Intention Progression with Temporally Extended Goals

Yuan Yao[1] , **Natasha Alechina**[2,3] , **Brian Logan**[4,3]
[1]University of Nottingham Ningbo China
[2] Open University Netherlands
[3]Utrecht University
[4]University of Aberdeen
yuan.yao@nottingham.edu.cn, {n.a.alechina, b.s.logan}@uu.nl

Abstract

The Belief-Desire-Intention (BDI) approach to agent development has formed the basis for much of the research on architectures for autonomous agents. A key advantage of the BDI approach is that agents may pursue multiple intentions in parallel. However, previous approaches to managing possible interactions between concurrently executing intentions are limited to interactions between simple achievement goals (and in some cases maintenance goals). In this paper, we present a new approach to intention progression for agents with *temporally extended goals* which allow mixing reachability and invariant properties, e.g., "travel to location A while not exceeding a gradient of 5%". Temporally extended goals may be specified at run-time (top-level goals), and as subgoals in plans. In addition, our approach allows human-authored plans and plans implemented as reinforcement learning policies to be freely mixed in an agent program, allowing the development of agents with 'neuro-symbolic' architectures.

1 Introduction

The Belief-Desire-Intention (BDI) approach to agent development [Rao and Georgeff, 1992] has formed the basis for much of the research on architectures for autonomous agents [de Silva *et al.*, 2020]. In BDI-based agent programming languages, e.g., [Bordini *et al.*, 2007; Dastani, 2008], the behaviour of an agent is specified in terms of beliefs, goals, and plans. *Beliefs* represent the agent's information about the environment. *Goals* represent states of the environment the agent is trying to bring about. *Plans* are the means by which the agent can modify the environment in order to achieve its goals. Plans are composed of *steps* which are either *basic actions* that directly change the agent's environment or *subgoals* which are in turn achieved by other plans. For each top-level goal, the agent selects a plan which forms the root of an *intention*, and commences executing the steps in the plan. If the next step in an intention is a subgoal, a (sub)plan is selected to achieve the subgoal and added to the intention, and the steps in the (sub)plan are then executed and so on. This process of repeatedly choosing and executing plans is referred to as the agent's *deliberation cycle*.

While the BDI approach has been very successful, it has the disadvantage that all the agent's plans must be written by a human developer.[1] For nondeterministic environments, e.g., control of cyber-physical systems, this can be challenging. In addition, in many BDI languages, e.g., [Bordini *et al.*, 2007], the developer must anticipate any possible interactions between plans to achieve the agent's goals. This typically involves writing special purpose scheduling code which specifies the order in which goals should be achieved and which plans can execute concurrently. For example, consider a Mars Rover which has a goal to perform a soil experiment at location A and another goal to recharge its battery at location B. Suppose its current battery level only allows the Mars Rover to move from the current position to A or B, and it cannot move from A to B (or B to A) without recharging its battery. In this situation, the only way to achieve both goals is to pursue the goal of recharging the battery first.

There has been work on allowing the agent to autonomously manage possible interactions between concurrently executing plans for multiple goals at run-time. This is termed *intention progression* in [Logan *et al.*, 2017]. Several approaches have been proposed including: Summary Information-based e.g., [Thangarajah and Padgham, 2011], Coverage-based approaches e.g., [Waters *et al.*, 2015] and most recently, MCTS-based approaches e.g., [Yao and Logan, 2016]. These approaches typically involve reasoning over a representation of the agent's programme called a goal-plan tree. A *goal-plan tree* (GPT) represents the relationships between an agent's goals, plans, the subgoals in those plans and their associated subplans as an and-or tree structure [Thangarajah *et al.*, 2003a]. However, these approaches are limited to handling interactions between simple achievement goals, e.g., "move to A'", and, in some cases, maintenance goals, e.g., "maintain a battery level of at least 10%" e.g., [Duff *et al.*, 2006; Wu *et al.*, 2023].

In this paper we present a new approach to intention progression for agents with temporally extended goals. *Temporally extended goals* allow mixing reachability and invari-

[1]There has been some work on generating new plans at run-time using AI planning techniques, e.g., [de Silva *et al.*, 2009], however, this is not supported in widely-used BDI languages and platforms.

ant properties; for example, "travel to location A while not exceeding a gradient of 5%". Temporally extended goals have been proposed before in the BDI literature; for example, PRS supported reachability and invariant goals [Georgeff and Lansky, 1987]. However, the agent developer was responsible for writing code (meta-plans) to ensure that the execution of the agent maintains the invariant. To the best of our knowledge, our approach is the first to allow an agent to autonomously progress its intentions to achieve multiple temporally extended goals concurrently. Temporally extended goals may be specified at run-time (top-level goals), and as subgoals in plans. Our approach does not rely on goal-plan trees [Thangarajah *et al.*, 2003a], and supports a more general notion of plan than that typically used in the BDI literature. In particular, it allows human-authored plans and plans implemented as reinforcement learning (RL) policies to be freely mixed in an agent program, allowing the development of agents with 'neuro-symbolic' architectures which combine behaviours produced by machine learning and programmed behaviours [Bordini *et al.*, 2020]. For example, the overall sequencing of tasks may be specified by a human developer, with low-level control implemented using RL policies.

2 BDI Agents with Temporally Extended Goals

In this section, we introduce the basic components of a BDI agent with temporally extended goals, including beliefs, goals, actions and plans, and define the intention progression problem.

Beliefs. We assume a finite set of propositions P. $Q \subseteq \wp(P)$ is a non-empty finite set of environment states (truth assignments to propositions in P). We denote the set of all literals over P by $\bar{P}$. The agent's *beliefs* $B = \{b_1, \ldots, b_n\}$ are a finite set of ground literals defined over $\bar{P}$ representing its information about the environment.

Goals. The agent's *state goals* $S = \{f_1, \ldots, f_m\}$ are a finite set of propositional formulas built from P that can be evaluated to true or false in the states. State goals are sometimes termed *achievement goals* in the BDI literature [de Silva *et al.*, 2020]. The agent's *temporally extended goals* T are temporal formulas built from S and $\bar{P}$. More precisely a temporally extended goal is a formula ϕ of the form:

$$\psi = p \mid \neg p \mid p \wedge q \mid p \vee q$$
$$\phi = f \mid F\phi \mid \psi\, U\, \phi \mid G\, \psi \mid \phi \wedge \phi' \mid \phi \vee \phi'$$

where $p, q \in P$ and $f \in S$.[2] Observe that we have a set of propositional formulas ψ that is not the same as the set of state goals; state goals are a strict subset of the set of propositional formulas, as there are only finitely many state goals. We also only allow propositional formulas as invariants. As explained below, invariants do not have a corresponding plan; rather they restrict the choice and execution of other plans.

The temporally extended goal $F\phi$ indicates that ϕ should be achieved in some future state, $\psi\, U\, \phi$ that ϕ should be achieved in some future state while maintaining the invariant ψ in all intervening states, and $G\, \psi$ that ψ should be true in all future states. Temporally extended goals thus constrain the future execution of the agent so as to bring about state goals in ϕ and/or maintain literals in ψ. For example, $F\mathit{batt\_full} \wedge G\, \neg \mathit{batt\_empty}$ states that the agent should recharge at some point and never allow the battery to become fully discharged.

Temporally extended goals are interpreted on finite traces. A *trace* τ is a sequence of states $q \in \wp(P)$. We denote the length of a trace τ as $\mathit{length}(\tau)$. We denote the positions on the trace as $\tau(i)$ with $0 \leq i \leq \mathit{last}$, where $\mathit{last} = \mathit{length}(\tau) - 1$ is the last state of the trace. Given an interpretation τ we inductively define when a temporally extended goal ϕ is true at position i (for $0 \leq i \leq \mathit{last}$), $\tau, i \models \phi$ as follows:

- $\tau, i \models p$ for $p \in P$ iff $p \in \tau(i)$
- $\tau, i \models \neg\phi$ iff $\tau, i \not\models \phi$
- $\tau, i \models \phi_1 \wedge \phi_2$ iff $\tau, i \models \phi_1$ and $\tau, i \models \phi_2$
- $\tau, i \models \phi_1 \vee \phi_2$ iff $\tau, i \models \phi_1$ or $\tau, i \models \phi_2$
- $\tau, i \models F\phi$ iff for some j, $i \leq j \leq \mathit{last}$, $\tau, j \models \phi$
- $\tau, i \models \psi\, U\, \phi$ iff for some j, $i \leq j \leq \mathit{last}$, $\tau, j \models \phi$, and for all k, $i \leq k < j$, $\tau, k \models \psi$
- $\tau, i \models G\, \phi$ iff for all j, $i \leq j \leq \mathit{last}$, $\tau, j \models \phi$

A goal ϕ is true in τ, denoted $\tau \models \phi$, if $\tau, 0 \models \phi$.

Each temporally extended goal ϕ is associated with a *utility* r_ϕ representing the utility obtained by the agent if its future execution is such that ϕ holds, i.e., if the trace of the agent's execution τ is such that $\tau \models \phi$. The agent receives r_ϕ when ϕ becomes true on τ, i.e., for reachability goals at the state i, $0 \leq i \leq \mathit{last}$ such that $\tau, i \models \phi$, and for all j, $0 \leq j < i$, $\tau, j \not\models \phi$, and for invariants $G\, \phi$ in the last state on the trace.

Actions. The agent can perform a set $\mathit{Act} = \{\alpha_1, \ldots, \alpha_k\}$ of primitive *actions* in the environment. The *preconditions* of an action α_i, $\phi = \mathit{pre}(\alpha_i)$, are a set of literals that must be true for the action to be executable, and the *postconditions* of α_i, $\psi = \mathit{post}(\alpha_i)$, are a set of literals that are true after the execution of the action. For simplicity, we assume that actions are deterministic: if the preconditions of an action hold, then the postconditions of the action hold after executing the action.[3] An action is *executable* if $B \models \phi$.

Plans. Each of the agent's state goals $f \in S$ is associated with a set of plans $\pi_1, \ldots, \pi_n$ that achieve f. Each plan π_i is a three tuple of the form $(f, \chi_i, \mathit{next}_i)$, where f is the state goal achieved by π, $\chi = \mathit{pre}(\pi_i)$ is a set of literals specifying the *context condition* which must be true for π_i to begin execution, and $\mathit{next}_i \rightarrow \mathit{Act}$ is a function of zero arguments which returns the next step in the plan, which may be either an action, a temporally extended subgoal, or nil (indicating that the plan is complete).[4] Note that this approach to defining plans is more general than that typically used in the BDI

[2] We do not include a "next" temporal operator in the syntax as we do not want to make any assumptions about when the next time instant occurs.

[3] Our approach can be extended in a straightforward way to handle non-deterministic actions as in [Yao *et al.*, 2016]

[4] Test goals which are evaluated against the agent's beliefs can be easily accommodated, e.g., by having the next_i function return a "test" action whose preconditions are the test.

literature, where plans are defined as sequences of actions and (state) subgoals [de Silva *et al.*, 2020]. In particular, it allows human-authored plans and plans implemented as RL policies to be freely mixed in an agent program, allowing the development of agents with 'neuro-symbolic' architectures which combine behaviours produced by machine learning and programmed behaviours [Bordini *et al.*, 2020]. For example, if a plan is implemented as an RL policy, $next_i$ may be implemented as a lexical closure with a reference to the current state of the agent's environment, which simply returns the action specified by the policy in the current environment state; for "traditional" BDI plans, $next_i$ may, e.g., be implemented as a closure with a reference to the plan which maintains the index to the next step to be executed, which is updated on each call.

Intention Progression The *intention progression problem* [Logan *et al.*, 2017] for an agent with temporally extended goals is the problem of selecting and executing plans so as to maximise the agent's utility. More precisely, as the agent chooses and executes plans to achieve its goals, the execution of actions in these plans gives rise to a *path*, i.e., a sequence of environment states and actions $\rho = q_0, \alpha_1, q_1, \alpha_2, \ldots, \alpha_n, q_n$, where q_{i+1} results from updating the state q_i with the postconditions of α_i (and any exogenous changes to the environment). The *trace* corresponding to a path ρ is the sequence of states $\tau = q_0, q_1, \ldots, q_n$ generated by omitting the actions in ρ. The intention progression problem for an agent with temporally extended goals is to generate a trace that maximises the agent's utility.

In [Logan *et al.*, 2017], the intention progression problem is defined in terms of goal-plan trees. However, goal-plan trees are not an appropriate representation for the intentions of a BDI agent with temporally extended goals. For example, the temporally extended goal $F\phi_1 \wedge F\phi_2$ does not specify the order in which ϕ_1 and ϕ_2 should be achieved — the agent is free to achieve them in either order, depending on which is more convenient given its other goals, its location, etc. In general, representing all possible orders in which a temporally extended goal (and the temporally extended subgoals in the associated plans) can be achieved would require an exponentially-sized goal-plan tree.[5] Nor is it clear how to represent invariants such as $G\,\psi$ in a GPT-based approach, as GPTs have no way to talk about properties of executions of the GPT, only what needs to hold for a plan to be selected or an action executed. We therefore adopt a different representation of intentions for agents with temporally extended goals as explained in the next section.

3 Representing Intentions

In this section, we explain how the progression of an agent's temporally extended intentions can be represented using reward machines.

[5]Note that we can't simply consider each conjunct in a temporally extended goal as a top-level goal in a GPT-based approach. The utility is associated with the conjunction and there is no way to express that it's not worth achieving one top-level goal if it is not possible to achieve the other. In addition, "lifting" the conjuncts to top-level goals isn't possible for temporally extended subgoals in plans.

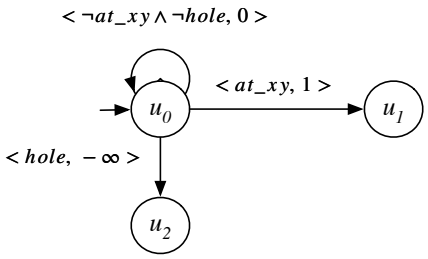

Figure 1: RM for moving to cell (x, y) while avoiding holes

3.1 Reward Machines

A *reward machine* (RM) [Toro Icarte *et al.*, 2018] is a Mealy machine where states represent abstract 'steps' or 'phases' in a task, and transitions correspond to observations of *high-level events* in the environment indicating that an abstract step/phase in the task has (or has not) been completed. We assume that events are sets of literals over $\bar{P}$. Formally, we require that events are generated by a *labelling function* $L : S \times Act \times S \to \wp(\bar{P})$; for example, a labelling could consist of postconditions of actions.

Definition 1. *A reward machine is a tuple* $R = (U, u_I, \Sigma, \delta_R, r_R)$ *where:*

- U *is a finite non-empty set of states;*
- u_I *is the initial state;*
- Σ *is a finite set of environment events;*
- $\delta_R : U \times \Sigma \to U$ *is a transition function that, for every state* $u \in U$ *and environment event* $\sigma \in \Sigma$*, gives the state resulting from observing event* σ *in state* u*; and*
- $r_R : U \times \Sigma \to \mathbb{R} \cup \{-\infty\}$ *is a reward function that for every state* $u \in U$ *and event* $\sigma \in \Sigma$ *gives the reward resulting from observing event* σ *in state* u*.*

We restrict our attention to *task completion* reward machines [Illanes *et al.*, 2020] which, on producing a positive reward, go into a final state from where it is not possible to make a transition to any other state. We denote by F_R the set of 'positive reward' states of R, that is, states u' such that for some u, σ, $\delta_R(u, \sigma) = u'$ and $r_R(u, \sigma) > 0$. Similarly, we define the set of 'infinite negative reward states' $N_R = \{u' \mid \exists u\, \exists \sigma . \delta_R(u, \sigma) = u' \wedge r_R(u, \sigma) = -\infty\}$. N_R states represent violations of invariant properties.

Reward machines can be automatically synthesised from task specifications expressed in a range of goal and property specification languages, including temporally extended goals defined in Section 2, using reactive synthesis techniques. For example, a Mars rover may have a goal to move to location (x, y) without falling into a hole, which can be expressed as the formula $\neg hole\, U\, at\_xy$. The corresponding reward machine is shown in Figure 1. The agent receives a positive reward when the event at_{xy} is observed and the reward machine transitions to state u_1, and a negative reward if it falls into a hole (state u_2). Depending on the magnitude of the negative reward and whether it is possible to return from the negative

reward state, invariants can be specified as preferences (avoid ψ states) or hard constraints which terminate the corresponding goal (as in Figure 1). In the interests of brevity, we refer the reader to [Camacho *et al.*, 2019] for details of reward machine generation.

Reward machines were originally proposed as a way of increasing sample efficiency in reinforcement learning. However, in our approach, they are used to formalise the progression of the agent's intentions.

3.2 Progressing Intentions

We represent an intention by a stack of intents. An *intent* is a five tuple of the form $(\phi, R, u, \sigma, next_\pi)$, where ϕ is a temporally extended goal, R is the reward machine corresponding to ϕ, u is the current state in R, σ is an environment event in Σ and $next_\pi$ is the next step function of a plan to achieve σ. The root of an intention γ_i is an intent $(\phi_i^0, R_i^0, u_i^0, \sigma_i^0, next_{\pi_i}^0)$ in which ϕ_i^0 corresponds to a top-level temporally extended goal in T. Initially u_i^0 is u_I in R_i^0, and σ_i^0, $next_{\pi_i}^0$ are ϵ.

Progressing an intention proceeds by first choosing a transition from u_i^0 with event (state goal) σ and updating σ_i^0, $\sigma_i^0 = \sigma$, and then choosing a plan π which makes σ true. That is, σ_i^0 represents a choice regarding which state goal in ϕ_i^0 the agent is currently pursuing, and $next_{\pi_i}^0 = next_\pi$ represents the plan chosen to achieve σ_i^0. The agent then begins executing steps in π. If the next step in π is a temporally extended subgoal ϕ_i^1, a new intent $(\phi_i^1, R_i^1, u_i^1, \sigma_i^1, next_{\pi_i}^1)$ is pushed onto the stack, and a choice is made regarding which event (state goal) in ϕ_i^1 to pursue and which plan should be used to achieve it. When the agent observes an event in $\sigma \in \Sigma_k^l$, for each intent in each intention γ_i where $\delta_{R_i}^j(u_i^j, \sigma) = u'$, the current state in the reward machine R_i^j is updated with $u_i^j = \delta_{R_i}^j(u_i^j, \sigma)$, and σ_i^j and $next_{\pi_i}^j$ are reset to ϵ. Updating all intents in all intentions on each observation allows the agent to exploit *synergies* between intentions [Yao *et al.*, 2016]. For example, when intentions share a common subgoal, e.g., being at a particular location, the states in all the relevant RMs are updated when the agent reaches the location, essentially marking the subgoal as achieved. Note, however, that if the intentions sharing the common sub-goal must be executed sequentially, e.g., due to resource constraints, the RMs of the deferred intentions will be updated when the agent leaves the location, essentially re-posting the subgoal for those intentions. When an event σ causes a reward machine R_i^j to transition to a state in $F_{R_i}^j \cup N_{R_i}^j$ (i.e., the temporally extended goal ϕ_i^j is true on the trace or an invariant in ϕ_i^j has been violated), the intent is popped from the intention and the agent receives reward $r_{\phi_i^j}$. Note that the current RM state, selected event and plan in the intent for the parent goal ϕ_i^{j-1} are also reset by σ, indicating completion of the subgoal. An intention containing no intents (i.e., all intents have been popped) is dropped.

3.3 Operational Semantics

In this section, we give the operational semantics of an agent with temporally extended goals in terms of a transition system on agent configurations defined by a set of derivation rules [Plotkin, 1981]. Each derivation transforms one agent configuration into another and corresponds to a single computation/execution step. An *agent configuration* is a tuple $\langle \Lambda, \Pi, B, \Gamma, \rho, p \rangle$, where Λ are the actions the agent can perform, Π is the agent's plan library, B is the agent's beliefs, Γ is the agent's intentions, ρ is a path consisting of alternating states and actions, and $p \in \{b, g, a\}$ is a phase indicator. (As Λ and Π do not change during execution, we omit them from the derivation rules.) Agent configurations represent the execution state of an agent. The agent begins execution in an *initial agent configuration* $\langle \Lambda, \Pi, B_0, \Gamma_0, \epsilon, b \rangle$, where B_0 are the agent's initial beliefs, $\Gamma_0 = \{[(\phi_1^0, R_1^0, u_{I_1}^0, \epsilon, \epsilon)], \ldots, [(\phi_n^0, R_n^0, u_{I_n}^0, \epsilon, \epsilon)]\}$ for each $\phi_i \in T$ are the agent's initial intentions, the path is empty, and the phase is b.

The execution of an agent modifies its initial configuration by means of transitions that are derivable from the rules below. Execution proceeds in phases. In the *belief* (b) phase, the agent updates its beliefs and the current state of each reward machine in each intention where the belief update event triggers a transition. In the *goal* (g) phase, any goals which have been achieved by the execution of a plan or by changes to the environment are dropped, together with their associated plans. In the *action* (a) phase, the agent executes a step in one of its intentions.

In what follows, we denote by $\gamma[i]$ the ith intent in the intention γ, and by $\gamma[0, \ldots, i]$ the intents from the root of γ to $\gamma[i]$. The functions $head$ and $tail$ return the head and tail of an intention respectively, and $\circ$ denotes concatenation. The head of an intention is termed an *active intent*, and an *active plan* is a plan in an active intent. The intents forming the tail of an intention are said to be *suspended*.

Belief Update In the *belief* phase, the agent's beliefs are updated. The function *sense* takes information about the current environment state env and the agent's current beliefs B, and returns a new set of beliefs, B'. The state of each intent in the agent's intentions is also updated with the new beliefs and execution enters the *goal* phase. Note that all intents are updated, not just active intents, as events perceived by the agent may result in transitions in suspended intents, e.g., if an environment change results in the achievement of a suspended goal.

$$\frac{\begin{array}{c} B' = sense(env, B) \wedge \\ \Gamma' = \Gamma\,[(\phi, R, u, \sigma, next_\pi)/(\phi, R, u', \epsilon, \epsilon)] \\ \forall (\phi, R, u, \sigma, next_\pi) : \delta_R(u, \sigma) = u' \wedge B' \models \sigma \end{array}}{\langle B, \Gamma, \rho, b \rangle \longrightarrow \langle B', \Gamma', \rho \circ B', g \rangle}$$

where $B' \models \sigma$ means that the propositional valuation specified by B' satisfies σ, and $\Gamma[x/y]$ is the result of replacing every intent of the form x in Γ by an intent of the form y.

Goal Update Any temporally extended goals ϕ which have become true on the trace or where an invariant in ϕ has been violated are dropped, together with their associated plans and any intents for subgoals and subplans to achieve ϕ. Execution enters the *action* phase.

$$\frac{\Gamma' = \{\gamma[0,\ldots,i] \mid \gamma \in \Gamma \wedge i = max\{i \mid \forall j : 0 \leq j < i \Rightarrow \gamma[j] \neq (\phi, R, u, \epsilon, \epsilon)\}\}}{\langle B, \Gamma, \rho, g\rangle \longrightarrow \langle B, \Gamma', \rho, a\rangle}$$

Goal Selection Step If an active intent of an intention γ for a temporally extended goal ϕ has no event and plan selected, an event (state goal) σ in the reward machine R_ϕ for ϕ and a plan π to achieve σ are selected. ι and π become the active intent and plan of γ, and execution enters the *belief* phase.

$$\frac{\begin{array}{c}\exists \gamma \in \Gamma : (\phi, R, u, \epsilon, \epsilon) = head(\gamma) \wedge \\ \exists \iota = (\phi, R, u, \sigma, next_\pi) : \sigma \in st(R, u) \wedge \\ \exists \pi = (\sigma, \chi, \pi) : B \models \chi\end{array}}{\langle B, \Gamma, \rho, a\rangle \longrightarrow \langle B, (\Gamma \setminus \{\gamma\}) \cup \{\iota \circ tail(\gamma)\}, \rho, b\rangle}$$

The function $st(R, u) = \{\sigma \mid \delta_R(u, \sigma) = u', u \neq u', r_R(u, \sigma) > 0\}$ returns the labels on transitions from u in R which are state goals.

Primitive Action Step If the next step in an active plan π is a primitive action α whose preconditions hold in the current environment, the action is executed, and execution enters the *belief* phase.

$$\frac{\begin{array}{c}\exists \gamma \in \Gamma : (\phi, R, u, \sigma, next_\pi) = head(\gamma) \wedge \alpha = next_\pi() \wedge \\ action(\alpha) \wedge B \models pre(\alpha)\end{array}}{\langle B, \Gamma, \rho, a\rangle \longrightarrow \langle B, \Gamma, \rho \circ \alpha, b\rangle}$$

Subgoal Step If the next step in an active plan π is a temporally extended subgoal ϕ' a new intent $\iota = (\phi', R_{\phi'}, u_I, \epsilon, \epsilon)$ for ϕ' is pushed onto the top of the intention γ containing π. Execution of π is suspended, and ι becomes the active intent of γ. Execution enters the *belief* phase.

$$\frac{\begin{array}{c}\exists \gamma \in \Gamma : (\phi, R, u, \sigma, next_\pi) = head(\gamma) \wedge \\ \phi' = next_\pi() \ \wedge goal(\phi')\end{array}}{\langle B, \Gamma, \rho, a\rangle \longrightarrow \langle B, (\Gamma \setminus \{\gamma\}) \cup \{(\phi', R_{\phi'}, u_I, \epsilon, \epsilon) \circ \gamma\}, \rho, b\rangle}$$

4 Intention Progression with MCTS

In this section, we present our approach to intention progression with temporally extended goals. Our approach is based on Single-Player Monte-Carlo Tree Search (SP-MCTS) [Schadd *et al.*, 2012], a well-known best-first search in which pseudorandom simulations are used to guide the expansion of the search tree. It was originally developed to solve single-player puzzles (games against the environment). However, it has also been used to solve single- and multi-agent intention progression problems, and represents the state of the art in GPT-based approaches to intention progression [Yao and Logan, 2016; Dann *et al.*, 2020].

MCTS is an anytime algorithm which iteratively builds a search tree until a computation budget is reached. Each node in the search tree represents an interleaving of steps from the agent's intentions, together with the environment state resulting from the execution of this interleaving. In addition, each node records the current state of each tuple in each intention, the number of times the node has been visited, the total simulation value, the total squared simulation value and the best simulation performed from the node (described below).

Algorithm 1 Return the step to be executed at this cycle

```
1:  C ← ⟨B, Γ, ρ, a⟩
2:  procedure PROGRESS-INTENTION(α, β)
3:      n_0 ← node_0(C)
4:      for i ← 1, α do
5:          n_e ← MAX-UCT-LEAF-NODE(n_0)
6:          children(n_e) ← EXPAND(n_e)
7:          n_s ← RANDOM-CHILD(children(n_e))
8:          S ← ∅
9:          for j ← 1, β do
10:             S ← S ∪ {SIMULATE(n_s)}
11:         v(n_s) ← max(S)
12:         BACKUP(n_s)
13:     n_b ← max children(n_0)
14:     return n_b
```

Edges represent the choice of an environment event to advance the state of a reward machine, i.e., which state goal in a temporally extended goal ϕ the agent is trying to achieve or the selection of a plan to achieve a state goal, or the execution of primitive action in a plan.[6]

Each iteration of the main loop consists of 4 phases: selection, expansion, simulation and back-propagation. In the *selection* phase, a leaf node, n_e is selected for expansion (line 5). n_e is selected using a modified version of Upper Confidence bounds applied to Trees (UCT) [Schadd *et al.*, 2012], which models the choice of node as a k-armed bandit problem. Starting from the root node, we recursively follow child nodes with the highest UCT value until a leaf node n_e is reached.

In the *expansion* phase, n_e is expanded by adding child nodes representing the agent configuration (the environment state and the current state of each intention in Γ) resulting from progressing each intention $\gamma_i \in \Gamma$ where a transition in the operational semantics is possible (line 6). Each child node therefore represents a different choice of which intention to progress at this cycle and how it should be progressed (which environment event to consider in an RM, or which plan to choose to achieve a state goal). One of the newly created child nodes, n_s, is then selected at random for simulation (line 7).

In the *simulation* phase, the value of n_s is estimated (lines 8–11). Starting in the agent configuration represented by n_s, a next step of an intention that is progressable in node n_s is randomly selected and executed, and the environment, current state of the selected intention and path are updated in the current agent configuration. This process is repeated until a state is reached in which no intention can be progressed or all top-level goals are achieved. Each temporally extended goal ϕ in each intent in Γ is evaluated on the trace τ (obtained by projecting on the state components of ρ) generated by the simulation. The value of the simulation, v, is the sum of the utilities r_ϕ for each temporally extended goal ϕ where $\tau \models \phi$ (including the negative utilities associated with any invariants

[6] Our approach is capable of interleaving intentions at either the plan level, where the choice of which intention to progress next is made only when a subgoal is reached, or the action level, where the choice is made at each plan step. [Yao and Logan, 2016].

violated on τ). The simulation with the highest value is then returned as the value of n_s. Finally, the simulation value is *back-propagated* from n_s to all nodes on the path to the root node n_0 (line 12).

After α iterations, the step that leads to the best child n_b of the root node n_0 is returned, and the current agent configuration C is updated based on the selected step. Algorithm 1 essentially resolves the nondeterministic choice in the *action* phase rules in the operational semantics.

5 Evaluation

In this section, we evaluate our approach using a version of the Mars Rover domain [Duff *et al.*, 2006], in which the tasks assigned to the Rover are specified using *temporally extended goals*.

5.1 Mars Rover Domain

The Mars Rover domain is a two-dimensional grid environment where the agent (the rover) can move around and conduct soil experiments at different locations (see Figure 5). Moving and performing experiments consumes battery power, and there is a base station at which the agent can recharge. Some cells in the environment contain holes, which may cause the agent to get stuck and should be avoided. The primitive actions the agent can perform are: *moveUp*, *moveDown*, *moveLeft* and *moveRight*, which change its current location, *performExp* which performs a soil experiment, and *recharge*. For simplicity, we assume all actions except *recharge* consume 1 unit of battery.

The agent has the following temporally extended goals;

$$G\,\neg batt\_empty$$

which states its battery should never be fully discharged; a set of goals of the form

$$F(exp\_x_i y_i \wedge F\,at\_base)$$

which states the agent should conduct a soil experiment at cell (x_i, y_i) and then return to the base; and

$$F\,batt\_full$$

which states that at some point the agent should recharge. The invariant $G\,\neg batt\_empty$ prevents the agent from reaching a state in which its battery is empty. The $F\ batt\_full$ goal ensures that the agent will recharge at some point. The only way to satisfy both goals is to ensure that the recharge happens before the battery is empty (i.e., while the agent can still reach the charging station).

The plan $\pi_{exp\_x_i y_i}$ to achieve the state goal $exp\_x_i y_i$ consists of the temporally extended subgoal $\neg hole\,U\,at\_x_i y_i$ stating that the rover should move to cell (x_i, y_i) while avoiding holes, followed by the action *performExp*. The plan to achieve the state goal $batt\_full$ consists of the subgoal $\neg hole\,U\,at\_base$ followed by the action *recharge*. The plans to achieve the state goals $at\_x_i y_i$ and at_base contain no subgoals. Note that a set of goals of the form $\{F(exp\_x_1 y_1 \wedge F\,at\_base), F(exp\_x_2 y_2 \wedge F\,at\_base), \ldots\}$ allows the agent to perform experiments in any order, and it will return to base after the last experiment, even if it visits the base to recharge before completing all the experiments. Note also that the agent does not have to return to the base after each experiment—each goal simply specifies that the agent should return to the base at some future point; if the agent can perform all experiments without recharging, then returning to the base once satisfies all goals.

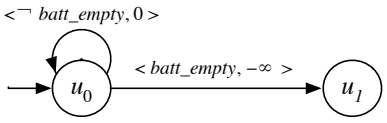

Figure 2: RM for maintaining battery charge

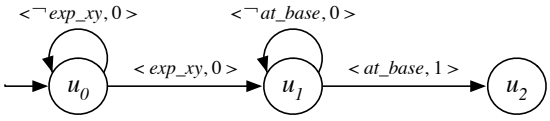

Figure 3: RM for conducting a soil experiment at cell (x, y) and then returning to base

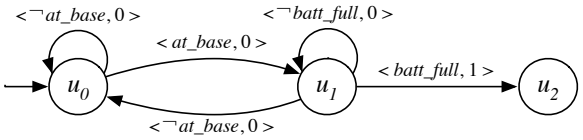

Figure 4: RM to return to the base and recharge

The reward machine for the goal $G\,\neg batt\_empty$ is shown in Figure 2: so long as the event $batt\_empty$ is not observed, the RM remains in state u_0 and the agent receives no reward; the agent receives a negative reward if $batt\_empty$ is observed and the RM transitions to state u_1. The reward machine for the goal $F(exp\_x_i y_i \wedge F\,at\_base)$ is shown in Figure 3: the RM loops in state u_0 until an event indicating the experiment has been performed is observed, and then loops in u_1 until the agent returns to the base. The reward machine for the goal $\neg hole\,U\,at\_x_i y_i$ is shown in Figure 1 in Section 3 (the RM for $\neg hole\,U\,at\_base$ is similar). The RM for the goal $F\,batt\_full$ is shown in Figure 4: the agent receives a reward if the events at_base and $batt\_full$ are observed, in that order; if the agent leaves the base before recharging, the RM transitions back to u_0.

5.2 Experiment Setup

For all the experiments reported below, the Mars Rover environment is a 21 × 21 grid containing 10 holes.[7] The base and the initial position of the agent are the centre of the grid. The locations for conducting soil experiments and the holes are randomly generated. However, the base and the experiment locations are not located in a hole. The battery capacity for the rover is 50 units. The performance of the agent is based on the total reward specified by the reward machines. That is, the agent will receive a reward of negative infinity if it enters

[7]The software used in the experiments is freely available at https://github.com/yvy714/TEG-MCTS.

a grid cell with a hole or runs out of battery, otherwise, its performance is equivalent to the number of experiments carried out. For simplicity, we assume that the agent recharges once, i.e., the agent is given a single recharge goal.

We provided both human-authored plans and plans based on RL policies for state goals $at\_x_iy_i$ and at_base. The human-authored plans are sequences of actions forming the shortest path between each x_iy_i, and the base. The RL policy is trained to find a shortest path to the goal. None of the plans take the location of holes into account. As plans require moving to a particular location in the environment, the MCTS scheduler was configured to interleave intentions at the plan level. In each deliberation cycle, the scheduler performs 100 iterations ($\alpha = 100$) and 10 simulations per iteration ($\beta = 10$).

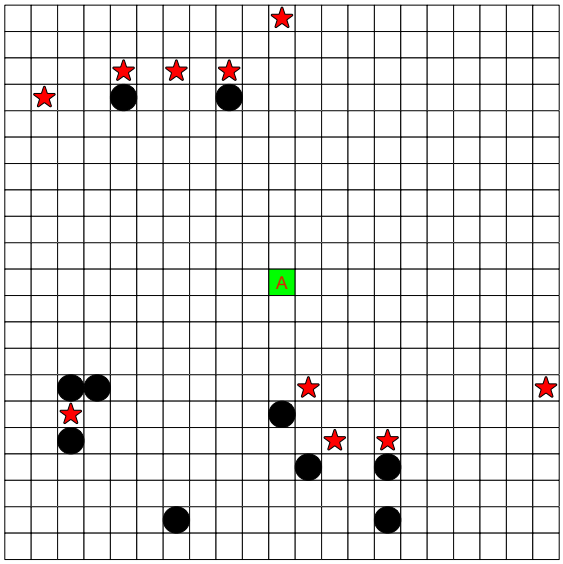

Figure 5: The Mars Rover domain; black circles represent holes and stars represent soil experiments

5.3 Results

In each run, the agent has a goal to avoid the battery becoming discharged, a goal to recharge the battery (at some point), and a variable number of soil experiment goals. We varied the number of soil experiment goals (# Goals) from 8 to 15 and report the average performance over 50 runs. The results are shown in Table 1. As can be seen, with fewer than 11 goals the agent is able to achieve all goals without violating the invariant. When the number of top-level goals increases, there may be no solution which achieves all top-level goals in some of the randomly generated environments, e.g., the location of a soil experiment is more likely to be affected by the holes, and thus one recharge is not enough to achieve all goals. However, our approach can still achieve more than 96% of the goals.

The computational overhead of our approach depends on the search configuration, i.e., how many iterations should be performed at each deliberation cycle, and how many simulations are performed at each iteration. With the search configuration used for the experiments above, i.e., α = 100, β = 10, our prototype implementation requires about 50 milliseconds to return the best next plan for 15 concurrent intentions on a 1.4 GHz Quad-Core Intel Core i5, which suggests that the computational overhead compared to GPT-based approaches, e.g., *SA* [Yao and Logan, 2016] is small.

#G	8	9	10	11	12	13	14	15
Reward	8	9	10	11	11.9	12.75	13.6	14.5

Table 1: Rewards with increasing number of goals

6 Related Work

Temporally extended goals were introduced in the Procedural Reasoning System (PRS) [Georgeff and Lansky, 1987], and in much of the early work on BDI logics [Rao and Georgeff, 1993] goals are viewed as temporal formulas. PRS used a graph-based representation for plans, and goals were defined on an interval of time: $(!p)$ specifies that p should be true at some future state; $(?p)$ that p should be true in the next state; and $(\#p)$ specifies that p should hold in all states. Later versions of PRS [Ingrand, 1991] supported achievement goals, and two forms of maintenance goal: '*passive preserve*' $\mathrm{PR}_p(!e, \phi)$ which specifies that e should be achieved while monitoring ϕ and aborted if e fails or ϕ does not hold; and '*active preserve*' $\mathrm{PR}_a(!e, \phi)$, which is similar, except that if ϕ does not hold, the plan for e is suspended while the agent attempts to re-establish ϕ, and only if this fails, e is aborted. However PRS is a rich and complex language which is difficult to reason about, and it is only recently that an operational semantics was proposed [de Silva *et al.*, 2018]. Compared to [Georgeff and Lansky, 1987], our approach supports $(!p)$ and $(\#p)$ goals, but not $(?p)$ goals. Compared to [Ingrand, 1991], our approach supports achievement and passive preserve goals, the nesting of temporal operators in goal specifications (which neither [Georgeff and Lansky, 1987] or [Ingrand, 1991] support), and the semantics we present in Section 3.3 is considerably simpler than that given in [de Silva *et al.*, 2018]. Moreover, the developer of a PRS program is responsible for anticipating and writing code (meta-plans) to manage interactions between the agent's intentions, and the language does not support plans implemented using RL policies.

Much of the later work in the BDI paradigm focused on simpler languages, e.g., [Rao, 1996]. There has been considerable work on intention progression for such languages, mostly for agents with achievement goals. Thangarajah et al. [2003a; 2003b; 2002; 2011] proposed a summary-information-based (SI) approach which avoids conflicts and exploits synergies between intentions by reasoning about the conditions that will definitely or possibly be brought about by each top-level goal. Waters et al. [2014; 2015] proposed a coverage-based (CB) approach that prioritises the progression of intentions that are most likely to be affected by the changes in the environment. Yao *et al.* [2014; 2016; 2016; 2020; 2021] proposed an MCTS-based approach which returns the "best" action to perform at each deliberation cycle. They showed that *SA* [Yao and Logan, 2016] out-performed round-

robin (RR), first-in-first-out (FIFO), SI, and CB intention progression in both static and dynamic environments. There has also been some work on intention progression for agents with *active preserve* maintenance goals [Duff *et al.*, 2006; Wu *et al.*, 2023]. However, in this work maintenance goals are restricted to top-level goals, i.e., they do not support temporally extended subgoals in plans.

In most of the work on autonomous intention progression, the agent's program is represented using goal-plan trees (see Section 2). In [Dann *et al.*, 2022] reward machines are used to represent the high-level structure of *other agents'* achievement goals in 'intention-aware' intention scheduling, i.e., where an agent progresses its intentions taking into account the implications of action scheduling for both its own goals and the goals of other agents. However, in [Dann *et al.*, 2022] the other agents' goals are simple achievement goals and GPTs are used to represent the program of the 'intention-aware' agent.

There has also been some work on integrating learned behaviours and learning into BDI agents. For example, Tan *et al.* [2011], propose the BDI-FALCON architecture which extends a low-level TD-FALCON reinforcement learning agent with desire and intention modules. A BDI-FALCON agent uses explicit goals instead of an external reward signal, and is able to reason about a course of action using the intention module and a plan library. In the absence of a plan, the agent learns a new plan using the FALCON module. In [Singh and Hindriks, 2013], reinforcement learning was integrated into the GOAL agent programming language to improve action selection. Bosello and Ricci [2020] address the problem of how to exploit reinforcement learning in the process of developing an agent and how to integrate plans written by the agent programmer with plans designed to be learnt. They propose a proof of concept programming language, Jason(RL), where learning is integrated into the deliberation cycle. Finally, Al Shukairi and Cardoso [2023] present a hybrid BDI-NN architecture for controlling a vehicle in the CARLA simulation environment where interactions between the agent's collision avoidance plans are pre-specified by the developer. However, in all this work, the agent's goals are limited to simple achievement goals.

Dastani *et al.* [2011] present a rich taxonomy of goal types, where the goals are represented by LTL formulas which are translated into combinations of more basic declarative and maintenance goals, and sketch how such temporal goals can be realised in standard agent programming languages. Hindriks et al. [Hindriks *et al.*, 2009] discuss how temporally extended goals expressed in LTL can be integrated into the language Goal, and sketch a decision procedure based on bounded lookahead for choosing between programmer-specified enabled actions.

7 Discussion

In this paper, we presented a new approach to intention progression for agents with *temporally extended goals*. Temporally extended goals allow mixing reachability and invariant properties, e.g., "move to location A without falling into a hole", and may be specified at run-time (top-level goals) and as subgoals in plans. The introduction of invariants significantly increases the expressiveness of BDI agent programming by allowing constraints to be placed on a particular execution of the agent's plans. For example, a standard movement plan may be used in an environment with holes, rather than writing a new plan that avoids holes. In contrast, with GPTs, the agent's plans would have to consider all possible constraints on execution, e.g., "move to location A while avoiding holes / maintaining line of sight to the base / etc.". To the best of our knowledge, our approach is the first to allow an agent to autonomously progress its intentions to achieve multiple temporally extended goals concurrently.

In addition, our approach allows human-authored plans and plans implemented as reinforcement learning policies to be freely mixed in an agent program, allowing the development of agents with 'neuro-symbolic' architectures which combine behaviours produced by machine learning and programmed behaviours [Bordini *et al.*, 2020]. One of the main motivations for introducing support for plans implemented as RL policies is to allow temporally extended goals and the BDI approach in general to be applied to cyber-physical systems (CPS). We believe BDI has seen limited application in CPS due to the difficulty of writing BDI plans for non-deterministic environments, as the sequential ordering of actions typically assumes that action preconditions have been established by previous actions in the plan. RL policies have been shown to be highly effective in such environments, and we envisage "hybrid" systems, where low-level control is handled by plans implemented as policies, while higher-level decision making is handled by human-authored plans.

Acknowledgements

This work was supported in part by the Yongjiang Talent Introduction Program Young Talent (2022A-234-G).

References

[Al Shukairi and Cardoso, 2023] Hilal Al Shukairi and Rafael C. Cardoso. ML-MAS: A hybrid AI framework for self-driving vehicles. In *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2023*, pages 1191–1199. ACM, 2023.

[Bordini *et al.*, 2007] Rafael H. Bordini, Jomi Fred Hübner, and Michael Wooldridge. *Programming multi-agent systems in AgentSpeak using Jason*. Wiley Series in Agent Technology. Wiley, 2007.

[Bordini *et al.*, 2020] Rafael H. Bordini, Amal El Fallah Seghrouchni, Koen Hindriks, Brian Logan, and Alessandro Ricci. Agent programming in the cognitive era. *Autonomous Agents and Multi-Agent Systems*, 34(37), 2020.

[Bosello and Ricci, 2020] Michael Bosello and Alessandro Ricci. From programming agents to educating agents - A jason-based framework for integrating learning in the development of cognitive agents. In Louise A. Dennis, Rafael H. Bordini, and Yves Lespérance, editors, *Engineering Multi-Agent Systems - 7th International Work-*

shop, EMAS 2019, volume 12058 of *Lecture Notes in Computer Science*, pages 175–194. Springer, 2020.

[Camacho *et al.*, 2019] Alberto Camacho, Rodrigo Toro Icarte, Toryn Q. Klassen, Richard Valenzano, and Sheila A. McIlraith. LTL and beyond: Formal languages for reward function specification in reinforcement learning. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI 2019)*, pages 6065–6073. IJCAI, 2019.

[Dann *et al.*, 2020] Michael Dann, John Thangarajah, Yuan Yao, and Brian Logan. Intention-aware multiagent scheduling. In *Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2020)*, pages 285–293. IFAAMAS, 2020.

[Dann *et al.*, 2022] Michael Dann, Yuan Yao, Brian Logan, and John Thangarajah. Multi-agent intention progression with reward machines. In Luc De Raedt, editor, *Proceedings of the 31st International Joint Conference on Artificial Intelligence and the 25th European Conference on Artificial Intelligence (IJCAI-ECAI 2022)*, pages 215–222, Vienna, Austria, July 2022. IJCAI.

[Dastani *et al.*, 2011] Mehdi Dastani, M. Birna van Riemsdijk, and Michael Winikoff. Rich goal types in agent programming. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, pages 405–412, 2011.

[Dastani, 2008] Mehdi Dastani. 2APL: a practical agent programming language. *Autonomous Agents and Multi-Agent Systems*, 16(3):214–248, 2008.

[de Silva *et al.*, 2009] Lavindra de Silva, Sebastian Sardina, and Lin Padgham. First Principles Planning in BDI systems. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-09)*, pages 1105–1112, 2009.

[de Silva *et al.*, 2018] Lavindra de Silva, Felipe Meneguzzi, and Brian Logan. An operational semantics for a fragment of PRS. In Jérôme Lang, editor, *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI 2018)*, pages 195–202, Stockholm, Sweden, July 2018. IJCAI.

[de Silva *et al.*, 2020] Lavindra de Silva, Felipe Meneguzzi, and Brian Logan. BDI agent architectures: A survey. In *Proceedings of the 29th International Joint Conference on Artificial Intelligence (IJCAI 2020)*. IJCAI, 2020.

[Duff *et al.*, 2006] Simon Duff, James Harland, and John Thangarajah. On proactivity and maintenance goals. In *5th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2006), Hakodate, Japan, May 8-12, 2006*, pages 1033–1040. ACM, 2006.

[Georgeff and Lansky, 1987] Michael P. Georgeff and Amy L. Lansky. Reactive reasoning and planning. In *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI-87)*, pages 677–682, 1987.

[Hindriks *et al.*, 2009] Koen V. Hindriks, Wiebe van der Hoek, and M. Birna van Riemsdijk. Agent programming with temporally extended goals. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 137–144, 2009.

[Illanes *et al.*, 2020] León Illanes, Xi Yan, Rodrigo Toro Icarte, and Sheila A McIlraith. Symbolic plans as high-level instructions for reinforcement learning. In *Proceedings of the Thirtieth International Conference on Automated Planning and Scheduling (ICAPS 2020)*, pages 540–550. AAAI Press, 2020.

[Ingrand, 1991] Félix Ingrand. *OPRS Development Environment Version 1.1b7*, 1991. Last updated January 2014. https://git.openrobots.org/projects/openprs.

[Logan *et al.*, 2017] Brian Logan, John Thangarajah, and Neil Yorke-Smith. Progressing intention progresson: A call for a goal-plan tree contest. In *Proceedings of the 16th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2017)*, pages 768–772. IFAAMAS, 2017.

[Plotkin, 1981] Gordon D. Plotkin. A structural approach to operational semantics. Technical report, University of Aarhus, 1981.

[Rao and Georgeff, 1992] Anand S. Rao and Michael P. Georgeff. An abstract architecture for rational agents. In *Proceedings of Knowledge Representation and Reasoning (KR&R-92)*, pages 439–449, 1992.

[Rao and Georgeff, 1993] Anand S. Rao and Michael P. Georgeff. A model-theoretic approach to the verification of situated reasoning systems. In Ruzena Bajcsy, editor, *Proceedings of the 13th International Joint Conference on Artificial Intelligence. Chambéry, France, August 28 - September 3, 1993*, pages 318–324. Morgan Kaufmann, 1993.

[Rao, 1996] Anand S. Rao. Agentspeak(l): BDI agents speak out in a logical computable language. In *MAAMAW'96: Proceedings of the 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World: Agents Breaking Away*, pages 42–55, Secaucus, NJ, USA, 1996. Springer-Verlag New York, Inc.

[Schadd *et al.*, 2012] Maarten P. D. Schadd, Mark H. M. Winands, Mandy J. W. Tak, and Jos W. H. M. Uiterwijk. Single-player Monte-Carlo tree search for SameGame. *Knowledge-Based Systems*, 34:3–11, 2012.

[Singh and Hindriks, 2013] Dhirendra Singh and Koen V. Hindriks. Learning to improve agent behaviours in GOAL. In Mehdi Dastani, Jomi Fred Hübner, and Brian Logan, editors, *Programming Multi-Agent Systems - 10th International Workshop, ProMAS 2012*, volume 7837 of *Lecture Notes in Computer Science*, pages 158–173. Springer, 2013.

[Tan *et al.*, 2011] Ah-Hwee Tan, Yew-Soon Ong, and Akejariyawong Tapanuj. A hybrid agent architecture integrating desire, intention and reinforcement learning. *Expert Syst. Appl.*, 38(7):8477–8487, 2011.

[Thangarajah and Padgham, 2011] John Thangarajah and Lin Padgham. Computationally effective reasoning about

goal interactions. *Journal of Automated Reasoning*, 47(1):17–56, 2011.

[Thangarajah *et al.*, 2002] John Thangarajah, Michael Winikoff, Lin Padgham, and Klaus Fischer. Avoiding resource conflicts in intelligent agents. In *Proceedings of the 15th European Conference on Artificial Intelligence (ECAI 2002)*, pages 18–22, 2002.

[Thangarajah *et al.*, 2003a] John Thangarajah, Lin Padgham, and Michael Winikoff. Detecting & avoiding interference between goals in intelligent agents. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI-03)*, pages 721–726. Morgan Kaufmann, 2003.

[Thangarajah *et al.*, 2003b] John Thangarajah, Lin Padgham, and Michael Winikoff. Detecting & exploiting positive goal interaction inintelligent agents. In *Proceedings of the Second International Joint Conference on Autonomous Agents & Multiagent Systems (AAMAS 2003)*, pages 401–408, 2003.

[Toro Icarte *et al.*, 2018] Rodrigo Toro Icarte, Toryn Klassen, Richard Valenzano, and Sheila McIlraith. Using reward machines for high-level task specification and decomposition in reinforcement learning. In *Proceedings of the 35th International Conference on Machine Learning (ICML 2018)*, pages 2107–2116. PMLR, 2018.

[Waters *et al.*, 2014] Max Waters, Lin Padgham, and Sebastian Sardiña. Evaluating coverage based intention selection. In *Proceedings of the 13th International Conference on Autonomous Agents and Multi-agent Systems (AAMAS 2014)*, pages 957–964. IFAAMAS, 2014.

[Waters *et al.*, 2015] Max Waters, Lin Padgham, and Sebastian Sardiña. Improving domain-independent intention selection in BDI systems. *Autonomous Agents and Multi-Agent Systems*, 29(4):683–717, 2015.

[Wu *et al.*, 2023] Di Wu, Yuan Yao, Natasha Alechina, Brian Logan, and John Thangarajah. Intention progression with maintenance goals. In A. Ricci, W. Yeoh, N. Agmon, and B. An, editors, *Proceedings of the 22nd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2023)*, pages 2400–2402, London, May 2023. IFAAMAS, IFAAMAS.

[Yao and Logan, 2016] Yuan Yao and Brian Logan. Action-level intention selection for BDI agents. In *Proceedings of the 15th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2016)*, pages 1227–1236. IFAAMAS, 2016.

[Yao *et al.*, 2014] Yuan Yao, Brian Logan, and John Thangarajah. SP-MCTS-based intention scheduling for BDI agents. In *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI-2014)*, pages 1133–1134. ECCAI, IOS Press, 2014.

[Yao *et al.*, 2016] Yuan Yao, Brian Logan, and John Thangarajah. Robust execution of BDI agent programs by exploiting synergies between intentions. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI-16)*, pages 2558–2565. AAAI Press, 2016.

[Yao *et al.*, 2020] Yuan Yao, Natasha Alechina, Brian Logan, and John Thangarajah. Intention progression under uncertainty. In Christian Bessiere, editor, *Proceedings of the 29th International Joint Conference on Artificial Intelligence (IJCAI 2020)*, Yokohama, Japan, July 2020. IJCAI.

[Yao *et al.*, 2021] Yuan Yao, Natasha Alechina, Brian Logan, and John Thangarajah. Intention progression using quantitative summary information. In Ulle Endriss, Ann Nowé, Frank Dignum, and Alessio Lomuscio, editors, *Proceedings of the 20th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2021)*, pages 1416–1424, London, May 2021. IFAAMAS, IFAAMAS.