Goal Analytics via Model Checking for Autonomous Systems

Jin Song Dong[1,2], **Naipeng Dong**[1], **Zhé Hóu**[2], **Brendan Mahony**[3], **Martin Oxenham**[3]
j.dong@griffith.edu.au, dcsdn@nus.edu.sg, z.hou@griffith.edu.au,
{brendan.mahony, martin.oxenham}@dst.defence.gov.au
[1]National University of Singapore, [2]Griffith University, [3]DST Australia

Abstract

Analogous to planning via model checking, we propose to apply model checking on goal analytics of self-regulated agents in autonomous systems. Our project involves investigating how to integrate goal reasoning techniques into the Process Analysis Toolkit (PAT) model checking framework and evaluating which model checking algorithms are efficient for complex and concurrent systems with non-determinism. This paper presents the initial research progress on a new Australian Defence Science and Technology (DST) Group funded project on goal analysis for autonomous systems.

1 Introduction

Many existing autonomous systems rely on controlled environments in order to function reliably. However, in typical everyday situations, such control cannot be guaranteed. As such, to broaden the range of future applications for autonomous systems, techniques need to be developed to ensure that the autonomous systems can be trusted to function safely and effectively even in the face of unexpected situations which may arise. This will require the agents underpinning the autonomous systems to be able to reason about their goals and to plan tasks to achieve those goals in an agile manner.

We propose to apply model checking techniques to goal reasoning and dynamic planning for autonomous agents. We take planning via model checking as our foundation. That is, given a goal, model checking assists with reasoning about all possible action sequences that can lead to the goal. In addition, model checking can also calculate the cost and reward of each action sequence, which assists decision making when finding the optimal plan. Furthermore, we propose to extend the planning procedure with goal reasoning via model checking by generating and selecting goals and updating the task hierarchy accordingly.

Motivation and Background. The behaviour of an intelligent agent is often pre-defined in explicit detail in existing autonomous systems. However, in the real-world, the agent may have to operate in an uncertain environment which is dynamic and only partially observable. Therefore, pre-defined behaviour or even online guided behaviour is not sufficient to handle unanticipated situations in such environments. External events often require postponing or even abandoning the current goal. This often occurs in Defence operations and activities to provide humanitarian assistance and support disaster relief. For instance, self-driving vehicles that survey natural disaster areas must be able to adjust their goals without human interaction. In such settings, domain modelling is impractical because each disaster situation is distinct. The self-driving vehicle must be able to adjust its goals when, e.g., a survivor is found. Despite that many self-driving vehicles are being developed, tested and tentatively applied in many countries, few are able to adjust their goals autonomously.

Most existing planning approaches rely on static goals. That is, goals are specified at the beginning of the planning process and they are not changed during the course of the process. In contrast, enabling agents to respond to uncertain and dynamic environments autonomously requires active goal reasoning and planning. (Active) Goal reasoning of autonomous agents has become a hot research topic recently. In the literature, *goal analytics and reasoning* are used to recognise when the world diverges from what is planned, to understand various hypotheses that interpret these changes of circumstance, and to reason about what goals to pursue in response to the changes. There have been formalisms which define goals, tasks and the procedure of recognising and formulating new goals [Alford *et al.*, 2016]. However, most of them are manual. Notable exceptions which designed algorithms to generate new goals suffer from the scalability problem.

Model checking is a technology to automatically verify whether certain properties are satisfied by a model using exhaustive search. Model checking is especially strong in addressing uncertain and concurrent behaviours. This technology has been successfully applied to modelling and verifying uncertain environments, such as a network attacker which may perform arbitrary behaviour in communication protocols. More importantly, model checking has been successfully applied to automatic planning in complex systems in various domains [Li *et al.*, 2014].

As a concrete application, we plan to deploy the model checking based planning and goal reasoning framework on Autonomous Underwater Vehicles (AUVs). Our approach will provide a higher level of automation for the AUV sys-

tem so that the vehicle can smartly select its goals and plan tasks automatically, and run with minimal human interaction.

Our approach. One approach to address the goal reasoning aspects of goal-driven autonomy that has emerged in the last few years is based on goal hierarchies in which goals are specified in terms of sub-goals and parent-goals in a hierarchical fashion. In the planning community, planners based on so-called Hierarchical Task Networks (HTNs) have enjoyed their fair share of success [Alford *et al.*, 2016]. In HTNs, a complete specification of tasks is usually required beforehand. Recently, an alternative concept in the form of Hierarchical Goal Networks (HGNs) has been proposed which replaces tasks with goals in the planning hierarchy. Our approach will be based on a new framework which is a combination of HTN and HGN: Goal Task Networks (GTNs) [Alford *et al.*, 2016]. In GTNs, hierarchies specify the goals (resp. tasks) and sub-goals (resp. sub-tasks) that need to be performed, high-level goals (resp. tasks) are reified in terms of sub-goals (resp. sub-tasks) in a goal/task tree. This offers three advantages: multiple methods for achieving a goal can be applied, planning with incomplete models is supported, and the representation of goals in the goal network is easier to reason about. These properties make GTNs a useful method for supporting goal-driven autonomy.

The goal reasoning literature focuses on the procedure or architecture of goal reasoning rather than on how to reason about the goals in an automatic manner. Since model checking has proven useful in solving planning problems [Sun *et al.*, 2013; Li *et al.*, 2014], and goal reasoning is closely related to planning [Cox *et al.*, 2016; Alford *et al.*, 2016], it is natural to adapt the techniques in the previous work and use model checking to solve goal reasoning problems. In this approach, the translation from Planning Domain Definition Language (PDDL) to Communicating Sequential Process # (CSP#) is important, because the former language can be used to express goal reasoning problems [Wilson *et al.*, 2016], and the latter language is recognised by existing model checkers [Sun *et al.*, 2009].

We propose to extend CSP# [Li *et al.*, 2012] with new features à la [Sun *et al.*, 2013] for goal reasoning. We translate GTNs to the extended CSP# language, and use model checking to obtain plans and goals in this framework. This framework allows a much richer GTN environment in which tasks and goals can be combined using the full power of the extended CSP# process algebra. We then model the goal hierarchy using logical relations between sub-goals in the model checking framework, which is used to perform goal reasoning. Finally, we demonstrate how temporal properties such as the duration of actions can be added in the CSP# language, this results in a more expressive framework which is useful in modelling the actions of unmanned vehicles.

2 Related Work

Effective decision-making is key to agents that can intelligently react to unforeseen situations. A fundamental technique is *planning* – building a plan for achieving a given goal on a given dynamic domain. Different approaches exist according to the assumptions about the domain, the goals, the plans and the planning algorithm. Conceptually, the domain evolves according to the performed actions, a controller provides the actions according to the observations on the domain and a plan [Ghallab *et al.*, 2016]. An example of applying automated reasoning techniques on planning is Kress-Gazit et al.'s framework which automatically translates high-level tasks defined in linear temporal logic formulae to hybrid controllers [Kress-Gazit *et al.*, 2009]. This framework allows for reactive tasks, which may change depending on the information the robot gathers at runtime. This is similar to the goal reasoning literature where goals may change depending on the environment at runtime.

As stated in the introduction, when unexpected events happen in the domain, the agent's goal may be affected and thus selecting a new goal and re-planning are necessary. This generally follows a *note-assess-guide* procedure, where *note* is detecting discrepancies (e.g., [Paisner *et al.*, 2013]), *assess* is hypothesizing causes for discrepancies, and *guide* performs a suitable response [Anderson *et al.*, 2006]. Differing from classical planning where the goal is fixed, when a discrepancy is detected, it is often necessary to change the current goal. Goal reasoning is about scheduling a suitable goal for the planning process.

Goal reasoning has been used in a number of projects about controlling autonomous machines in a dynamic environment. [Cox *et al.*, 2016] propose to use classical planning to formalise goal reasoning. They present an architecture with a cognitive layer and a metacognitive layer to model problem-solving and dynamic event management in self-regulated autonomy. The architecture is realised in the Metacognitive Integrated Dual-Cycle Architecture 1.3, which is shown useful in experiment. Consider a *classical planning problem* $P = (\Sigma, s_0, g)$, where Σ is a state transition system, s_0 is the initial state, and g is a set of conditions that the plan has to satisfy. A *plan* π is a sequence of actions $\langle \alpha_1, \cdots, \alpha_n \rangle$ where each action α_i effects a transition from the current state to the next state. A successful plan is thus one that goes from s_0 and eventually transitions to a final state $s_g \vDash g$ that satisfies all the conditions in g. To model the dynamic change in goals, they use a function $\beta(s, g) \rightarrow g'$ to obtain a new goal from the current state and goal. Therefore, in addition to planning, a goal reasoning system formulates new goals at runtime and creates new plans to achieve these goals.

[Roberts *et al.*, 2015] give more detailed definitions of goal reasoning in their framework. They divide the states and goals into two parts: the external part is a modified or incomplete version of the transition system, and the internal part represents the predicates and state required for the refinement strategies. The authors use a data structure called *goal memory* to represent the relationship between goals, subgoals, parent goals etc., and propose to solve the goal reasoning problem using refinement. They use a *goal lifecycle* model to capture the evolution of goals and the decision points involved in the process. The goal lifecycle includes the formulation, selection, expansion, execution, dispatch, evaluation, termination, and discard of goals. This model is adapted by [Johnson *et al.*, 2016], who give a system called Goal Reasoning with Information Measures. In the scenario of controlling

Unmanned Air Vehicles to survey certain areas, the goals are formulated with parameters such as *maximum uncertainty* in the search area, *acceptable uncertainty* under which the goal is considered complete, and *deadline* by which the search must complete. The goal reasoning method is shown useful for unmanned aerial vehicles operating in dynamic environments.

A more theoretical foundation about planning and goal reasoning is surveyed by [Alford *et al.*, 2016]. The authors unify HGN planning and HTN planning into GTN planning. They also provide plan-preserving translations from GTN problems to HTN semantics. Several computability and tractability results are given. For example, GTN, HTN, and HGN are semi-decidable, and a restricted form called GTN_I is NEXPTIME.

An important application of our project is applying the planning and goal reasoning framework to AUVs. A relevant work in this aspect is goal reasoning for AUVs [Wilson *et al.*, 2016]. The authors use a goal-driven autonomy conceptual model which has three parts: the planner, the goal controller, and the state transition system. The goal reasoning problem is formalised in PDDL, which is the standard language for representing classical planning problems and is widely used by many planners. The authors test their approach in simulations where the AUV surveys a defined area and it has to respond (change the goal) to the actions from a nearby unmanned surface vehicle dynamically.

3 Challenges and Proposed Solutions

There are three main challenges in utilizing model checking to perform goal reasoning.

First, how to specify tasks, goals and their relations? As mentioned earlier, a given problem can be specified as goals/tasks and sub-goals/sub-tasks in GTNs. However, in GTNs, the goal may be changed as a consequence of environmental events, and it cannot be a fixed specification any more. In addition, there may be multiple sequences of actions that can achieve the same goal. This further introduces complexity of the tasks and goal specifications. Moreover, a high-level goal may contain several sub-goals that may need to be achieved in (partial) order or with constraints.

Second, how to model the evolution of plans and goals? As consequences of the dynamic environment and nondeterminism, there may be multiple task plans to achieve one goal, and there may be multiple goals to replace an old goal when exogenous events break the current plan. How to figure out the optimal plan and goal is not yet well-studied in the literature, especially when the evaluation may require reasoning about unexpected events and future events (i.e., temporal evaluation) for planning and goal selection.

Third, how to make the goal reasoning system scale? In the literature, the work on goal driven autonomy is mostly theoretical, and the algorithms suffer from scalability problems. This prevents goal driven autonomy from being widely applied in the real-world.

To address the challenges, analogous to classical planning via model checking, we propose to use model checking, in particular, the model checker PAT [Sun *et al.*, 2009] to perform goal reasoning. PAT is a self-contained framework that supports composing, simulating and reasoning about concurrent, probabilistic and timed systems with non-deterministic behaviours. It comes with user friendly interfaces, a featured model editor and an animated simulator. PAT implements various model checking techniques which cater for different properties (reachability, linear temporal logic and fairness) and facilitate new language and algorithm design and implementation.

The first challenge is going to be addressed by proposing a specification framework for GTN based on PAT. The basic idea is to model the environment as a set of variables and events (CSP# supports complex data structures) and model actions as symbolic events in CSP# (input language for PAT), and formalising the goals as assertions/properties. In this way, given an initial environment (a state in the model), model checking automatically finds all possible plans to achieve the goal (the traces satisfying the assertions). That is, we relate the enabled plans to their corresponding goals. The relations between goals are captured by the logical relations between assertions. Similarly, we can apply model checking to identify the environment (the states in the model) of the agent when an unanticipated event happens, by searching the transitions labelled by the unanticipated event. With the possible new environments, model checking then can figure out the possible new goals and the task sequences to achieve each new goal.

For the second challenge, the evaluation of plans and goals can be addressed by extending the specification of tasks and goals with costs and rewards, similar to the extension with probability and time in PAT. Therefore, finding the optimal plan or goal can be termed as the verification of plans with the least (resp. best) costs (resp. rewards) with temporal properties in PAT. In particular, we formulate a goal with additional information such as whether the goal *must* be satisfied (analogy to laws in society) or is *preferably* satisfied with a quantifiable preference weight (analogous to personal habits). For temporal reasoning, the PAT framework supports temporal elements such as wait, execute with time-out and interrupt, start execution within a time, and finish before a deadline.

For the third challenge, with the support of efficient model checking algorithms implemented in PAT, goal reasoning is achievable in an automatic manner. Especially, PAT has been successfully applied in verifying large real-world systems.[1]

4 Planning and Goal Reasoning with PAT

GTN Planning. Automatic model checking has been successfully applied to planning. Intuitively, a classical planning problem can be converted as model checking of the truth of a formula. Given a planning problem containing an initial state, a goal and a set of actions, one can construct a system model by translating every action into a corresponding state transition in model checking. The initial state of the planning problem is also the initial state of the model by assigning value to each variable accordingly. The goal is expressed using a propositional formula. Then we use model checking to verify the negation of the formula, so that the model

[1] PAT Model Checker. http://pat.comp.nus.edu.sg

checker provides a counterexample path consisting of actions that lead to a state where the formula is satisfied.

In our previous work [Li *et al.*, 2014], we have formally established a relation between the classical planning domain and the model checking domain, which helps to reduce errors (e.g., lack of type information) in manual specification or translation. We provide formal semantics for translating PDDL to CSP# and have implemented the translation in PAT. This implementation has been validated with several case studies, which show that using the existing model checker PAT to solve classical planning problems is both feasible and efficient. Thus, we can directly use our previous work [Li *et al.*, 2014] to perform classical planning. For instance, in the sliding game/eight-tiles game, given an initial configuration of the eight tiles, as illustrated in Figure 1a, the objective is to find a path or the shortest plan to reach the goal, as on the right-hand side (Figure 1b). The only actions allowed in this game are to slide the tiles with numbers. By specifying this game as a model checking problem as in Figure 2, we are able to use PAT to find the shortest plan immediately.

8	7	6
	4	1
2	5	3

(a) Initial Configuration

1	2	3
4	5	6
7	8	

(b) Goal

Figure 1: Sliding Game

```
var board = [8,7,6,0,4,1,2,5,3];
var emptypos = 3;
Game() = Left() [] Right() [] Up() [] Down();
Left()=[emptypos!=2&&emptypos!=5&&emptypos!=8]
        goleft{board[emptypos]=board[emptypos+1];
        board[emptypos+1]=0;emptypos=emptypos+1;} -> Game();
Right()=[emptypos!=0&&emptypos!=3&&emptypos=6]
         goright{board[emptypos]=board[emptypos-1];
         board[emptypos-1]=0;emptypos=emptypos-1;}-> Game();
Up()=[emptypos!=6&&emptypos!=7&&emptypos!=8]
      goup{board[emptypos]=board[emptypos+3];
      board[emptypos+3]=0;emptypos=emptypos+3;} -> Game();
Down()=[emptypos!=0&&emptypos!=1&&emptypos!=2]
        godown{board[emptypos]=board[emptypos-3];
        board[emptypos-3]=0;emptypos=emptypos-3;} -> Game();
#assert Game() reaches goal;
#define goal
board[0] == 1 && board[1] == 2 && board[2] == 3 &&
board[3] == 4 && board[4] == 5 && board[5] == 6 &&
board[6] == 7 &&  board[7] == 8 && board[8] == 0;
```

Figure 2: Sliding Game Specification

In our proposed approach we translate GTNs into CSP# in order to treat goal reasoning within the planning algorithm. The outline of this translation is as follows:

- Build an explicit CSP# model $\mathcal{E}$ of the environment, which gives dynamic environment events during execution, in the planning problem.
- Use model checking methods to reason about goals against the composition of the system $\mathcal{S}$ together with the environment $\mathcal{E}$. This allows the treatment of "unspecified" events in a rudimentary way.
- Translate the GTN into CSP# so as to use the task/goal hierarchy to constrain the search space and thus allow significantly larger problems to be solved. This translation is detailed below:
  - tasks are sequences of events in CSP#;
  - goals are guards in CSP#;
  - action methods are primitive state update events;
  - task methods are processes that call sub-task methods according to the task hierarchy;
  - goal methods are processes that call lower level goal methods according to the goal hierarchy; and
  - goal realisation methods are processes that find plans for achieving the goal from the current state.

In addition, we propose to extend the planning framework with the following aspects:

- Introduce timing and general reward/cost functions to the GTN planning formalism with translation into GTN.
- Introduce a full external goal reasoning environment that makes use to the PAT model checker to determine and evaluate suitable goal update responses to dynamic environmental conditions.

Goal Reasoning. It is not expected that the translation from GTN will produce a tractable solution to every goal reasoning problem, thus we need techniques specific to the goal reasoning domain. Based on our previous work on classical planning, we propose to solve goal reasoning as follows. First, we design a formal language (syntax) which is expressive enough to capture the exogenous goal which may contain dynamics or ambiguity. Intuitively, the dynamics and ambiguity are captured by the introduction of non-determinism in goal specification. Thus instead of having a fixed goal, our approach permits *non-deterministic goals*: multiple goals are allowed in the specification, and the choice of the goal is not determined beforehand. The hierarchy in goal networks is captured by the logical relations between assertions. For instance, the goal of the sliding game can be "a visually sorted order of the tiles where the empty tile can either be the first or the last tile" instead of a deterministic goal. The dynamic goal may contain several deterministic goals (sub-goals) as in Figure 3, depending on the position of the empty tile and whether the order is increasing or decreasing. Each sub-goal is captured in Figure 4, and the sub-goals are connected by the *OR* (||) relation.

1	2	3
4	5	6
7	8	

(a) `goal_sub3`

8	7	6
5	4	3
2	1	

(b) `goal_sub4`

	1	2
3	4	5
6	7	8

(c) `goal_sub5`

	8	7
6	5	4
3	2	1

(d) `goal_sub6`

Figure 3: Flexible Goals

The PAT model checker can provide new plans when the goal is changed. Consider the following situation: initially

```
# define goal1
board [0] == 1 && board [1] == 2 && board [2] == 3 &&
board [3] == 4 && board [4] == 5 && board [5] == 6 &&
board [6] == 7 && board [7] == 8 && board [8] == 0;
# define goal2
board [0] == 8 && board [1] == 7 && board [2] == 6 &&
board [3] == 5 && board [4] == 4 && board [5] == 3 &&
board [6] == 2 && board [7] == 1 && board [8] == 0;
# define goal3
board [0] == 0 && board [1] == 1 && board [2] == 2 &&
board [3] == 3 && board [4] == 4 && board [5] == 5 &&
board [6] == 6 && board [7] == 7 && board [8] == 8;
# define goal4
board [0] == 0 && board [1] == 8 && board [2] == 7 &&
board [3] == 6 && board [4] == 5 && board [5] == 4 &&
board [6] == 3 && board [7] == 2 && board [8] == 1;

# define goal_root goal1 || goal2 || goal3 || goal4;
# assert Game() reaches goal_root;

#define goal_sub1 goal_root && board[8] == 0;
#assert Game() reaches goal_sub1;

#define goal_sub2 goal_root && board[0] == 0;
#assert Game() reaches goal_sub2;

#define goal_sub3 goal_sub1 && board[0] == 1;
#assert Game() reaches goal_sub3;

#define goal_sub4 goal_sub1 && board[0] == 8;
#assert Game() reaches goal_sub4;

#define goal_sub5 goal_sub2 && board[1] == 1;
#assert Game() reaches goal_sub5;

#define goal_sub6 goal_sub2 && board[1] == 8;
#assert Game() reaches goal_sub6;
```

Figure 4: Flexible Goals Specification

the dynamic goal is `goal_root` in Figure 4. As the autonomous system executes, it may receive input from the outside environment and react. For example, if the outside environment now requires that `board[0] == 0`, that is, the empty tile must be the first one, then we can use PAT to re-plan towards `goal_sub2`, which is a sub-goal of `goal_root`. Later in execution, we may receive further input from the environment that `board[1] == 8`, then we use PAT to re-plan towards `goal_sub6`, which is a sub-goal of `goal_sub2`. The hierarchical relation between goals is visualised in Figure 5. The goals `goal_sub3`, `goal_sub4`, `goal_sub5`, `goal_sub6` are respectively the goals in Figure 3.

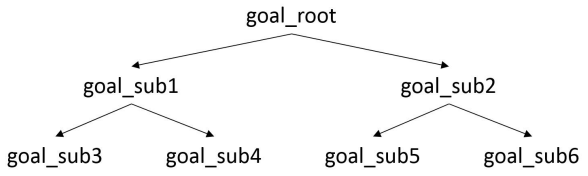

Figure 5: Hierarchy of goals in Figure 4.

Once an exogenous goal is specified, we use model checking to find out all sub-goals and check the reachability of each goal or find the shortest path to a goal, given the current state. Second, we make use of our GTN to CSP# translation so that we can verify a model in PDDL with hierarchical and non-deterministic goals. In addition, we specify extra quantification variables to figure out the optimised plan or goal in dynamic environments. Third, the above-mentioned extension of semantics will then be incorporated in an algorithm for goal reasoning and planning. This algorithm will be based on the existing planning and model checking algorithms.

Temporal Planning and Reasoning. Timed events are an important aspect when modelling the behaviour of agents such as AUV, so supporting them is essential in our work. Since PAT supports timed systems [Sun *et al.*, 2013], we can extend PDDL with durative actions/goals and action/goal preferences. To capture that the actions/goals have durations, we can assign an amount of time units to each action/goal, and enhance the semantics of CSP# so that we allow timed actions. Similar to the semantic rules in timed CSP#, suppose the duration of the action a is $a.time$, then the semantics of executing the action a in the process $P\ deadline[d]$ (cf. [Sun *et al.*, 2013]) can be modified to the following rule:

$$\frac{(V,P) \xrightarrow{a} (V',P')}{(V,P\ deadline[d]) \xrightarrow{a} (V',P'\ deadline[d-a.time])}\ [dl1]$$

Other rules for timed processes (timeout, interrupt, and within) can be changed in the same way. We further extend the syntax of CSP# with new constructs for timed executions. For instance, on top of the syntax of timed CSP#, we can add $P\ repeat[d]$, which means that the process P is repeatedly executed for d time units. This is useful when modelling situations where the agent is required to keep surveying an area for a period of time. Model checking of the temporal reasoning with duration and deadline will be supported by the sophisticated model checking algorithms on timed systems, which involve time duration and deadlines [Mahony and Dong, 2000; Sun *et al.*, 2013].

5 Conclusion and Future Work

We proposed to extend CSP# syntax and semantics, as well as the translation from PDDL to CSP#, to capture goal reasoning and use model checking algorithms implemented in PAT for automatic goal analytics. Currently, we have established the foundation for solving planning problems using PAT. We have developed formal semantics for translating PDDL to CSP# and have implemented the translation in PAT. We also have done preliminary work on extending the CSP# language to solving temporal planning and goal reasoning. Implementing this new language and model checking algorithms for it in PAT is our immediate future task. Another direction of future work is building a larger scale goal reasoning language in which the model checking and planning algorithms are components in this framework.

In addition to realising our idea, we plan to perform goal analytics on AUV systems. There is an increasing interest in the development of AUVs which are capable of being deployed for extended periods (months and longer) with limited communications. Such AUVs need to be capable of automatically modifying their tasks and goals to respond to unexpected situations that arise during the deployment. Thus, the AUV is a perfect case study to validate our proposal.

References

[Alford *et al.*, 2016] Ron Alford, Vikas Shivashankar, Mark Roberts, Jeremy Frank, and David W. Aha. Hierarchical planning: Relating task and goal decomposition with task sharing. In *Proc. 25th International Joint Conference on Artificial Intelligence IJCAI*, pages 3022–3029. IJCAI/AAAI Press, 2016.

[Anderson *et al.*, 2006] Michael L. Anderson, Tim Oates, Waiyian Chong, and Donald Perlis. The metacognitive loop I: enhancing reinforcement learning with metacognitive monitoring and control for improved perturbation tolerance. *J. Exp. Theor. Artif. Intell.*, 18(3):387–411, 2006.

[Cox *et al.*, 2016] Michael T. Cox, Zohreh Alavi, Dustin Dannenhauer, Vahid Eyorokon, Hector Munoz-Avila, and Don Perlis. MIDCA: A metacognitive, integrated dual-cycle architecture for self-regulated autonomy. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, pages 3712–3718. AAAI Press, 2016.

[Ghallab *et al.*, 2016] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning and Acting*. Cambridge University Press, New York, NY, USA, 1st edition, 2016.

[Johnson *et al.*, 2016] Benjamin Johnson, Mark Roberts, Thomas Apker, and David W. Aha. Goal reasoning with informative expectations. In *ICAPS Workshop*, London, UK, 2016.

[Kress-Gazit *et al.*, 2009] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas. Temporal-logic-based reactive mission and motion planning. *IEEE Transactions on Robotics*, 25(6):1370–1381, 2009.

[Li *et al.*, 2012] Yi Li, Jing Sun, Jin Song Dong, Yang Liu, and Jun Sun. Translating PDDL into CSP# - the PAT approach. In *2012 IEEE 17th International Conference on Engineering of Complex Computer Systems*, pages 240–249, July 2012.

[Li *et al.*, 2014] Yi Li, Jin Song Dong, Jing Sun, Yang Liu, and Jun Sun. Model checking approach to automated planning. *Formal Methods in System Design*, 44(2):176–202, 2014.

[Mahony and Dong, 2000] Brendan P. Mahony and Jin Song Dong. Timed communicating object Z. *IEEE Trans. Software Eng.*, 26(2):150–177, 2000.

[Paisner *et al.*, 2013] Matthew Paisner, Michael T. Cox, and Don Perlis. Symbolic anomaly detection and assessment using growing neural gas. In *Proc. 25th IEEE International Conference on Tools with Artificial Intelligence*, pages 175–181. IEEE Computer Society, 2013.

[Roberts *et al.*, 2015] Mark Roberts, Thomas Apker, Benjamin Johnston, Bryan Auslander, Briana Wellman, and David W. Aha. Coordinating robot teams for disaster relief. In *Proceedings of the Twenty-Eighth International Florida Artificial Intelligence Research Society Conference, FLAIRS 2015, Hollywood, Florida. May 18-20, 2015.*, pages 366–371, 2015.

[Sun *et al.*, 2009] Jun Sun, Yang Liu, Jin Song Dong, and Jun Pang. Pat: Towards flexible verification under fairness. volume 5643 of *Lecture Notes in Computer Science*, pages 709–714. Springer, 2009.

[Sun *et al.*, 2013] Jun Sun, Yang Liu, Jin Song Dong, Yan Liu, Ling Shi, and Étienne André. Modeling and verifying hierarchical real-time systems using stateful timed CSP. *ACM Trans. Softw. Eng. Methodol.*, 22(1):3:1–3:29, March 2013.

[Wilson *et al.*, 2016] Mark A. Wilson, James McMahon, A. Wolek, David W. Aha, and B.H. Houston. Toward goal reasoning for autonomous underwater vehicles: Responding to unexpected agents. In *25th International Joint Conference on Artificial Intelligence (IJCAI) Workshop on Goal Reasoning*, New York, NY, 2016.