeds. Otherwise, the query fails, and the $\triangleright$ construct ensures that the plan $P; ?\neg C \triangleright fail$ fails. Due to the semantics of CAN, this means that the plan for *loopP* is restarted, and P and the subsequent query are performed again. This process will only halt when C becomes true. Based on this kind of rule, one could add an explicit *while* construct to CAN.

7 A similar occurrence is found in the change from stage 0 to 1, but as no goals have completed at this point, all the completion estimates are 0% despite this narrowing of the resource estimates.

8 For a maintenance goal, the plan/means is that used for the R or P sub-goal when it becomes active.

9 There is a caveat in the case of resources: $CR^t_{lb} = 1$ does not imply that the goal is complete w.r.t. $S(G)$ – the moment a plan uses all the necessary resources does not imply the plan is complete at the moment – whereas vRYS have $\psi_L \geq a_{\min}$ implies $\psi \geq a_{\min}$. That is, Hetal's resources lower bound estimate is not a strict lower bound w.r.t. completeness if used in the way proposed. In the case of effects, CR^t_{lb} is a strict lower bound provided effects are monotonic.