Formal Methods for Control Synthesis in Partially Observed Environments: Application to Autonomous Robotic Manipulation

Thesis by

Rangoli Sharan

In Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

California Institute of Technology

Pasadena, California

2014

(Defended April 2, 2014)

Acknowledgements

I owe my deepest gratitude to my advisor, Professor Joel Burdick, for his support and guidance throughout my Ph.D. studies. In addition to providing crucial insight and direction for the completion of this research, his encouragement to pursue my own interests is greatly appreciated. I have been very fortunate to have Joel's unreserved understanding – in pursuing my own research problem of interest, exploring opportunities outside the PhD program, and the challenges of my long distance marriage. He has greatly shaped my values and attitude towards work and life. I am also thankful to have had the opportunity to work on the DARPA ARM-S challenge at the Jet Propulsion Laboratory (JPL). My time there was not only a great learning opportunity, but also provided the main motivation for this work.

It has been a great pleasure to have Professor Richard Murray as a mentor. It has been a wonderful experience to interact frequently with him. The research presented in this thesis is greatly inspired by the work that he and his students are actively developing. It has been eye-opening to see the process of developing fundamental theoretical results and the effort to bring together collaborators from other academic institutions and partners from the industry to move new knowledge into crucial application areas.

I am honored to have Professor Jim Beck on my thesis committee. Some of the research during my early Ph.D. years utilized Bayesian inference algorithms, to which I was first exposed in a class taught by Professor Beck. To him I owe new lenses with which I view many technical problems that I face in daily life or find worthy to solve for the world at large.

Some of my happiest workdays have been at JPL working with Dr. Nicolas Hudson on the DARPA ARM-S challenge. While my research was peripheral to the team's success in the competition, I always found myself welcome to his limited time and the resources of his lab. It is a great pleasure to have him on my thesis committee.

I want to acknowledge the wonderful experience with the whole team at JPL: Paul Hebert, Jeremy Ma, Max Bajracharya, and especially Dr. Paul Backes who was extremely supportive and facilitated my work at JPL.

A special thanks to Maria Koeper, whose generous help has been a secret sauce in making my time at Caltech extremely smooth.

My friends and colleagues from the Burdick and Murray groups: Tom, Melissa, Matanya, Krishna, Jeff, Eric, Scott and Vasu; and my roommates for various lengths of time: Piya and Na, have inspired me with their relentless pursuit of deeper understanding and also provided much needed light moments during the workdays.

I am fortunate to have enjoyed the unconditional friendship of Sushree and Jaykrishnan. They provided a much needed support structure in Pasadena. The tea breaks and home-cooked meals shared together are some of my fondest memories during the years at Caltech. My closest friends – Anjali, Jyoti, Romila, Shruti and Vertika – have stood by me through every personal and professional upheaval of the last decade. They are a never-ending source of joy and support.

Finally, I want to acknowledge the incredible support I receive everyday from my family. My parents, Shashi and Rita, have dedicated their lives and sacrificed greatly to support their two children in any and all endeavors. I can only hope to navigate life's challenges with their perseverance and foresight. My brother Anand, who has always been my safety net, and his wife Nidhi, bring laughter and joy everyday. I take this opportunity to especially remember my late grandfather Vaidehi, my earliest academic mentor, to whom I owe my apetite for learning and problem solving. I am extremely fortunate to have found new sources of love and support in Pramila, Toshita, Vaibhav, Svana, and Shwetank, great new additions to my family, that occured during my PhD years.

I have relied implicitly on Shwetank's companionship for the last six years. He has been my pillar of strength, steadfast in his belief in me during moments of doubt. He inspires me to learn new things, think new thoughts, and live more mindfully every single day. Most importantly, the days are filled with love, laughter, and meaning because of him.

Abstract

Modern robots are increasingly expected to function in uncertain and dynamically challenging environments, often in proximity with humans. In addition, wide scale adoption of robots requires on-the-fly adaptability of software for diverse application. These requirements strongly suggest the need to adopt formal representations of high level goals and safety specifications, especially as temporal logic formulas. This approach allows for the use of formal verification techniques for controller synthesis that can give guarantees for safety and performance. Robots operating in unstructured environments also face limited sensing capability. Correctly inferring a robot's progress toward high level goal can be challenging.

This thesis develops new algorithms for synthesizing discrete controllers in partially known environments under specifications represented as linear temporal logic (LTL) formulas. It is inspired by recent developments in finite abstraction techniques for hybrid systems and motion planning problems. The robot and its environment is assumed to have a finite abstraction as a Partially Observable Markov Decision Process (POMDP), which is a powerful model class capable of representing a wide variety of problems. However, synthesizing controllers that satisfy LTL goals over POMDPs is a challenging problem which has received only limited attention.

This thesis proposes tractable, approximate algorithms for the control synthesis problem using *Finite State Controllers* (FSCs). The use of FSCs to control finite POMDPs allows for the closed system to be analyzed as finite global Markov chain. The thesis explicitly shows how transient and steady state behavior of the global Markov chains can be related to two different criteria with respect to satisfaction of LTL formulas. First, the maximization of the probability of LTL satisfaction is related to an optimization problem over a parametrization of the FSC. Analytic computation of gradients are derived which allows the use of first order optimization techniques.

The second criterion encourages rapid and frequent visits to a restricted set of states over infinite executions. It is formulated as a constrained optimization problem with a discounted long term reward objective by the novel utilization of a fundamental equation for Markov chains - the Poisson equation. A new constrained policy iteration technique is proposed to solve the resulting dynamic program, which also provides a way to escape local maxima.

The algorithms proposed in the thesis are applied to the task planning and execution challenges

faced during the DARPA Autonomous Robotic Manipulation - Software challenge.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Motivation

Robots and autonomous control systems are regularly required to function in uncertain and dynamically changing environments. They are also increasingly required to satisfy complex sets of rules that specify desired system behavior. These requirements are often specified in addition to the traditional control theoretic goals, e.g., set point or trajectory tracking, stability margins, response time, etc. Such complex goal satisfaction requirements arise in diverse areas like aerospace, energy management systems, robotics, civic and transportation planning, resource and supply chain management, autonomous vehicles and manufacturing. Many of these application areas also employ numerous, possibly distributed, sensors and actuators.

Autonomous platforms such as self-driving cars and unmanned aerial vehicles (UAVs) are now being deployed in the commercial, military, and public domains. The success of these platforms are crucially dependent on their safe operability and ability to adapt to continuously evolving local laws. This presents the challenge of verifying the control and sensing algorithms against safety and abstract rules, and synthesizing new controllers if required.

This work is motivated by the challenges faced by autonomous robots deployed in unstructured environments. A particular motivation is the need of such robots to manipulate objects in their environment through the use of articulated limbs. Applications for such robots are numerous, some examples being

- Personal Robotics: Robots that are capable of manipulating made-for-human objects to provide assistance at home and in the office. Of special interest are robots that can assist the disabled or elderly.

- Disaster Response: These robots are envisioned as first responders for search and rescue in disaster areas. Alternatively, they can be deployed to unsafe environments to diagnose and restore safe operation. A recent example is the Fukushima nuclear power plant incident, in

which damaged areas could not be accessed safely by humans due to nuclear radiation.

- Security and Defense: Applications in security and defense include searching for IEDs in luggage, vehicles or on persons at checkpoints, which requires dextrous manipulation in unstructured and unknown environments, or in reconnaissance.

In particular, this thesis tackles a few of the challenges that were faced by the Caltech/JPL team during the DARPA Autonomous Robotic Manipulation - Software (ARM-S) challenge [3]. This challenge, which focused on autonomous robotic manipulation of real-world objects during complex tasks in semi-structured environments, is described in detail in Chapter 7. For example, one challenge task involved the use of a two-armed robot to replace a tire attached to an immobile hub, as shown in Figure 1.1. A typical process to remove the wheel might include the following sequence: the robot locates and acquires an impact driver, grasps the driver so that it can properly depress the power trigger, positions the driver on the lug nuts, removes the nuts one by one, puts down the impact driver so that both hands are free, and removes the wheel. The robot faces many hurdles during this process. Since the tool and tire manipulation tasks are kinematically linked, the robot may pick up the impact driver in a pose where the lug nuts can be removed, but the driver trigger cannot be depressed. Second, sensor-based operations, such as localization of objects, are noisy, which may result in failure at the task level. Worse still, it may be impossible to determine if some subtasks were executed correctly, e.g. visual or force-torque sensors may be incapable of determining precisely whether the lug nuts were successfully removed. It is currently difficult to program such tasks at a high level of abstraction, and to guarantee some level of performance.

Two main problems arise in the context of complex manipulation tasks requiring a sequence of actions. First, the free configuration space, C_{free}, [91, 92] can have high dimension and a complex representation due to the presence of multiple manipulator joints and numerous objects, commonly referred to as configuration space obstacles, C_O. The dimension and complexity are further exacerbated because these obstacles can be moved, and attached to/detached from other obstacles or the end effector by grasping or placing objects. Furthermore, the attachment can be any of a variety of rigid or non-rigid configurations. These characteristics imply a "hybrid" planning problem. The movement of the end effector in C_{free} when C_O is unchanging is a path planning planning problem in continuous space, whereas the process of grasping and attachment of objects introduces discrete changes to the definition of C_O and the topology of C_{free} - a problem which and falls under the purview of task planning.

Second, further challenges are faced due to imperfect control and the inability to accurately localize, using on-board sensors, the robot, its end effector and the obstacles in the workspace. In fact, imperfect observation makes the decoupling of planning and execution difficult, if not impossible. By execution, it is implied that a set of local controllers are sequenced in time to effect the robot

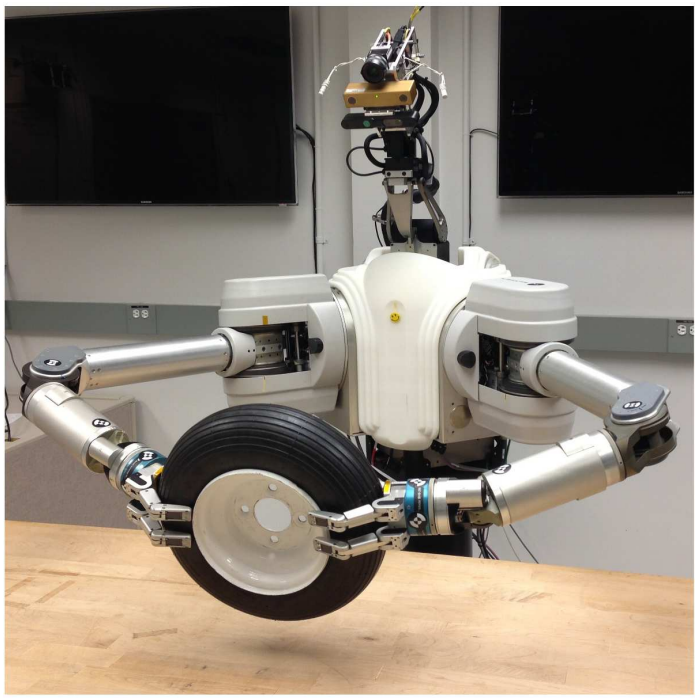


Figure 1.1: The DARPA ARM-S Robot. It consists of two arms are Barrett Technology 7-DOF WAM arms with a 6-DOF force sensor at each wrist. The head is mounted on a 4-DOF neck and consists of a Point Grey Research Bumblebee2 color stereo camera, Primesense ASUS Xtion-Pro depth camera, Prosilica Gig-E color camera and two microphones.

manipulator in such a way that changes in C_O only occur at the end of any local control task. However, since the outcome of the local control execution cannot be guaranteed, nor can the change in C_O be observed accurately, the task level plan must incorporate the execution explicitly, i.e., it must describe how to interpret sensor observations to determine the next local controller to deploy.

Instead of tackling the problem of autonomous manipulation planning and its control and execution in their full hybrid dynamical system domain, this thesis focuses exclusively on planning over finite discrete choices. The imperfect control and sensing encountered at the continuous low level controllers are assumed to be abstracted to obtain a finite discrete model by the introduction of uncertainty in the discrete motion model and by incorporating partial observability of the discrete states.

Additionally, this work focuses on synthesizing a control policy for goals or specifications that can be expressed in a formal language. Specifically, these specifications will be expressed as formulas belonging to the class of propositional logic called *Linear Temporal Logic*. Linear Temporal Logic offers a rich set of temporal specifications or properties that the controlled system can be verified against. The key benefit of these specifications is that they allow verification of properties over *paths* taken by the controlled system rather than properties of the goal state alone. For example, for a robot on an assembly line shared with humans, it may be required that the robot halts all motion when a human is within a specified distance of its workspace, and then resumes its trajectory to a goal state when the human recedes. Another example of a temporal and logical specification over the robot's trajectory is the requirement to reach multiple goal states in a specified order.

1.2 Related Work

Validation and verification form two aspects of testing a software or hardware system's fitness for its intended purpose. While validation is concerned with generating or designing specifications as per need, verification is concerned with checking if the system design or implementation satisfies those specifications. In computer science and control theory, formal methods from mathematics are used extensively for specification, design, implementation and verification of software and hardware systems. Formal verification is concerned with proving or disproving whether the algorithms will behave properly with respect to the specifications. Software systems are usually modeled as discrete state and discrete time, and typical specifications are concerned with avoiding deadlocks and livelocks in concurrent systems, or ensuring safety, reachability and liveness conditions in more general settings. The specifications are usually in the form of temporal logics such as Linear Temporal Logic (LTL) or Computational Tree Logic (CTL).

There are two main techniques to formal verification. The first is model checking, which entails systematic and exhaustive checking of the mathematical model, and this process is constrained to

finite models. Pioneering work in model checking formalisms can be found in [36, 43, 125]. Model checking suffers from the curse of dimensionality: even a small symbolic representation of either the specifications or the underlying system, can result in very large intermediate state spaces which the model checker must explore. However, this has been mitigated recently in two ways. Recently, a large and expressive subset of temporal logic specifications were shown to have solutions with only polynomial time complexity [114]. Additionally, several techniques in symbolic model checking allow sets of states to be explored simultaneously, circumventing explicit state enumeration. These methodologies include Binary Decision Diagrams [26], model abstraction and counterexample based refinement [37] and partial order reduction [48,111,142]. The work presented in this thesis takes the model checking approach to formal verification.

The second approach is deductive verification. In this approach, the system model and the specifications are used to generate a set of proof obligations. The burden is then on automated or interactive theorem provers to establish the truth of these proof obligations. Deductive verification is based on the framework found in [45, 58]. The initial introduction of temporal logic for specification and verification of concurrent systems also relied on theorem proving [116, 117]. While this methodology allows for infinite state space models, often skilled interaction between the designer and the proof system is required, an approach which is difficult to scale to large systems.

In typical control design problems, the physical systems are usually not exclusively discrete. For systems governed by differential equations, Lyapunov based techniques and reachability analysis provide methods to design controllers that satisfy safety, reachability and stability criteria. However, many complex controlled autonomous systems are hybrid in nature, e.g. robots, aircraft, climate control, energy management systems, etc. The hybrid nature arises due to the complex digital software, abstraction of mixed signal circuits, switches, etc. A survey of formal verification techniques for hybrid systems can be found in [2].

Recently, formal methods have become increasing popular in robotics [40, 76–78, 148], where simultaneous motion and task planning is a challenging problem. In [40,140] the authors synthesize controllers that give probabilistic guarantees of temporal logic specifications in discrete systems. In [68] take a sampling based approach to carry out motion planning by incrementally building discrete abstractions that satisfy local formal specifications. In [146], the authors demonstrate a receding horizon approach by decomposing a high level specification into several components – generation of a sequence of short term goals, trajectory planning and continuous controller design – while provably satisfying the overall specification. In [151], the authors demonstrate how game theoretic methods can be employed to carry out multi-robot motion planning.

In part, the adoption of formal methods in robotics and control applications has been spurred by new techniques and tools for obtaining discrete abstractions for hybrid systems and motion planning problems. For linear dynamical systems, automatic discrete abstraction and application of model

checking methods for control synthesis can be found in [75]. In [14], the authors abstract polygonal environments into a discrete model to synthesize controllers for temporal logic specifications. In [35], a noisy continuous state system is approximated as a finite MDP to satisfy various classes of temporal logics. The main idea is that the discrete abstraction is carried out by computing both the continuous state reachability properties and the invariance of the discrete states or high level properties. This ensures that a high level, discrete plan is guaranteed to be realizable by a series of continuous controllers [147, 148]. For cases when the environment is only partially known *a-priori*, on-line iterative and backtracking frameworks for formal control synthesis have also been proposed [87, 95]. Moreover, in [88], the authors propose techniques for locally patching the control strategy if parts of the original strategy are invalidated due to change in environment or incorrect assumptions.

Numerous practical applications of formal methods in robotics have been demonstrated. In [34] the authors demonstrate the applicability of these methods in dextrous manipulation tasks focusing on finger-gaiting. Similarly, model checking has been employed in autonomous vehicles [4, 121] and to specify emergent behaviors in swarm robotics [143].

It is also important to note the contributions from research in the area of domain independent planning, as robot task planning is a key application for the algorithms developed in that area. Traditionally, domain independent planning concerned itself with planning exclusively in the discrete and deterministic domain – e.g. problems representable in STRIPS-like representation scheme. Classical planning problems could also be exclusively characterized as reachability problems – the solution from the planning algorithm would provide a sequence of actions that can lead to a goal state from the initial state. However, domain independent planning has grown to include almost all of robot motion and task planning challenges in its purview – temporal goal satisfaction, timed actions, preference based action selection, probabilistic and partially observable domains, continuous states, etc. A bibliographic overview of task planning from a domain independent perspective is provided in Chapter 7.

In robotics, Linear Temporal Logic (LTL) is a popular choice for robot goal and safety specification as it is a powerful and expressive class of temporal logic with intuitive correlation to natural language [59, 104]. Notably, LTL formulas can represent goals over infinite executions. This is useful for representing persistent surveillance and applications where robots are always online. In order to capture environmental disturbance, it is often useful to model the dynamics as a probabilistic system. Markov decision processes are a popular choice for the discrete abstraction of noisy systems. In the case of (fully observable) Markov decision process (MDP), synthesis of controllers with probabilistic satisfaction guarantees of LTL specification is well understood [8]. In fact, for fully observable MDPs under LTL specifications, robust [144] and receding horizon controllers [146] have been formulated.

For POMDPs, design of LTL controllers is largely an open problem. For unbounded memory

strategies EXPTIME-completeness of a broad set of objectives (parity objectives) is proven in [32]. In a recent publication [33], the existence and construction of finite memory strategy for (strictly) positive probability of satisfaction is shown to be an EXPTIME-complete problem. The memory requirements for a controller that guarantees satisfaction with probability 1 are established to be exponential as well. In [145], the authors discretize the belief space, which is the set of all probability distributions over the state space, *a-priori*. Then the resulting discrete model is a fully observable MDP for which formal control synthesis is well established. However, for POMDPs a *a-priori* belief space discretization can be suboptimal and prohibitive for large state spaces. In this thesis, a goal centric approach is taken – the controller has discrete states and optimizes the input to the controlled system and its own dynamics for each state according to the LTL specification. Such class of controllers is known to implicitly partition the belief space of the controlled system into sets of regions – each corresponding to a discrete controller state.

1.3 Thesis Overview and Contributions

The objective of this thesis is to provide tractable algorithms for synthesis of control systems which satisfy Linear Temporal Logic specifications in a partially observable setting. Specifically, the thesis will take a model based approach – the robot and environment will be modeled as a finite, discrete time Partially Observable Markov Decision Process. The motivating application consists of combined task planning and execution in autonomous systems with imperfect sensing and control.

Chapter 2 provides the formalization and background of the problem domain. It introduces Linear Temporal Logic (LTL) in the context of a general state transition system, and the methodology of verifying LTL formulas by constructing an automaton. It also formalizes the model that is used to abstract the controlled system of interest. The system is assumed to be modeled as a finite Partially Observable Markov Decision Process (POMDP). The chapter also provides a brief overview of the classical problems and solution approaches for this class of model.

Chapter 3 formalizes the problem of synthesizing LTL controllers for POMDPs. This is done in the context of a specific class of controllers namely, the Finite State Controller (FSC). The choice of this class is motivated by the reduction in complexity when both the model and the controller are finite. The thesis leverages the well known fact that when a finite POMDP is controlled by an FSC the resulting closed loop system is a finite Markov chain. Several qualitative and quantitative questions relating to LTL formula satisfaction over POMDPs are posed in the context of FSCs. Moreover, key facts about discrete time Markov chains are presented which will be used to formulate the algorithms in the later chapters. Of importance is the probability space over *paths* of the Markov chains, since the thesis is concerned with the problem of maximizing the probability of satisfying LTL formulas.

Chapter 4 begins by introducing the notion of the *structure* and *quality* of an FSC. It then shows how to solve the problem of maximizing the probability of satisfying an LTL formula when the structure of the FSC is fixed, and the controller transition and action choices are parametrized. The problem is formulated as a nonlinear optimization problem and gradient computations are derived analytically to allow the use of first order optimization methods. Gradient based methods have been used for reward maximization problems over POMDPs. However, this thesis presents a new explicit formulation for gradient based optimization of LTL satisfaction probability in which there is no *a-priori* reward associated with states and actions of the controlled system. Case studies are presented at the end of this chapter to demonstrate some key aspects and limitations of these algorithm. A preliminary version of the work in this chapter of the thesis also appears in [131].

Chapter 5 deviates from the problem of maximizing LTL satisfaction *probability* explicitly. The main focus is to derive a reward scheme that drives the solution of the LTL satisfaction problem to rapid convergence to a steady state behavior in which certain states are visited often while others are avoided with probability 1. The notion of rewarding *frequent* visits to "good" states of the state space is introduced via discounting. Discounting not only ensures rapid convergence to steady state, but also frequent visits to "good" states *during* steady state system execution, setting it apart from the gradient based algorithm of Chapter 4. While having the practical importance of causing the controlled system to visit goal states sooner and frequently, the decision to use discounting is also motivated by the convergence properties and existence of efficient dynamic programming algorithms. The proposed reward schemes are completely new to the best of my knowledge. However, the reward scheme and the resulting optimization problems proposed in this thesis form sound, but not complete algorithms for maximization of LTL satisfaction probability over POMDPs.

Chapter 6 proposes a Policy Iteration algorithm to solve the optimization problem introduced in the preceding chapter. Some key aspects relating to this algorithm is that it offers a way to search directly over both the structure and the quality of the FSC. This overcomes the limitation of the gradient method of Chapter 4 which did not offer a way to choose the size or structure of the FSC. Moreover, the FSC transitions and action choices are no longer parametrized, but directly computed as a result of the optimization. The proposed algorithm is based on the bounded policy iteration algorithm proposed in [118] and [54]. However, the use of the Poisson Equation for Markov chains as a constraint to avoid certain states in steady state and its solution is entirely new to the best of my knowledge.

Chapter 7 begins by presenting a brief overview of the evolution of task planning problems in the AI and robotics communities. The focus is on domain independent planning. Next, a brief architecture level overview of the Caltech/JPL's entry to the DARPA ARM-S challenge is presented and the unique challenges of manipulation tasks are described. Some preliminary work, published in [64], that I carried out to tackle the challenges in task sequencing are used to motivate the need

for integrating a task level planner during execution for the system. The example task of removing a wheel from a fixed hub, introduced earlier in this chapter, is formalized using a recently introduced planning domain description language and two case studies of LTL satisfaction for the ARM-S robot are carried out.

The thesis concludes by discussing further insights about the advantages and limitations of the proposed methodology. Some key challenges and future work, whose successful resolution can enable solving LTL satisfaction in large scale models, are also discussed. These can be found in Chapter 8.

Chapter 2

Background

This chapter provides background for the rest of the thesis. First, an overview of Linear Temporal Logic, its representation using automata, and typical temporal properties in control are described. Next, finite partially observable Markov decision processes (POMDPs) are formally reviewed. Finally, classical problems in this domain and typical solution methods are described.

2.1 Linear Temporal Logic

Temporal logic is a branch of logic that enables representation and reasoning about temporal aspects of a system [8,42,65]. It deals with propositions qualified in terms of time, e.g., "The train is *always* on time." It was introduced first as *tense logic* in [119]. Since its first use as a specification language by Pnueli in [96], temporal logic has been demonstrated to be especially suited to reason about concurrent programs. Is has been utilized to formally specify and verify behavior in a variety of applications [30,38,46,60,66,86,115,148].

In this thesis, a subset of temporal logic, namely linear temporal logic (LTL) is considered. Before formally defining LTL, a brief overview of the underlying building blocks for LTL is provided. The notation in this section closely follows that of [148].

Definition 2.1.1 *A* system *consists of a set V of variables. The* domain *of V, denoted by $dom(V)$, is the set of valuations of V. A* state *of the system is an element $v \in dom(V)$.*

Definition 2.1.2 *An* atomic proposition *is a statement p on system variables $v \in V$ that has a unique truth value ($True$ or $False$) for a given state v. Let $v \in dom(V)$ be a state of the system and p be an atomic proposition. Writing $v \Vdash p$ signifies that p is $True$ at the state v. Otherwise $v \nVdash p$.*

Next, an infinite sequence of a system's states is called an *execution* of the system. For discrete time systems, which are the topic of this thesis, states are assumed to be evaluated only at time steps $t \in \{0, 1, \dots\}$. An execution can be written as $\sigma = v_0 v_1 v_2 \dots$ where for each $t \geqslant 0$, the state at time t, $v_t \in dom(V)$.

LTL is a propositional logic built inductively from logical connectives and temporal modal operators. The logical connectives are the familiar operators: *negation* $\neg$, *disjunction* $\vee$, *conjunction* $\wedge$, and *implication* $\implies$. The temporal operators include *always* $\Box$, *eventually* $\Diamond$, *next* $\bigcirc$, and *until* $\mathcal{U}$.

Syntax: An LTL formula is defined inductively as follows:

1. any atomic proposition p is an atomic formula; and
2. given LTL formulas φ and ψ, the expressions $\neg\varphi$, $\varphi \vee \psi$, $\bigcirc\varphi$, $\varphi \,\mathcal{U}\, \psi$ are LTL formulas.

Other formulas can be defined in terms of the above as follows:

- $\varphi \wedge \psi \triangleq \neg(\neg\varphi \vee \psi)$,
- $\varphi \implies \psi \triangleq \neg\varphi \vee \psi$,
- $\Diamond\varphi \triangleq True \,\mathcal{U}\, \varphi$, and
- $\Box\varphi \triangleq \neg\Diamond\neg\varphi$.

Semantics: An LTL formula is interpreted over an infinite sequence of states. Given an execution $\sigma = v_0 v_1 v_2 \ldots$ and an LTL formula φ, it is said that φ *holds at position* $t \geqslant 0$ of σ, written $v_t \models \varphi$, if and only if (iff) φ holds for the remainder of the execution σ starting at position i. Formally, the semantics of LTL is defined inductively as follows.

- For an atomic proposition p, $v_t \models p$ iff $v_t \Vdash p$,
- $v_t \models \neg\varphi$ iff $v_t \not\models \varphi$,
- $v_t \models \varphi \vee \psi$ iff $v_t \models \varphi$ or $v_t \models \psi$,
- $v_t \models \bigcirc\, \varphi$ iff $v_{t+1} \models \varphi$, and
- $v_t \models \varphi \,\mathcal{U}\, \psi$ iff $\exists t' \geqslant t$ such that $v_{t'} \models \psi$ and $\forall t'' \in [t, t'), v_{t''} \models \varphi$.

Thus the temporal operators are thus interpreted as follows: $\bigcirc\varphi$ holds at position t of σ iff φ holds at the next state $t+1$, $\Box\varphi$ holds at position t iff φ holds at every position of σ start at position t, and $\Diamond\varphi$ holds at position t iff φ holds at some position $t' \geqslant t$ in σ.

Definition 2.1.3 *An execution* $\sigma = v_0 v_1 \ldots$ satisfies φ, *denoted by* $\sigma \models \varphi$, *if* $v_0 \models \varphi$.

2.1.1 Verification of LTL satisfaction using Automata

From the previous section, it is clear that an LTL formula φ defines a set of infinite executions. In order to verify and synthesize controllers that enable the system to satisfy φ, the aim is to construct an automaton which accepts only those infinite executions [8]. It is well known that for any LTL formula φ over a set of atomic propositions AP, one can construct a *deterministic Rabin automaton* (DRA), with the input alphabet 2^{AP}, that accepts all and only those infinite words that satisfy the LTL formula [49,73,141]. Algorithms for converting LTL formulas to DRAs can be found in [74,141], and a common free software tool to carry out the conversion is at [72].

Definition 2.1.4 *A* deterministic Rabin automaton *(DRA) is a five-tuple* $\mathcal{A} = (Q, \Sigma, \delta, q_0, \Omega)$*, where*

- Q *is the set of states;*
- Σ *is the input alphabet. In the context of atomic propositions over a system,* $\Sigma = 2^{AP}$*.*
- $\delta : Q \times \Sigma \to Q$ *is the deterministic transition function.*
- $q_0 \in Q$ *is the initial state.*
- $\Omega = \{(Avoid_r, Repeat_r) | r \in \{1, \ldots, N_\Omega\}, \quad Avoid_r, Repeat_r \subseteq S\}$ *is the Rabin acceptance condition.*

Definition 2.1.5 (Rabin Acceptance) *A run* $\pi = q_0 q_1 \ldots$ *of a DRA* $\mathcal{A}$ *with acceptance condition* $\Omega = \{(Avoid_1, Repeat_i), \ldots (Avoid_{N_\Omega}, Repeat_{N_\Omega})\}$ *is* accepting *if there exists an* $r \in \{1, \ldots, N_\Omega\}$ *such that* $Inf(\pi) \cap Avoid_r = \varnothing$ *and* $Inf(\pi) \cap Repeat_r \neq \varnothing$*. Here* $Inf(\pi)$ *are the set of states that occur infinitely often in* π*.*

Stated otherwise, the Rabin acceptance conditions mean that for some pair $(Avoid_r, Repeat_r) \in \Omega$, no state in $Avoid_r$ is visited infinitely often, while some state in $Repeat_r$ is visited infinitely often.

The DRA is used for verification of an LTL formula φ as follows. It is assumed that the interesting properties of the system are given by a set of atomic propositions AP over the variables V of the system. An execution $\sigma = v_0 v_1 \ldots$ of the system therefore leads to a unique (infinite) *trace* over the truth evaluations of AP, given by $h(\sigma) \triangleq h(v_0)h(v_1)\ldots$. Here $h(v_t) \in 2^{AP}$ simply denotes the truth value of all atomic propositions in AP at time step t using the state v_t. At the start of the system execution, the DRA corresponding to φ is initialized to its initial state q_0. As the system execution progresses, the evaluations $h(v_t)$ for $t = 0, 1, \ldots$ dictate how the DRA evolves via the transition function δ. The execution σ satisfies φ iff the DRA accepts $h(\sigma)$. This idea is formalized concretely in Section 3.2 for the specific transition system of a partially observable Markov decision process, where a product of the system model with the DRA is formally constructed to embody this process of the DRA being driven by the POMDP execution.

2.1.2 Common LTL formulas in Control

There are a few LTL formulas that arise naturally for controlled systems, denoting the typical goals of control system design. A list of these typical formulas, along with a graphical representation of their corresponding deterministic Rabin automaton, is provided below. The translation to automaton representation was carried out using the software tools described in [72] and [61].

2.1.2.1 Safety or Invariance

A safety or invariance formula, Fig. 2.1(a), is of the form $\Box p$, where p is an atomic proposition or LTL formula. It asserts that the property p remains invariant throughout an execution. Invariance is used to specify the requirement that something bad never happens. For example, for a wheeled robot, "Never be close to a staircase" can ensure operation safety.

2.1.2.2 Guarantee or Reachability

A guarantee formula, Fig. 2.1(b), is of the form $\Diamond p$. It specifies that the proposition p will eventually become true during the execution. It denotes that something good is guaranteed to occur, or some goal state will be reached by the system.

2.1.2.3 Progress or Recurrence

A progress property, Fig. 2.1(c), is given by $\Box\Diamond p$. It ensures that the property p holds infinitely often during execution. It essentially ensures that some good property will always be achieved.

2.1.2.4 Stability or Persistence

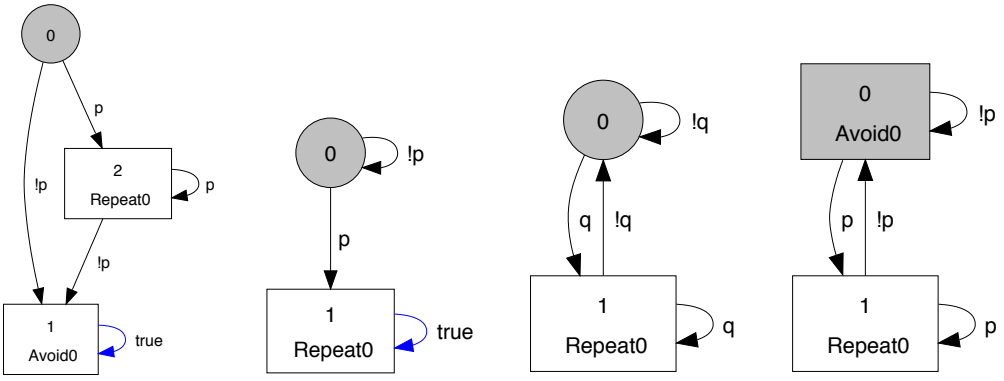
A stability property, Fig. 2.1(d) given by $\Diamond\,\Box\, p$, ensures that after a transient period, a property p becomes true and *then* remains invariant for the rest of the execution. This is similar to the notion of stability in classical control, in which the system is steered towards an operating point and the control must ensure that it operates close to, or at, this operating point.

2.1.2.5 Obligation

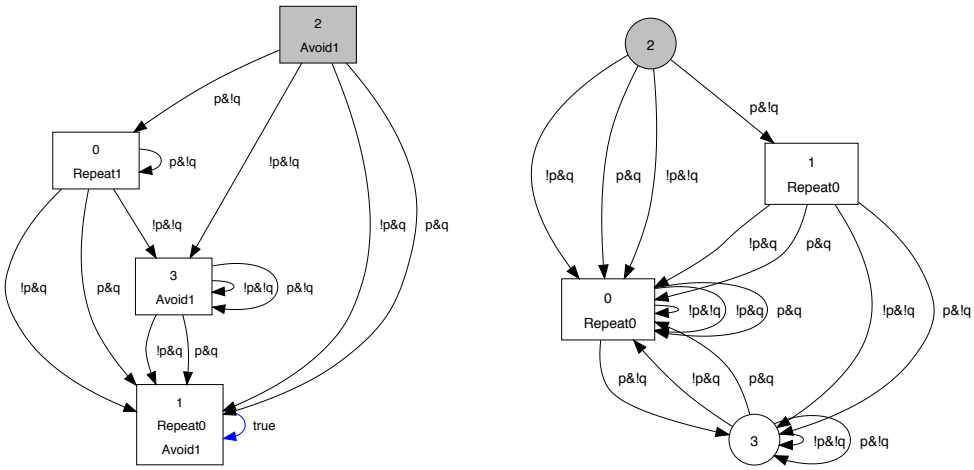
The obligation property, Fig. 2.1(e), given by $\Box p \vee \Diamond q$, is a disjunction of invariance and guarantee. Thus, it ensures that either some reachability goal is met or some invariant property always holds.

2.1.2.6 Response

The response property, Fig. 2.1(f), given by $\Box(p \implies \Diamond q)$, ensures that if p becomes true at any point in the execution, then q is guaranteed to be true at some point in the future subsequently. It can be used to describe desired system response to external disturbances.



(a) Safety $\Box p$ (b) Guarantee $\Diamond p$ (c) Progress $\Box\Diamond p$ (d) Stability $\Diamond\Box p$



(e) Obligation $\Box p \vee \Diamond q$ (f) Response $\Box(p \implies \Diamond q)$

Figure 2.1: Graphical representation of the DRA translations of common LTL specifications. The states of the DRA are given by the nodes (circles or boxes) and are numbered. The gray nodes denote start state of the DRA. The deterministic transitions are marked by the truth evaluations of AP. Square nodes belong to some numbered Rabin acceptance condition pair $(Repeat_0, Avoid_0), \ldots$. Note that in the DRA for Obligation specification in (e), the node 1 simultaneously belongs to $Repeat_0$ and $Avoid_1$.

2.2 Labeled Partially Observable Markov Decision Process

Definition 2.2.1 (Labeled-POMDP) *Formally, a* labeled-POMDP*, $\mathcal{PM}$ consists of:*

- $|\mathcal{S}^{model}|$ *states* $\mathcal{S}^{model} = \{s_1^{model}, \ldots, s_{|\mathcal{S}^{model}|}^{model}\}$ *of the world,*
- $|Act|$ *actions or controls* $Act = \{\alpha_1, \ldots, \alpha_{|Act|}\}$ *available to the reactive agent,*
- $|\mathcal{O}|$ *observations* $\mathcal{O} = \{o_1, \ldots, o_{|\mathcal{O}|}\}$,
- $|AP|$ *atomic propositions* $AP = \{p_1, p_2, \ldots p_{|AP|}\}$.
- *Possibly a reward,* $r(s_i^{model}) \in \mathbb{R}$*, for each state* $s_i^{model} \in \mathcal{S}^{model}$.

This thesis assumes that the POMDPs are *finite*, in which $\mathcal{S}^{model}$, Act, $\mathcal{O}$, and AP are finite sets. Each action $\alpha \in Act$ determines the probability of making a transition from some state $s_i^{model} \in \mathcal{S}^{model}$ to state $s_j^{model} \in \mathcal{S}^{model}$ given by $T(s_j^{model}|s_i^{model}, \alpha)$. Additionally, for each state s_i^{model}, an observation $o \in \mathcal{O}$ is generated independently with probability $O(o|s_i^{model})$. It is also assumed that the probability that the start state of the world is s_i^{model} is given by the distribution $\iota_{init}(s_i^{model})$. The probabilistic components of a POMDP model must satisfy the following:

$$\sum_{s^{model} \in \mathcal{S}^{model}} T(s^{model}|s_i^{model}, \alpha) = 1 \quad \forall s_i^{model} \in \mathcal{S}^{model}, \alpha \in Act$$

$$\sum_{o \in \mathcal{O}} O(o|s^{model}) = 1 \quad \forall s^{model} \in \mathcal{S}^{model}$$

$$\sum_{s^{model} \in \mathcal{S}^{model}} \iota_{init}(s^{model}) = 1.$$

Finally, for each state s_i^{model}, a labeling function $h(s_i^{model}) \in 2^{AP}$ assigns the truth value to all the atomic propositions in AP in each state. Note that, the truth valuation of propositions in AP can in general be both a function of state and action, and can even be stochastic. For ease of exposition, this work exclusively focuses on state dependent deterministic propositions.

While in general, rewards may be a function of both the state and the action taken by an agent, it is assumed that rewards are a function of the state only, and are awarded once during state transition. To state this formally, if the world state transitions from s_i^{model} to s_j^{model}, then reward $r(s_j^{model})$ is issued. The initial state, $s^{model}(t = 0)$, of the world also gathers the reward $r(s^{model}(t = 0))$.

While this restriction of the rewards to a function of state is not required, this reward scheme will be sufficient for synthesizing controllers that satisfy LTL formulas over POMDPs.

The final restriction is that the world model is assumed to be time invariant, i.e., $\mathcal{S}^{model}$, Act, $\mathcal{O}$, AP, T, O, h and r do not change with time.

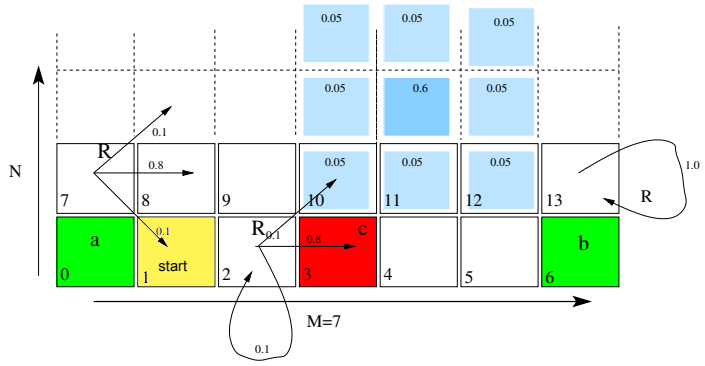


Figure 2.2: Partially observable probabilistic grid world. It is described in example 2.2.2.

Example 2.2.2 (GridWorld-A) An example whose variants will be used repeatedly to demonstrate various aspects of the problem requirements and solution methodology is the labeled-POMDP represented in Fig. 2.2. It represents a grid world of size $M \times N$, with fixed $M = 7$ and varying $N \geqslant 1$ in which a robot can move from cell to cell. Thus the state space is given by

$$\mathcal{S} = \{(s_i | i = x + My, x \in \{0, \ldots, M-1\}, y \in \{0, \ldots, N-1\}\}.$$

The action set available to the robot is

$$Act = \{Right,\ Left,\ Up,\ Down,\ Stop\}.$$

The actions $Right$, $Left$, Up and $Down$, cause probabilistic motion of the robot from its current cell to a neighboring cell. The probabilities for the state transitions for various types of cell (near a wall, or interior) are shown for action $Right$ in the figure. $Left$, Up, and $Down$ have symmetric definitions. The action $Stop$ is deterministic, in which the robot stays in its current cell. Partial observability arises because the robot does not precisely know its cell location. It can take measurements to ascertain it. The observation space is given by

$$\mathcal{O} = \{o_i | i = x + My, x \in \{0, \ldots, M-1\}, y \in \{0, \ldots, N-1\}\}.$$

Given the actual cell position (dark blue) of the robot, the location measurement has a distribution over the actual position and nearby cells (light blue). Cell 1 in yellow shows the initial state of the robot. Thus,

$$\iota_{init}(s_1) = 1.$$

In this example, the initial state is known exactly, but it is not required in general. Finally, there

are three atomic propositions of interest in this world, giving

$$AP = \{a, b, c\}.$$

In cell 0, a is true, while b and c are true in cells 6 and 3 respectively.

Definition 2.2.3 (Path in a POMDP) *An infinite* path *in a (labeled) POMDP, $\mathcal{PM}$, with states $s \in \mathcal{S}$ is an infinite sequence $\pi = s_0 o_0 \alpha_1 s_1 o_1 \alpha_2 \cdots \in (\mathcal{S} \times \mathcal{O} \times Act)^\omega$, such that $\forall t \geqslant 0$*

$$T(s_{t+1}|s_t, \alpha_{t+1}) > 0,$$

$$O(o_t|s_t) > 0, \text{ and}$$

$$\iota_{init}(s_0) > 0.$$

Any finite prefix of π that ends in either a state or an observation is a finite path fragment.

Pictorially, the process evolves as in Fig. 2.3. Even though the underlying states at each time step are not fully observable, the model is assumed to start at some state s_0. Associated with each state s_t, an observation o_t is emitted according to O and is available to the agent. Since the labels $h(.)$ are tied to the state, the labeling $h(s_t)$ is also partially observed at any time step t. Subsequently an action a_{t+1} causes the state of the world to evolve from s_t to s_{t+1}. Note that since no method for choosing actions has been introduced until now, the actions α_i are non-deterministic.

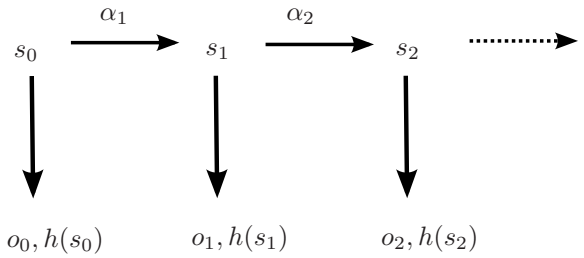


Figure 2.3: Evolution of a labeled POMDP.

The above formulation has minor differences from common formulations in literature, which assume that the first observation is available *after* the first action is applied. This is different from Figure 2.3, where the first action occurs after receiving an observation from the world, which can be used to further refine the initial distribution ι_{init}. Thus, if there is an agent that chooses actions, then it deliberately takes a measurement before taking the first action.

2.2.1 Information State Process (ISP) induced by the POMDP

Let I_t denote all information available until time step t. Since at any step the action taken and the observation received are the only two new quantities known to an agent, $I_{t+1} = [I_t, \alpha_{t+1}, o_{t+1}]$. This leads to the following information state process (ISP) $\mathcal{I} = \{I_0, I_1, \dots\}$:

$$\begin{aligned} I_0 &= [\iota_{init}, o_0] \\ I_1 &= [I_0, \alpha_1, o_1] \\ I_2 &= [I_1, \alpha_2, o_2] \\ &\vdots \end{aligned}$$

The domain of the ISP $\mathcal{I}$ can be defined inductively

$$\begin{aligned} dom(I_0) &= M_{\mathcal{S}} \times \mathcal{O} \\ dom(I_t) &= dom(I_{t-1}) \times Act \times \mathcal{O} \end{aligned}$$

so that the overall domain is

$$dom(\mathcal{I}) = \bigcup_t dom(I_t)$$

In the above, $M_{\mathcal{S}}$ is the set of all distributions over $\mathcal{S}$. In the context of transition systems, distributions are sometimes thought of as *beliefs* over the states $\mathcal{S}$, signifying the likelihood of actually being in different states at a given time step.

The formal definition of a distribution and its derivation from probability measures is discussed in appendix A.

2.2.2 Belief State Process (BSP) induced by the POMDP

Clearly, the representation of the ISP in computer memory requires an exponentially bounded ($(|Act||\mathcal{O}|^t)$) amount of space. Instead, a popular method of modeling the process to keep track of a sufficient statistic with finite description, such as the belief or distribution over states at each time step. Belief can be computed from the ISP using Bayes law as follows. $\forall s \in \mathcal{S}$

$$b_0(s) = \frac{\iota_{init}(s)O(o_0|s)}{\sum_{o\in\mathcal{O}} \iota_{init}(s)O(o|s)} \tag{2.1}$$

$$b_t(s) = \frac{1}{c}O(o_t|s,\alpha_t)\sum_{s'\in\mathcal{S}} T(s|s',\alpha_t)b_{t-1}(s'),\ t \geqslant 1 \tag{2.2}$$

where c is a normalizing constant given by

$$c = \sum_{s\in\mathcal{S}} O(o_t|s,\alpha_t)\sum_{s\in\mathcal{S}} T(s|s',\alpha_t)b_{t-1}(s'),\ t \geqslant 1.$$

Equation 2.2 is called the *belief update* equation. In fact, the belief state process (BSP) $\mathcal{B} = \{b_0, b_1, \dots\}$ forms a continuous state space (fully observable) Markov decision process, evolving in the space of all distributions $M_{\mathcal{S}}$ as defined in Section 2.2.1. At any time step t, the belief state b_t can be computed from the information state I_t inductively.

2.2.3 POMDP controllers

From the definition of a (labeled) POMDP, and the associated ISP or BSP, it is unclear how actions at each time step are chosen. This is the task of the *agent*, used interchangeably with the *controller* or *scheduler*. The controller chooses the actions α_t using a *policy* which formalizes the rules used to decide actions using some or all information available to the controller up to the current time step. Typically, policies are categorized into deterministic and stochastic policies.

Definition 2.2.4 (Deterministic Policy) *Let $\mathcal{PM}$ be a POMDP, and its ISP given by $\mathcal{I}$. A* deterministic policy *for $\mathcal{PM}$, denoted μ is function*

$$\mu : dom(\mathcal{I}) \to Act \tag{2.3}$$

Definition 2.2.5 (Stochastic Policy) *Let $\mathcal{PM}$ be a POMDP, and its ISP given by $\mathcal{I}$. A* stochastic policy *for $\mathcal{PM}$, denoted μ is a function*

$$\mu : dom(\mathcal{I}) \to M_{Act} \tag{2.4}$$

where M_{Act} is the set of all probability distributions over the natural $\sigma-$algebra over Act.

For deterministic policies, actions are directly determined as $\alpha_t = \mu_t \triangleq \mu(I_{t-1})$. For stochastic policies $\mu(I_{t-1})$ is a distribution that must be sampled independently to generate the action α_t and this should be written as $\alpha_t \sim \mu_t = \mu(I_{t-1})$. However, the equality $\alpha_t = \mu_t$ will be used to represent both of these cases, and the methodology for obtaining α_t will be inferred from context.

Most importantly, the outcome of this definition of a policy is that the ISP $\mathcal{I} = \{I_0, I_1, \dots\}$ is Markovian.

2.2.3.1 Markov Chain Induced by a Policy

Given a POMDP $\mathcal{PM}$ with state space $\mathcal{S}$, a $t-$step execution is given by

$$\sigma_{0:t} = s_0 s_1 \dots s_t \in \underbrace{\mathcal{S} \times \dots \mathcal{S}}_{t+1 \text{ times}} = \mathcal{S}^{t+1} \tag{2.5}$$

Then the process $\mathcal{M}^{\mu}_{\mathcal{S}^+} = \{\sigma_{0:0}, \sigma_{0:1}, \dots\}$ evolves stochastically in the space

$$\mathcal{S}^+ = \bigcup_{t=0}^{\infty} \mathcal{S}^{t+1}. \tag{2.6}$$

The initial probability distribution is given by

$$\Pr[\sigma_{0:0}] = b_0(s_0), \tag{2.7}$$

and the state transition probabilities are given by

$$T(\sigma_{0:t+1}|\sigma_{o:t}) = T(s_{t+1}|s_t, \mu(I_t)). \tag{2.8}$$

Since I_t itself is Markovian, the process $\mathcal{M}^{\mu}_{\mathcal{S}^+}$ forms a Markov chain. However, this Markov chain has infinite state space, $\mathcal{S}^+$. In Section 3.1.1, it will be shown that a class of controllers leads to a Markov chain over a finite state space that is equivalent to $\mathcal{M}^{\mu}_{\mathcal{S}^+}$.

2.2.4 Probability Space over Markov Chains

Let $\mathcal{M}$ be a Markov chain over a countable set of states $\mathcal{S}$. Let the initial state distribution be given by $\iota_{init}(s),\ s \in \mathcal{S}$ and the state transition probability given by $T(s_j|s_i)$. In this work, the notion of probability of events in a Markov chain is used extensively. It is therefore necessary to formally define the probability space associated with $\mathcal{M}$. A detailed description can be found in [8] from where the following definitions are borrowed. A brief primer on probability measures is provided in appendix A. Three quantities are of interest to fully specify a probability space $(X, \mathcal{F}, \mu)$, where X is the underlying set of outcomes, $\mathcal{F} \subseteq 2^X$ is a $\sigma-$algebra over X and μ is a probability measure over the sets in $\mathcal{F}$. For infinite executions of Markov chains, i.e., $t \to \infty$, each of these is defined next.

The underlying set of outcomes X is given by the set of (infinite) paths, $Paths(\mathcal{M})$, of the Markov chain, $\mathcal{M}$. The formal definition of paths is given below.

Definition 2.2.6 (Paths of a Markov chain) *Define as* $Paths(\mathcal{M})$ *the set of all infinite sequences* $\pi = s_0 s_1 \cdots \in S^\omega$, *such that* $\forall t \geqslant 0,\ T(s_{t+1}|s_t) > 0$ *and* $\iota_{init}(s_0) > 0$. *In addition, also define the set of all finite path fragments* $Paths_{fin}(\mathcal{M}) = \{prefix(\pi)|\pi \in Paths(\mathcal{M})\}$, *i.e., by collecting all finite prefixes from every infinite path* $\pi \in Paths(\mathcal{M})$.

Next, in order to specify the $\sigma-$ algebra, $\mathcal{F}$, of concern, define the cylinder sets of the Markov chain as follows.

Definition 2.2.7 (Cylinder Set) *The* cylinder set *of* $\hat{\pi} = s_0 s_1 \ldots s_n \in Paths_{fin}(\mathcal{M})$ *is defined as*

$$Cyl(\hat{\pi}) = \{\pi \in Paths(\mathcal{M}) \mid \hat{\pi} \ \in \ prefix(\pi)\}. \tag{2.9}$$

In other words, the cylinder set spanned by finite path fragment $\hat{\pi}$ is the set of all infinite paths that start with $\hat{\pi}$. Next, denote the set of all possible cylinder sets as CYL. Formally

$$CYL = \{Cyl(\hat{\pi})|\hat{\pi} \in Paths_{fin}(\mathcal{M})\}. \tag{2.10}$$

Clearly, $CYL \in 2^X$. As stated in Section A.5, there exists a (unique) smallest $\sigma-$algebra that contains CYL, denoted $\sigma(CYL)$. This gives the $\mathcal{F} = \sigma(CYL)$, where the *basis* events are given by $Cyl(\hat{\pi})$.

Finally, in order to specify the probability measure over all the sets or events in $\mathcal{F} = \sigma(CYL)$, it is sufficient to provide the probability of each cylinder set in CYL. This is computed as

$$\Pr{}_{\mathcal{M}}\left[Cyl(s_0 \ldots s_n)\right] = \iota_{init}(s_0) \prod_{0 \leqslant t < n} T(s_{t+1}|s_t). \tag{2.11}$$

Once the probability measure is defined over the cylinder sets, the expectation operator $\mathbb{E}_{\mathcal{M}}$ is also uniquely defined from its definition. In the sequel, if the underlying Markov chain is clear from context, the subscript will be dropped and Pr and $\mathbb{E}$ will be used.

2.2.5 Typical Problems over POMDPs

In a typical POMDP control problem, states are assumed to issue rewards as the agent influences the world to visit various states. For a finite execution $\sigma_{0:T} = s_0 s_1 \ldots s_T$, the cumulative reward received by the agent is given by $\sum_{t=0}^{T} r(t)$. Another quantity of interest is the average reward per time step $\frac{1}{T}\sum_{t=0}^{T} r(t)$ especially when $T \to \infty$. Many problems also assume that good events visited early in the execution are better. In order to incentivize temporal greediness, the accumulated reward is formulated as the discounted sum $\sum_{t=0}^{T} \beta^t r(t)$, with $0 < \beta < 1$.

Note that even if the policy used by the agent is deterministic, the controlled system is still a Markov chain due to the probabilistic world state transitions and observations. Therefore, the above

quantities of interest are maximized in *expectation*.

Two main maximization objectives are of interest in this work, and are studied widely in literature.

Definition 2.2.8 (Total Expected Discounted Reward) *For a finite horizon or time steps T, this criterion is given by*

$$\eta_\beta(T) = \mathbb{E}_{\mathcal{M}_\mu}\left[\sum_{t=0}^{T}\beta^t r(t)\middle|\iota_{init}\right],\ \text{for } 0<\beta<1, \tag{2.12}$$

where β is the discount factor. For infinite horizon, the criterion is given by the limit

$$\eta_\beta = \lim_{T\to\infty}\mathbb{E}_{\mathcal{M}_\mu}\left[\sum_{t=0}^{T}\beta^t r(t)\middle|\iota_{init}\right],\ \text{for } 0<\beta<1. \tag{2.13}$$

Definition 2.2.9 (Expected Average Reward) *For a finite horizon or time steps T, this criterion is given by*

$$\eta_{av}(T) = \mathbb{E}_{\mathcal{M}_\mu}\left[\frac{1}{T}\sum_{t=0}^{T} r(t)\middle|\iota_{init}\right]. \tag{2.14}$$

For infinite horizon, the criterion is given by the limit

$$\eta_{av} = \lim_{T\to\infty}\mathbb{E}_{\mathcal{M}_\mu}\left[\frac{1}{T}\sum_{t=0}^{T} r(t)\middle|\iota_{init}\right]. \tag{2.15}$$

Let η be one of the above criteria; then the optimization problem of interest in the classical POMDP control setting is

$$\underset{\mu}{\text{maximize}} \quad \eta. \tag{2.16}$$

2.2.6 Optimal and $\epsilon-$optimal Policy

Assume that an optimal value η^* of a given objective from the various reward criteria exists. Then any policy μ^* that attains this optimum is called an *optimal* policy. Let η^μ be the value of the objective under some known policy μ. For any $\epsilon > 0$, μ is called ϵ-optimal if

$$\eta^* \geqslant \eta^\mu \geqslant \eta^* - \epsilon, \tag{2.17}$$

thus signifying that the value of the objective is at most ϵ below the optimal value under the policy μ.

2.2.7 Brief overview of solution methods

The solution methods for typical POMDP problems fall under two categories: *exact* and *approximate*. A detailed survey is found in the introductory chapters of [1], and in [24]. These are briefly summarized here.

2.2.7.1 Exact Methods

Exact methods typically need to employ the full information state I_t to design the policy. However this requires infinite memory as $t \to \infty$ and is intractable. As has already been mentioned, the belief state b_t offers a sufficient statistic for I_t . Following [29, 137], the problem is solved using Dynamic Programming [16] over a belief space MDP that can be constructed from a POMDP. However, since the belief state space is uncountably infinite, these algorithms may require infinite memory for representation. Also the complexity of these algorithms grows exponentially with the size of the state space, and hence it is difficult to solve problems with more than a few tens of states, observations and actions.

2.2.7.2 Approximate Methods

In recent years, there has been a lot of work in approximating the value function. For example, one method is to assume that the underlying system is an MDP and learning the underlying Q-function and employing heuristics such as the most likely state heuristic, the voting heuristic, Q_{MDP}-heuristic, or exploiting the entropy of the belief state [67,106,135]. Another is to use grid based methods [22,55]. Other approximate methods search only over the reachable belief states and fall under point-based POMDP planning [79,113]. The algorithms in this thesis also utilize an approximate method because they search exclusively in the space of policies that require finite internal memory. This particular controller, called a Finite State Controller is introduced in detail in the next chapter in Section 3.1.

2.3 Concluding Remarks

This chapter laid down the foundation for the rest of the thesis. Mathematical formalisms for LTL, the goal specification language of concern, and the POMDP model class were provided. An automata based formal verification technique was also introduced for general transition systems, which will be used in the remainder of the thesis for control synthesis. Moreover, a brief overview of the typical optimization problems over POMDP models and solution approaches were outlined. The algorithms proposed in this thesis will utilize the methods used to solve these optimization objectives to synthesize controllers for LTL specifications. In the next chapter, automata based LTL verification of over POMDPs will be formally described with focus on a specific type of POMDP

controller, namely the Finite State Controller. This will be used in the rest of the thesis to propose methods for finding such controllers given a POMDP model and LTL specification.

Chapter 3

LTL Satisfaction using Finite State Controllers

In the previous chapter, the POMDP and its associated problems in the form of optimization objectives were introduced. Further, the various methods of designing controllers to maximize these criteria were mentioned. In this chapter, one particular class of controllers, called the finite state controller is studied in detail. The choice of this class of controllers is shown to lead to a finite state space Markov chain for the closed loop controlled system. This allows easy analysis of infinite executions of the system in the context of satisfying an LTL formula of interest. Next, the various categories of problems relating to LTL formulas over POMDPs controlled by FSCs are formalized. Finally, a brief overview of the solution methodology for these problems is provided.

It is a well known fact that POMDP, and for some criteria, MDP controllers require memory or internal states [1,29,67]. Let the controller's internal states be denoted by $g \in G = \{g_1, g_2, \ldots, g_{|G|}\}$. *Finite state controllers* have finite $|G|$. As mentioned before, infinite horizon problems typically require infinite $|G|$. The most popular method that employs infinite memory design controllers that work in the belief space which is continuous, which effectively implies uncountably infinite G. For this case the above definition does not hold.

Finite state controllers are formally defined next.

3.1 Finite State Controllers

Definition 3.1.1 (Deterministic Finite State Controller (det-FSC)) *Let $\mathcal{PM}$ be a POMDP with observation set $\mathcal{O}$, action set Act and initial distribution ι_{init} as in Definition 2.2.1s. A* deterministic finite state controller (det-FSC) *for $\mathcal{PM}$ is given by the tuple $\mathcal{G} = (G, \omega, \kappa)$ where*

- *$G = \{g_1, g_2, \ldots, g_{|G|}\}$ is a finite set of internal states.*
- *$\omega : G \times \mathcal{O} \to G \times Act$ is a function such that given a current internal FSC state g_k, and observation o, $(g_l, \alpha) = \omega(g_k, o)$ chooses the next internal state of the FSC and the action to*

apply to $\mathcal{PM}$.

- $\kappa : M_{\mathcal{S}} \rightarrow G$ *chooses the start state* $g_0 = \kappa(\iota_{init})$*, of the FSC given initial distribution* ι_{init}*.*

Definition 3.1.2 (Stochastic Finite State Controller (sto-FSC)) *Let $\mathcal{PM}$ be a POMDP with observation set $\mathcal{O}$, action set Act and initial distribution ι_{init}. A* stochastic finite state controller (sto-FSC) *for $\mathcal{PM}$ is given by the tuple $\mathcal{G} = (G, \omega, \kappa)$ where*

- $G = \{g_1, g_2, \ldots, g_{|G|}\}$ *is a finite set of internal states.*
- $\omega : G \times \mathcal{O} \rightarrow M_{G \times Act}$ *is a function such that given a current internal state of FSC, g_k and observation o, $\omega(g_k, o)$ is a probability distribution over $G \times Act$. The next internal state and action pair (g_l, α) are chosen by independent sampling of $\omega(g_k, o)$. By abuse of notation, we will use $\omega(g_l, \alpha | g_k, o)$ as the probability of transitioning to I-state g_l and taking action α, when the current I-state is g_k and observation received is o.*
- $\kappa : M_{\mathcal{S}} \rightarrow M_G$ *chooses the starting internal state g_0, by independent sampling of $\kappa(\iota_{init})$, given initial distribution ι_{init} of $\mathcal{PM}$. Again, by abuse of notation, we will use $\kappa(g|\iota_{init})$ to denote the probability of starting the FSC in internal state g when the initial distribution is given by ι_{init}.*

Any deterministic FSC can be written as a special case of stochastic FSCs. This thesis will exclusively consider the stochastic version and so the term FSC will denote a stochastic FSC unless otherwise stated.

A schematic diagram of how an FSC controls the POMDP is shown in Figure 3.1. Under the FSC, the POMDP evolves as follows.

1. Set $t = 0$. POMDP initial state s_0 is initialized by drawing independently from the distribution ι_{init}. The deterministic or stochastic function $\kappa(\iota_{init})$ is used to determine or sample the initial FSC I-state g_0.
2. At each time step $t \geqslant 0$, the POMDP emits an observation o_t according to the distribution $O(.|s_t)$.
3. The FSC determines its new state g_{t+1} and action α_{t+1} according to the deterministic function or stochastic distribution given by $\omega(.|g_t, o_t)$.
4. The action α_{t+1} is applied to the POMDP, which transitions to a new state s_{t+1} according to the distribution $T(.|s_t, \alpha)$.
5. $t = t + 1$, Go to 2.

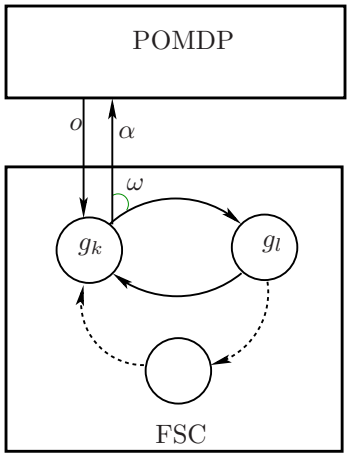


Figure 3.1: POMDP controlled by an FSC

3.1.1 Markov Chain induced by an FSC

Closing the loop around a POMDP with an FSC, as in Figure 3.1, yields the following transition system.

Definition 3.1.3 (Global Markov Chain) *Let $\mathcal{S}$ be the state space of the POMDP, $\mathcal{PM}$, and G be the set of I-states of the FSC, $\mathcal{G}$, as in Definition 3.1.2. The global Markov chain $\mathcal{M}_{\mathcal{S}\times G}^{\mathcal{PM},\mathcal{G}}$ with execution $\sigma = \{[s_0, g_0], [s_1, g_1], \dots\}$, $[s_t,\ g_t] \in \mathcal{S} \times G$ evolves as follows:*

- *The probability of the initial global state $[s_0, g_0]$ is given by*

$$\iota_{init}^{\mathcal{PM},\mathcal{G}}\left[[s_0, g_0]\right] = \iota_{init}(s_0)\kappa(g_0|\iota_{init}) \tag{3.1}$$

- *The state transition probability is given by*

$$T^{\mathcal{PM},\mathcal{G}}\left[[s_{t+1}, g_{t+1}]\,|[s_t, g_t]\right] = \sum_{o\in\mathcal{O}}\sum_{\alpha\in Act} O(o|s_t)\omega(g_{t+1}, \alpha|g_t, o)T(s_{t+1}|s_t, \alpha) \tag{3.2}$$

Note that for a finite state space POMDP, the global Markov chain has finite state space. Similar to the fully observable case of Markov decision process in [8], the global Markov chain induced by the finite state controller $\mathcal{M}_{\mathcal{S}\times\mathcal{G}}^{\mathcal{PM},\mathcal{G}}$ is probabilistically bisimilar to the infinite state space Markov chain described in Section 2.2.3.1. Probabilistic bisimilarity is discussed in [8, 80].

We also remind the reader, that this Markov chain is also associated with a probability space as described in Section 2.2.4.

3.2 LTL satisfaction over POMDP executions

This section formalizes how infinite traces obtained from POMDP executions can be verified against an LTL formula φ. This is carried out by constructing the product of the labeled POMDP, $\mathcal{PM}$, and the DRA obtained from translating φ.

Definition 3.2.1 (Product-POMDP) *Let $\mathcal{PM}$ be a labeled POMDP (Definition 2.2.1) with state space $\mathcal{S}^{model}$, actions Act, observations $\mathcal{O}$, atomic propositions AP, transition probabilities $T(.|s,\alpha) : S \to [0,1],\ \forall s^{model} \in \mathcal{S}^{model}, \alpha \in Act$, and labeling function $h : \mathcal{S}^{model} \to 2^{AP}$.*

Next, let the LTL formula φ be translated to the deterministic Rabin automaton (Definition 2.1.4), denoted $\mathcal{A}^{\varphi}$, with state space Q, initial state q_0, input alphabet 2^{AP}, transition function $\delta : Q \times 2^{AP} \to Q$ and with $\Omega = (Repeat_i, Avoid_i)$ s.t. $Repeat_i \subseteq Q, Avoid_i \subseteq Q$, the Rabin acceptance condition.

The product-POMDP *denoted $\mathcal{PM}^{\varphi}$ is a POMDP with state space $\mathcal{S} = \mathcal{S}^{model} \times Q$, the same action set Act and observations $\mathcal{O}$.*

- *The transition probabilities of $\mathcal{PM}^{\varphi}$ are given by*

$$T^{\varphi}\left(s' = \langle s_j^{model}, q_l\rangle | s = \langle s_i^{model}, q_k\rangle, \alpha\right) = \begin{cases} T(s_j^{model}|s_i^{model},\alpha) & \text{if } \delta(q_k, h(s_i^{model})) = q_l \\ 0 & \text{otherwise.} \end{cases} \tag{3.3}$$

- *The initial state probability distribution is given by*

$$\iota_{init}^{\varphi}\left(s = \langle s^{model}, q\rangle\right) = \begin{cases} \iota_{init}(s^{model}) & \text{if } \delta(q_0, h(s^{model})) = q \\ 0 & \text{otherwise.} \end{cases} \tag{3.4}$$

- *The observation probabilities are*

$$\mathcal{O}^{\varphi}(o|s = \langle s^{model}, q\rangle) = \mathcal{O}(o|s^{model}). \tag{3.5}$$

- *If rewards $r(s^{model})$ are defined over the original model $\mathcal{PM}$, then we define new rewards over the product states are defined as*

$$r^{\varphi}(s = \langle s^{model}, q\rangle) = r(s^{model}). \tag{3.6}$$

In addition, using the Rabin acceptance pairs Ω from $\mathcal{A}^{\varphi}$, we also define the accepting pairs $\Omega^{\mathcal{PM}^{\varphi}} = \{(Repeat_i^{\mathcal{PM}^{\varphi}}, Avoid_i^{\mathcal{PM}^{\varphi}}), 0 \leqslant i \leqslant |\Omega|\}$ for the product-POMDP as follows. A product

state $s = \langle s^{model}, q \rangle$ *of* $\mathcal{PM}^{\varphi}$ *is in* $Repeat_i^{\mathcal{PM}^{\varphi}}$ *iff* $q \in Repeat_i$ *and* $s = \langle s^{model}, q \rangle$ *is in* $Avoid_i^{\mathcal{PM}^{\varphi}}$ *iff* $q \in Avoid_i$. *Note that* $|\Omega^{\mathcal{PM}^{\varphi}}| = |\Omega|$.

3.2.1 Inducing an FSC for $\mathcal{PM}$ from that of $\mathcal{PM}^{\varphi}$

The product POMDP is the basis for a method to find a policy as defined by an FSC. However, the resulting policy is meant to be applied to the Product-POMDP. However the goal is to control the original POMDP, $\mathcal{PM}$ and it is necessary to formally define how to derive a policy for $\mathcal{PM}$ from a policy computed for $\mathcal{PM}^{\varphi}$.

Note that in the case of fully observable MDPs, the choice of action as a function of the current state of the Product-POMDP is usually sufficient [8]. In this case a policy $\alpha = \mu(s = \langle s^{model}, q \rangle)$ for the product-MDP induces the policy for the original MDP by setting $\alpha = \mu^{model}(s^{model}) = \mu(s = \langle s^{model}, q \rangle)$ at each time step [40].

In the case of POMDPs, given the internal state of the controller, the only new information at each time step is the most recent action and observation. These action and observation sets remain unchanged between the original model $\mathcal{PM}$ and the product-POMDP $\mathcal{PM}^{\varphi}$. However, the start state of the FSC is determined by the initial distribution of the product-POMDP.

Definition 3.2.2 (Inducing an FSC) *Let* $\mathcal{G} = (G, \kappa, \omega)$ *be the FSC that controls the product-POMDP* $\mathcal{PM}^{\varphi}$. *The FSC* $\mathcal{G}^{model} = (G^{model}, \kappa^{model}, \omega^{model})$ *that controls the original POMDP* $\mathcal{PM}$ *is induced as follows.*

- *The internal nodes of* $\mathcal{G}^{model}$ *are the same as that of* $\mathcal{G}$. *That is,* $G^{model} = G$.
- *Let* $\kappa(\iota_{init}^{\varphi})$ *be the distribution that determines the initial node of* $\mathcal{G}$. *Then the initial node of* $\mathcal{G}^{model}$ *is given by setting*

$$\kappa^{model}(\iota_{init}) = \kappa(\iota_{init}^{\varphi}). \tag{3.7}$$

- *The probability of transitioning between I-states and issuing an action* α *is obtained by*

$$\omega^{model}(g_l, \alpha | g_k, o) = \omega(g_l, \alpha | g_k, o), \ \forall o \in \mathcal{O}, \alpha \in Act, \text{and } g_k, g_l \in G = G^{model}. \tag{3.8}$$

3.2.2 Verification of LTL Satisfaction using Product-POMDP

Now let's consider the criterion for an (infinite) execution of $\mathcal{PM}$ to satisfy φ. Let $\sigma^{\varphi} = s_0 s_1 \ldots$, $s_t = \langle s_t^{model}, q_t \rangle$ be an execution of the product-POMDP under some FSC $\mathcal{G}$.

Definition 3.2.3 (Accepting execution) *We say that σ^{φ} is an* accepting execution *if, for some $(Repeat_i^{\mathcal{PM}^{\varphi}}, Avoid_i^{\mathcal{PM}^{\varphi}}) \in \Omega^{\mathcal{PM}^{\varphi}}$ such that σ^{φ} intersects with $Repeat_i^{\mathcal{PM}^{\varphi}}$ infinitely often, while it intersects with $Avoid_i^{\mathcal{PM}^{\varphi}}$ only a finite number of times.*

The notion of verifying LTL properties using product transition systems in well known in the literature [8, 40] and the following lemma is stated without a formal proof.

Lemma 3.2.4 *Let $\sigma^{\varphi} = s_0 s_1 \ldots$, with $s_t = \langle s_t^{model}, q_t \rangle$ be an execution of $\mathcal{PM}^{\varphi}$ and the corresponding execution of $\mathcal{PM}$ be given by $\sigma = s_0^{model} s_1^{model} \ldots$. Then, σ satisfies φ, i.e., $\sigma \models \varphi$, if and only if $\pi = q_0 q_1 \ldots$ is an accepting run on $\mathcal{A}^{\varphi}$.*

Lemma 3.2.4 can be explained by understanding the construction of the product-POMDP. Note that the run σ^{φ} can be projected onto its POMDP and DRA components as runs σ and π. Next, the trace generated by σ, given by $h(\sigma) = h(s_0^{model})h(s_1^{model})\ldots$ leads to the same unique path π in the DRA $\mathcal{A}^{\varphi}$. Thus if π is an accepting run in the DRA, then $\sigma \models \varphi$.

3.2.3 Measuring the Probability of Satisfaction of LTL Formulas

The above section described how the *accepting executions* of the product-POMDP, $\mathcal{PM}^{\varphi}$, under a given controller, have a one-to-one correspondence to those executions of the original POMDP, $\mathcal{PM}$, that satisfy φ.

Recall from Section 3.1.1 that if an FSC is used to control a POMDP, executions correspond to *paths* in the induced global Markov chain. For the product-POMDP, $\mathcal{PM}^{\varphi}$, controlled by FSC, $\mathcal{G}$, let the induced Markov chain be denoted as $\mathcal{M}_{\mathcal{S}\times G}^{\mathcal{PM}^{\varphi},\mathcal{G}}$, evolving on the finite state space $\mathcal{S} \times G = (\mathcal{S}^{model} \times Q) \times G$. Also recall from Section 2.2.4 that a probability measure can be defined over the paths of the global Markov chain. The *probability of satisfaction* of φ over the controlled (closed loop) system is defined below.

Definition 3.2.5 (Probability of satisfaction of φ) *For the product-POMDP $\mathcal{PM}^{\varphi}$ controlled by an FSC $\mathcal{G}$, the* probability of satisfaction of φ, *defined over $Paths(\mathcal{M}_{\mathcal{S}\times\mathcal{G}}^{\mathcal{PM}^{\varphi},\mathcal{G}})$, is given by*

$$\Pr(\mathcal{PM}^{\varphi} \models \varphi | \mathcal{G}) = \Pr_{\mathcal{M}_{\mathcal{S}\times G}^{\mathcal{PM}^{\varphi},\mathcal{G}}} \left[\sigma^{global} \in Paths(\mathcal{M}_{\mathcal{S}\times G}^{\mathcal{PM}^{\varphi},\mathcal{G}}) \,\middle|\, \perp_{\mathcal{S}} (\sigma^{global}) \text{ is an accepting execution} \right]. \tag{3.9}$$

The various terms appearing in Equation 3.9 are as follows. $\mathcal{M}_{\mathcal{S}\times\mathcal{G}}^{\mathcal{PM}^{\varphi},\mathcal{G}}$ is the global Markov chain induced by the Product-POMDP $\mathcal{PM}^{\varphi}$ with state space $\mathcal{S} = \mathcal{S}^{model} \times Q$ controlled by the FSC $\mathcal{G}$ with I-states G. Next,

$$\begin{aligned} \sigma^{global} &= [s_0, g_0][s_1, g_1]\ldots \\ &= \left[\langle s_0^{model}, q_0\rangle, g_0\right]\left[\langle s_1^{model}, q_1\rangle, g_1\right]\ldots \end{aligned}$$

is a path of the induced Markov chain, $\mathcal{M}_{\mathcal{S}\times\mathcal{G}}^{\mathcal{PM}^{\varphi},\mathcal{G}}$*, and* $\perp_{\mathcal{S}}(.)$ *is a projection operator defined as follows.*

$$\perp_{\mathcal{S}}(\sigma^{global}) = \langle s_0^{model}, q_0\rangle\langle s_1^{model}, q_1\rangle\ldots$$

that has the effect of extracting the Product-POMDP execution from the path in the global Markov chain.

Since the definition of the global Markov chain is unique, hereafter the subscript on the probability operator in the r.h.s. of Equation (3.9) will be dropped. Similarly, any use of the expectation operator $\mathbb{E}$ will also be defined using the underlying probability measure over the global Markov chain.

Finally, define $\Pr(\mathcal{PM} \models \varphi|\mathcal{G}) \triangleq \Pr(\mathcal{PM}^{\varphi} \models \varphi|\mathcal{G})$ as the probability of the original model satisfying the LTL formula.

3.3 Background: Analysis of Probabilistic Satisfaction

Typical formal verification or model checking problems focus on two different aspects of the transition system with respect to its environment: qualitative verification with respect to specification φ and quantitative verification with respect to φ.

3.3.1 Qualitative Problems

There are two main problems studied under qualitative model checking:

1. Almost Sure Satisfaction: This analysis verifies if

$$\Pr(\mathcal{PM} \models \varphi) = 1, \text{ almost surely (a. s.).} \tag{3.10}$$

The result sought is a binary "yes" or "no" answer, the former denoting that under the given controller, the specification or LTL formula holds with probability 1.

2. Positive Satisfaction: This analysis verifies if

$$\Pr(\mathcal{PM} \models \varphi) > 0, \text{ a. s..} \tag{3.11}$$

Again, a "yes" or "no" answer signifies if there is a finite probability that the LTL specification is satisfied. This analysis can be useful to determine if there is a positive probability of unsafe behavior, or the existence of a control policy to satisfy the specification. It is analogous to feasibility problems in control. However, in practice this analysis is carried out implicitly while minimizing or maximizing $\Pr(\mathcal{PM} \models \varphi)$.

	Almost Sure Satisfaction		Positive Satisfaction Maximizing Satisfaction Prob	
	Inf. Mem	Fin. Mem	Inf. Mem	Fin. Mem
Strategy Computation	Infinite Undecidable	Exponential EXP-Complete	Exponential EXP-Complete	Exponential EXP-Complete

Table 3.1: Complexity of control strategy and computational burden for analysis of LTL specifications over POMDPs. Finite and infinite memory strategies are shown for various classes of problems.

3.3.2 Quantitative Problems

In this subset of problems, there are two main topics of concern.

1. Computing Satisfaction Probability: The goal is to compute the probability of satisfying an LTL specification φ given a controller:

$$\textbf{compute: } \Pr(\mathcal{PM} \models \varphi | \mathcal{G}). \tag{3.12}$$

2. Maximizing Satisfaction Probability: One can pose the Product-POMDP control synthesis problem, as the following optimization problem:

$$\underset{\mathcal{G}}{\textbf{maximize}} \ \Pr(\mathcal{PM} \models \varphi | \mathcal{G}). \tag{3.13}$$

In general, maximizing the satisfaction probability generates the optimal value which can provide answers to the problems described above. One also seeks to find the FSC parameters that optimizes the satisfaction probability so that an agent can implement the controller to interact with the environment optimally with regard to the LTL specification. Out of the problems mention here, this thesis focuses on developing algorithms for maximizing the satisfaction probability as captured by Equation (3.13).

3.3.3 Complexity of Solution for Qualitative and Quantitative Problems

For POMDPs, the complexity results for finding controllers for general ω-regular specifications can be found in [**?**, 32, 33]. There are two complexities of concern. First, the computational complexity for solving the qualitative and quantitative problems described earlier are of concern. Second, is the complexity of the control strategy or policy itself, since it must be represented in a computer system for application. It is also useful to contrast the complexity of finite memory strategies, which FSCs satisfy, and infinite memory strategies which utilize the belief space. These are summarized in Table 3.1.

3.4 Excursus on Markov chains

Before proposing algorithms to solve the quantitative problems mentioned above, note that LTL formulas are verified over infinite executions of the POMDP, with the concern that certain states are visited infinitely often while others are avoided completely after a finite number of execution steps. Since a finite state controller leads to a finite state space Markov chain when controlling a POMDP, the long term (steady state) behavior of finite Markov chains is the key to synthesizing controllers that satisfy LTL formulas. This observation also relates to the qualitative questions posed in Section 3.3.1. On the other hand, the short term (transient) behavior will be crucial to the analysis of the quantitative problems in Section 3.3.2.

In this section, a few well known properties of Markov chains, especially those with finite state space, are reviewed. A full mathematical background can be found in [56, 71, 101, 126, 127]. In the following, I will consider a Markov chain $\mathcal{M}$ with state space $\mathcal{S}$, transition probability defined as the conditional distribution $T(.|s) : \mathcal{S} \to [0.1]$ such that

$$\sum_{s' \in \mathcal{S}} T(s'|s) = 1, \ \forall s \in \mathcal{S},$$

and the initial distribution ι_{init} such that

$$\sum_{s \in \mathcal{S}} \iota_{init}(s) = 1.$$

All probabilities and expectation operators assume the underlying probability space associated with paths $\pi = s_0 s_1 \ldots$, as described in Section 2.2.4, via cylinder sets.

Note that the transition probabilities T form a linear operator which can be represented as a matrix. Henceforth T will denote both the individual conditional distributions and the overall matrix representation. The conditional distribution form will be inferred when they appear to take explicit arguments, usually denoted as $T(.|.,.)$. The matrix form will be inferred when expressions involving vectors and matrices are encountered.

$$T := \begin{bmatrix} T_{11} & T_{12} & \ldots & T_{1|\mathcal{S}|} \\ T_{21} & T_{22} & \ldots & T_{2|\mathcal{S}|} \\ \vdots & & \ddots & \vdots \\ T_{|\mathcal{S}|1} & T_{|\mathcal{S}|2} & \ldots & T_{|\mathcal{S}||\mathcal{S}|} \end{bmatrix} : M_{\mathcal{S}} \to M_{\mathcal{S}} \tag{3.14}$$

where

$$T_{ij} = T(s_j|s_i).$$

Next, a distribution or belief $\vec{b}_t$ over states of the Markov chain at some time t can be written

as a row vector

$$\vec{b}_t = \Big(\; b_t(s_1) \;\; b_t(s_2) \;\; \ldots \;\; b_t(s_{|\mathcal{S}|}) \; \Big). \tag{3.15}$$

Then the operator T maps b_t to the state distribution or belief b_{t+1} at time $t+1$. Using matrix representation this is $\vec{b}_{t+1} = \vec{b}_t T$.

Definition 3.4.1 (Occupation Time, First Return Time and Return Probability) *Let $\pi = s_0 s_1 \ldots$ be a path in the Markov chain and $A \subseteq \mathcal{S}$,*

(a) The variable

$$f_A := \sum_{t=1}^{\infty} \mathbb{1}(s_t \in A) \tag{3.16}$$

is called the occupation time *of set A and*

$$\mathbb{1}(\phi) = \begin{cases} 1 & \textit{the mathematical statement } \phi \textit{ holds.} \\ 0 & \textit{otherwise,} \end{cases} \tag{3.17}$$

is the indicator function. Thus f_A counts the number of times the set A is visited after time step 0.

(b) Next, the variable

$$\tau_A := \min\{t \geqslant 1 | s_t \in A\} \tag{3.18}$$

is called the first return time*, denoting the first time after time 0 that set A is visited.*

(c) Lastly, define

$$L(s, A) := \Pr(\tau_A < \infty | s_0 = s) \tag{3.19}$$

as the return probability*. It denotes the probability of set A being visited in finite time when the start state is s.*

By abuse of notation, when A is a singleton set, i.e, $A = \{s'\}$ for some $s' \in \mathcal{S}$, then $f_{s'}$, $\tau_{s'}$ and $L(s, s')$ will respectively denote the occupation time, first return time and return probability.

Definition 3.4.2 (Communicating Classes) *The state $s \in \mathcal{S}$ is said to* lead to *state $s' \in \mathcal{S}$, denoted $s \to s'$, if $L(s, s') > 0$. By convention $s \to s$. Next, distinct states s, s' are said to* communicate*, denoted $s \leftrightarrow s'$ when $L(s, s') > 0$ and $L(s', s) > 0$. Moreover, the relation "$\leftrightarrow$" is an equivalence relation, and equivalence classes $C(s) = s' : s \leftrightarrow s'$ cover $\mathcal{S}$, with $s \in C(s)$ [56].*

Definition 3.4.3 (Irreducibility and Absorbing Sets) *If $C(s) = S$ for some $s \in \mathcal{S}$, then the Markov chain, $\mathcal{M}$, is called* irreducible*. This means that all states communicate. In addition, $C(s)$*

is absorbing *if*

$$\sum_{s'' \in C(s)} T(s''|s') = 1 \ \forall s' \in C(s). \tag{3.20}$$

Definition 3.4.4 (Restriction of $\mathcal{M}$ to an Absorbing Set) *Let $C \subseteq \mathcal{S}$ be an absorbing set. Then by Definition 3.4.3, if the initial state s_0 lies in C, then for any path $\pi = s_0 s_1 \ldots$, s_t lies in C for all $t \geqslant 0$. Hence, the Markov chain can be studied exclusively in the smaller state space C, and is called the restriction of $\mathcal{M}$ to C. It is denoted by $\mathcal{M}_{S|C}$.*

An absorbing set is alternatively called *invariant* or *stochastically closed*. It is possible that some communicating class $C(s)$ is not absorbing. In such a case there exists $s' \notin C(s)$ such that $s \to s'$. An absorbing set is said to be *minimal* if it does not contain a proper subset that is absorbing. A Markov chain $\mathcal{M}$ is *indecomposable* if S does not contain two disjoint absorbing sets.

Definition 3.4.5 (Recurrence and Transience) *The state $s \in \mathcal{S}$ is called* recurrent *if $\mathbb{E}[f_s|s_0 = s] = \infty$ and* transient *if $\mathbb{E}[f_s|s_0 = s] < \infty$, with f_s given by Equation (3.16).*

Recurrence and transience are class properties. In fact, recurrent classes coincide with minimally absorbing classes. Furthermore, let $m_s = \mathbb{E}[\tau_s]$. Then state $s \in \mathcal{S}$ is called *positive recurrent* if $m_s < \infty$, and *null recurrent* if $m_s = \infty$. In a recurrent class either all states are positive recurrent or all null recurrent. In addition, for a finite discrete-time Markov chain, all recurrent classes are positive recurrent [56].

Definition 3.4.6 (Ergodic Markov Chain) *A Markov chain $\mathcal{M}$ is said to be* ergodic *if the whole state space $\mathcal{S}$ is a single unique recurrent class. Equivalently, it is ergodic if it is irreducible and positive recurrent.*

Definition 3.4.7 (Invariant and Ergodic Probability Measures) *Let $\nu \in M_{\mathcal{S}}$ be a probability measure (p.m.) on $\mathcal{S}$. Then, ν is an* invariant p.m. *if it remains unchanged when operated upon by the transition operator T. In vector/matrix representation, this can be written as*

$$\vec{\nu} T = \vec{\nu} \tag{3.21}$$

An invariant p.m. ν is ergodic *if $\nu(A) = 0$ or $\nu(A) = 1$ for every invariant set A subseteq$S$. Here*

$$\nu(A) = \sum_{s \in A} \nu(s).$$

Proposition 3.4.8 *[56] Let T be the transition probability function of Markov chain $\mathcal{M}$. If T has a unique invariant p.m. ν, then ν is ergodic.*

Definition 3.4.9 (Occupation Measures) *Define the* t-step expected occupation measure *with initial state* s_0 *as*

$$T^{(t)}(A|s_0) := \sum_{s\in A} \frac{1}{t} \sum_{k=0}^{t-1} T^k(s|s_0),\ A \subseteq S,\ t = 1, 2, \dots \tag{3.22}$$

Note that in the r.h.s. of Equation (3.22), T^k *is the composition of* T *with itself* $k-1$ *times, i.e,* $T^k = \underbrace{T \circ \cdots \circ T}_{k-1 \text{ times}} \circ T$. *It has the effect of transforming a belief or distribution time step* t *to the distribution at time step* $t + k$. *In matrix notation, this computation can be realized by taking the* k*th power of the matrix* T*, to get* T^k.

Additionally, an emprical or pathwise occupation measure *can be defined as follows*

$$\pi^{(t)}(A) = \frac{1}{t} \sum_{k=1}^{t} \mathbb{1}(s_k \in A),\ A \subseteq S,\ t = 1, 2, \dots. \tag{3.23}$$

Proposition 3.4.10 *[56] The expected value of the path wise occupation measure is the* $t-$*step expected occupation measure.*

$$\mathbb{E}\left[\pi^{(t)}(A)|s_0\right] = T^{(t)}(A|s_0),\ \forall t \geqslant 1. \tag{3.24}$$

Proposition 3.4.11 *[56] (a) For every* $s, s' \in S$ *the following limit exists:*

$$\lim_{t\to\infty} T^{(t)}(s'|s) = \lim_{t\to\infty} \frac{1}{t} \sum_{k=0}^{t-1} T^k(s'|s) = \begin{cases} \rho_{s'|s} & \text{if } s' \text{ is recurrent} \\ 0 & \text{if } s' \text{ is transient} \end{cases}.$$

(b) For every positive recurrent state $s \in S$ *with period* d_s

$$\lim_{t\to\infty} T^t(s|s) = \frac{d_s}{m_s}.$$

where $m_s := \mathbb{E}_s(\tau_s)$ *is the expected time of the first return to state* s *when starting in* s.

(c) Let $C = \{s_{c_1}, s_{c_2}, \dots, s_{c_{|C|}}\} \subseteq \mathcal{M}$ *be a recurrent class and* $s_c, s'_c \in C$. *Then* $\rho_{s'_c|s_c} = \nu(s_c)$ *is independent of* s'_c. *In addition the collection* $\nu(s_{c_1}), v(s_{c_2}), \dots, v(s_{c_{|C|}})$ *gives the unique invariant p.m. of the restriction of* $\mathcal{M}$ *to the class* C.

Definition 3.4.12 (Limiting Matrix) *From Proposition 3.4.11, the matrix representation of* $T^{(t)}$ *is given by the Cesaro sum [71],*

$$T^{(t)} = \frac{1}{t} \sum_{k=0}^{t-1} T^k,\ t = 1, 2, \dots \tag{3.25}$$

and the limiting matrix

$$\Pi := \lim_{t\to\infty} T^{(t)} \tag{3.26}$$

exists for all finite Markov chains.

Proposition 3.4.13 *Given the limiting matrix* Π*, the limit of the Cesaro sum of transition matrix* T*, the quantity* $I - T + \Pi$ *is non-singular and its inverse*

$$Z := (I - T + \Pi)^{-1} \tag{3.27}$$

is called the fundamental matrix *[15, 56, 123].*

3.5 Overview of Solution Methodology

Let us now return to the global Markov chain $\mathcal{M}_{\mathcal{S}\times G}^{\mathcal{PM}^\varphi,\mathcal{G}}$ induced by the FSC $\mathcal{G}$ controlling the *product-POMDP*, $\mathcal{PM}^\varphi$. This chain evolves over the global state space $\mathcal{S} \times G$, where the Product-POMDP state space is given by $\mathcal{S} = (\mathcal{S}^{model} \times Q)$. Since the state space is finite, every state is either positive recurrent or transient. First, some notation is introduced.

Consider a product state $s \in \mathcal{S}$. If there exists $g \in G$ such that the global state $[s, g]$ is recurrent in the global Markov chain, then abusing notation, s is said to be *recurrent under* $\mathcal{G}$.

Next, if $[s, g]$ is a state of the global Markov chain, then $\perp_{\mathcal{S}} ([s, g]) = s$ projects the state onto the product-POMDP state space. In addition for a set $A = \{[s_i, g_i], \dots\} \in (\mathcal{S} \times G)$ the projection $\perp_{\mathcal{S}}$ is

$$\perp_{\mathcal{S}} (A) = \{s_i, \dots\} \quad \text{(taken uniquely)}.$$

Finally, let $\mathcal{R}^{\mathcal{G}}$ denote the set of all recurrent states of $\mathcal{M}_{\mathcal{S}\times G}^{\mathcal{PM}^\varphi,\mathcal{G}}$.

3.5.1 Solutions for Qualitative Problems

The qualitative problem of positive satisfaction can be solved as follows.

Proposition 3.5.1 *The probability of satisfaction* φ *is non-zero, i.e.,* $\Pr(\mathcal{PM} \models \varphi) > 0$*, if there exists an FSC,* $\mathcal{G}$*, such that for some Rabin acceptance pair* $(Repeat_i^{\mathcal{PM}^\varphi}, Avoid_i^{\mathcal{PM}^\varphi})$*,*

$$\begin{aligned} Repeat_i^{\mathcal{PM}^\varphi} \cap \perp_{\mathcal{S}} (\mathcal{R}^{\mathcal{G}}) &\neq \varnothing, \\ \exists [s, g] \text{ with } \iota_{init}^{\mathcal{PM}^\varphi,\mathcal{G}}([s, g]) > 0 \text{ s.t. } [s, g] &\rightarrow (Repeat_i^{\mathcal{PM}^\varphi} \times G) \cap \mathcal{R}^{\mathcal{G}}, \\ \text{and } [s, g] &\nrightarrow (Avoid_i^{\mathcal{PM}^\varphi} \times G) \cap \mathcal{R}^{\mathcal{G}}. \end{aligned} \tag{3.28}$$

The first equation above says that under the FSC $\mathcal{G}$ there is some recurrent state in $Repeat_i^{\mathcal{PM}^\varphi}$. The second equation requires that such a recurrent state is reachable for the given initial distribution. The last equation states that even if some states in $Avoid_i^{\mathcal{PM}^\varphi}$ could be visited when starting from the initial distribution, these cannot be recurrent.

The qualitative problem of almost sure satisfaction can be solved as follows.

Proposition 3.5.2 *The formula φ is satisfied almost surely, i.e.,* $\Pr(\mathcal{PM} \models \varphi) = 1$*, if there exists an FSC, $\mathcal{G}$, such that for some Rabin acceptance pair* $(Repeat_i^{\mathcal{PM}^\varphi}, Avoid_i^{\mathcal{PM}^\varphi})$

$$\begin{aligned} Repeat_i^{\mathcal{PM}^\varphi} \cap \perp_{\mathcal{S}} (\mathcal{R}^{\mathcal{G}}) &\neq \varnothing, \\ \forall [s,g] \text{ s.t. } \iota_{init}^{\mathcal{PM}^\varphi,\mathcal{G}}([s,g]) > 0,\ \Pr\big[[s,g] &\rightarrow (Repeat_i^{\mathcal{PM}^\varphi} \times G) \cap \mathcal{R}^{\mathcal{G}}, \text{ and} \\ [s,g] &\nrightarrow (Avoid_i^{\mathcal{PM}^\varphi} \times G) \cap \mathcal{R}^{\mathcal{G}}\big] = 1. \end{aligned} \tag{3.29}$$

The first equation is the same as for positive satisfaction. The last equation requires that for every possible initial state, the execution is *guaranteed* to reach a recurrent state that has some state from $Repeat_i^{\mathcal{PM}^\varphi}$ while simultaneously avoiding those states from $Avoid_i^{\mathcal{PM}^\varphi}$ that are recurrent under $\mathcal{G}$.

3.5.2 Solutions for Quantitative Problems

It is easier to reason about the quantitative problems by looking at the decomposition of the global Markov chain into its recurrent classes. Here, let the set of all recurrent states, $\mathcal{R}$, be partitioned into disjoint recurrent classes $RecSets^{\mathcal{G}} = \{R_1, R_2, \ldots R_N\}$ such that

$$\begin{aligned} R_1 \cup R_2 \cup \cdots \cup R_N &= \mathcal{R} \\ R_i \cap R_j &= \varnothing,\ i \neq j. \end{aligned} \tag{3.30}$$

In addition, the partitioning is required to be *maximal*. Formally, this means that for each $R_k, R_l \in RecSets^{\mathcal{G}}$,

$$\begin{aligned} s_i &\leftrightarrow s_j,\ \forall s_i, s_j \in R_k \\ s_i &\nleftrightarrow s_j,\ s_i \in R_k,\ s_j \in R_l, k \neq l. \end{aligned} \tag{3.31}$$

The first equation states that within each recurrent class or set, R_k, all states are reachable from one another. The second equation states that no two distinct recurrent classes can be combined to make a larger recurrent class, thus making the partitions maximal. We remind the reader that recurrence implies absorption, i.e., if the Markov chain path leads to a state in a recurrent set, then the path is forever confined to that recurrent set. This implies the following when considering probability of long term behavior of paths:

$$\Pr[\pi \rightarrow (R_k \cup R_l)] = \Pr[\pi \rightarrow R_k] + \Pr[\pi \rightarrow R_l],\ k \neq l \tag{3.32}$$

In addition, over infinite executions the path must end up in some recurrent set, i.e.,

$$\sum_{R_k \in RecSets^{\mathcal{G}}} \Pr[\pi \rightarrow R_k] = 1. \tag{3.33}$$

Next, the concept of a feasible recurrent set is introduced.

Definition 3.5.3 (φ-feasible Recurrent Set) *Under a given FSC, $\mathcal{G}$, a (maximal) recurrent set or class R_k is a* φ-feasible recurrent set *if $\exists(Repeat_i^{\mathcal{PM}^\varphi}, Avoid_i^{\mathcal{PM}^\varphi})$ such that,*

$$\begin{aligned} \perp_{\mathcal{S}}(R_k) \cap Repeat_i^{\mathcal{PM}^\varphi} &\neq \varnothing, \text{ and} \\ \perp_{\mathcal{S}}(R_k) \cap Avoid_i^{\mathcal{PM}^\varphi} &= \varnothing. \end{aligned} \tag{3.34}$$

Let φ-$RecSets^{\mathcal{G}} = \bigcup R_k$, s.t. R_k is φ-feasible.

Definition (3.5.3) is closely related to Equation (3.28). R_k is φ-feasible by the above definition if the last two criteria in Equation (3.28) hold.

Proposition 3.5.4 *The problem of computing the satisfaction probability φ under a given FSC, $\mathcal{G}$ can be solved by computing*

$$\Pr[\mathcal{PM} \models \varphi] = \sum_{R \in \varphi\text{-}RecSets^{\mathcal{G}}} \Pr[\pi \to R]. \tag{3.35}$$

The r.h.s. of the Equation (3.35) computes the probability of any path (given the initial distribution) reaching any of the φ-feasible recurrent sets introduced in Definition 3.5.3 wherein the relationship to Equation (3.28) was also described.

The problem of maximizing the probability of satisfaction can be solved as follows.

Proposition 3.5.5 *The satisfaction probability of an LTL formula can be maximized by optimizing the following objective*

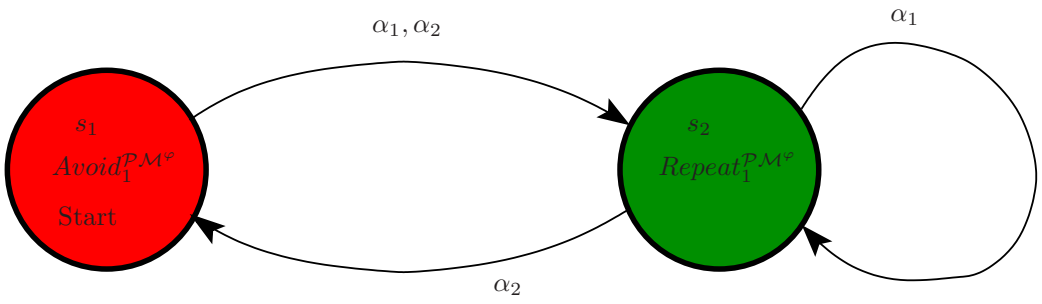
$$\max_{\mathcal{G}} \sum_{R \in \varphi\text{-}RecSets^{\mathcal{G}}} \Pr[\pi \to R] \tag{3.36}$$

To further understand the solution to Equation (3.36), note that there are two main components in the choice of the FSC, $\mathcal{G}$:

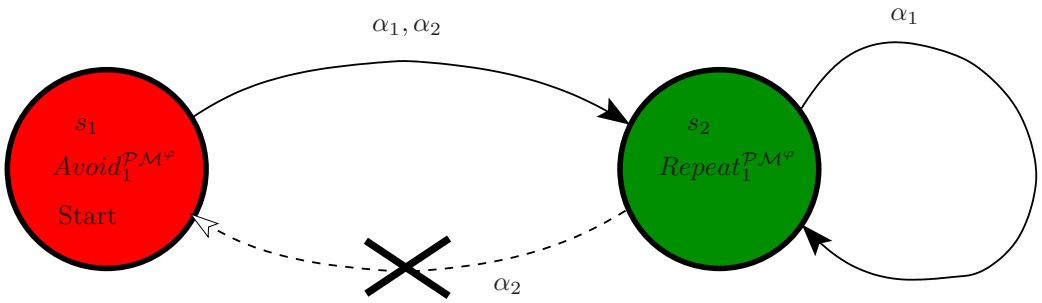
1. **Structure:** The structure of the FSC has two components:

   (a) The number of internal nodes, G, available. This determines the size of the global Markov chain state space.

   (b) The set of parameters in ω and κ with non-zero values. The actual numerical values of these parameters do not affect the structure, but the non-zero set determines the connectivity of the underlying graph of the global state space. The nodes of the graph are the states of the global Markov chain. A directed edge from a node $[s, g]$ to $[s', g']$ determines whether a one-step transition can be made from $[s, g]$ to $[s', g']$. The underlying graph completely and unambiguously determines the recurrent and transient classes of the global Markov chain. This is explained using a small example in Figure 3.2.



(a)



(b)

Figure 3.2: Effect of FSC structure on φ-feasibility. Consider a case in which the Product-POMDP state space comprises of two states $s_1 \in Avoid_1^{\mathcal{PM}^\varphi}$ and $s_2 \in Repeat_1^{\mathcal{PM}^\varphi}$. There are two actions $\alpha_1, \alpha_2 \in Act$, both causing deterministic state transitions as shown. For both figures consider a trivial FSC with only one I-state, i.e., $G = \{g_1\}$. The resulting global state space is given by $\mathcal{S} \times G = \{[s_1, g_1], [s_2, g_1]\}$. (a) For all actions $\alpha \in Act$, let $\omega(g_1, \alpha, g_1, o) > 0), \quad \forall o \in \mathcal{O}$. This means that the underlying graph results in global states $[s_1, g_1]$ and $[s_2, g_1]$ being recurrent and belonging to the recurrent class, which is given by $R_1 = \mathcal{S} \times G$. Since $Avoid_1^{\mathcal{PM}^\varphi, \mathcal{G}} \cap R_1 \neq \varnothing$, R_1 is not φ-feasible. Note that R_1 and its φ-feasibility is unchanged for any value of $omega > 0$ (b) Consider the same FSC with a different structure in which $\omega g_1, \alpha_2 | g_1, o = 0, \quad \forall o \in \mathcal{O}$. This results in a different recurrent class in the global state space and is given by $R_1 = \{[s_2, g_1]\}$. Since R_1 and $Avoid_1^{\mathcal{PM}^\varphi, \mathcal{G}}$ are disjoint while $R_1 \cap Repeat_1^{\mathcal{PM}^\varphi} \neq \varnothing$, R_1 is φ-feasible.

Thus the structure affects both the partitioning $RecSets^{\mathcal{G}}$ and also the φ-feasibility of these sets.

2. **Quality:** By quality I will mean the numerical values of non-zero parameters of ω and κ. These determine the probability with which paths reach each some $R \in RecSets$.

Therefore, to maximize the probability of φ-satisfaction, one must simultaneously search over all structures and qualities of FSCs for the best probability. The optimization variable, $\mathcal{G}$, encompasses both.

3.6 Concluding Remarks

This chapter concretely formalized the various problems related to LTL satisfaction over POMDPs. It also motivated the use of Finite State Controllers and the resulting finite global Markov chain. LTL satisfaction was then related to the global Markov chain and it was demonstrated how the structure and quality of the FSC affect the long term behavior of the system with respect to the LTL specification. In the next chapter, a gradient based algorithm will be introduced to find a locally optimal FSC for maximizing the probability of an LTL formula being satisfied by the controlled system.

Chapter 4

Policy Gradient Method

This chapter focuses on the basic problem of maximizing the LTL satisfaction probability when the *structure* of the FSC is predetermined. In order to carry this out, an FSC of a fixed structure will be parametrized in such a way that the FSC retains its structure over the entire domain of parametrization. Using this parametrization, the FSC control design problem will then be posed as the optimization of a parametrized objective function whose gradient can be computed in polynomial time. The approach presented in this chapter assumes that at least one recurrent set in the resulting global Markov chain is φ-feasible for a starting set of parameters. Finally, a heuristic randomized algorithm is presented that allows finding structures of FSCs of a given size that yield at least one φ-feasible recurrent set.

4.1 Structure and Parametrization of an FSC

4.1.1 Structure

Formally, the structure of the FSC is determined by the collection $\mathcal{I} = \{G, \mathcal{I}_\omega, \mathcal{I}_\kappa\}$, where the three components are:

1. The Size of FSC. The set $G = \{g_1, g_2, \ldots, g_{|G|}\}$ indexes the states of the FSC. Any *admissible* FSC must have non empty set of states

$$G \neq \varnothing. \tag{4.1}$$

2. Allowed State Transitions. The binary function $\mathcal{I}_\omega : (G \times \mathcal{O}) \times (G \times Act) \to \{0, 1\}$ describes the allowed state transitions as follows. For $\forall \alpha \in Act,\ \forall o \in \mathcal{O},$ and $\forall g_k, g_l \in G$,

$$\mathcal{I}_{g_l \alpha | g_k o} := \mathcal{I}_\omega(g_l, \alpha | g_k, o) = \begin{cases} 1 & \text{if observation } o \in \mathcal{O} \text{ in FSC I-state } g_k \\ & \text{can lead to an FSC transition to the I-state } g_l \\ & \text{while issuing action } \alpha \in Act \\ 0 & \text{otherwise.} \end{cases} \tag{4.2}$$

However, a given structure of the FSC is only *admissible* if

$$\forall (g_k, o) \in (G \times \mathcal{O}), \quad \exists \ (g_l, \alpha) \in (G \times Act) \quad \text{s.t.} \quad \mathcal{I}_\omega(g_l, \alpha | g_k, o) = 1. \tag{4.3}$$

The above condition can be understood as follows. Let the FSC be in any given I-state $g_k \in G$. Then for every observation $o \in \mathcal{O}$, there must be an I-state $g_l \in G$ and an action $\alpha \in Act$, such that the FSC can transition to g_l and issue action α to the product-POMDP.

3. Allowed Initial I-States. The binary function $\mathcal{I}_\kappa : G \to \{0, 1\}$ which is assumed to be fixed and known, describes the allowed FSC initial states. $\forall g \in G$,

$$\mathcal{I}_g := \mathcal{I}_\kappa(g) = \begin{cases} 1 & \text{if the FSC can start in state } g \\ 0 & \text{otherwise.} \end{cases} \tag{4.4}$$

As before, the FSC structure is *admissible* only if

$$\exists \ g \in G, \quad \text{s.t.} \quad \mathcal{I}_\kappa(g) = 1. \tag{4.5}$$

4.1.2 Structure Preserving Parametrization of an FSC

Note that any parametrization of the FSC is admissible as long as it obeys the laws of probability. This condition can be described as

$$\begin{aligned} \sum_{(g_l, \alpha) \in (G \times Act)} \omega(g_l, \alpha | g_k, o) &= 1 \ \ \forall (g_k, o) \in (G \times \mathcal{O}) \\ \sum_{g \in G} \kappa(g | \iota_{init}^{\mathcal{PM}^\varphi}) &= 1 \ . \end{aligned} \tag{4.6}$$

Let ω and κ be respectively parametrized by $\Phi = \{\phi_1, \ldots \phi_{n_\phi}\}$ and $\Theta = \{\theta_1, \ldots, \theta_{n_\theta}\}$ in the following structure.

$$\begin{aligned} \omega(g_l, \alpha | g_k, o) &:= \omega(g_l, \alpha | g_k, o, \Phi), \\ \kappa(g | \iota_{init}^{\mathcal{PM}^\varphi}) &:= \kappa(g | \iota_{init}^{\mathcal{PM}^\varphi}, \Theta). \end{aligned} \tag{4.7}$$

In addition to the probability laws in Equation (4.6), it is required that the parametrization of the FSC by ω and κ results in quantities

$$\nabla_\Phi \omega, \nabla_\Theta \kappa, \frac{\left| \frac{\omega(g_l, \alpha | g_k, o, \Phi)}{\partial \phi_i} \right|}{\omega(g_l, \alpha | g_k, o, \Phi)}, \frac{\left| \frac{\kappa(g|\Theta)}{\partial \theta_j} \right|}{\kappa(g|\Theta)},$$

which are all uniformly bounded from above so that (conjugate) gradient optimization methods can be applied.

Softmax Parametrization: Of several possible parametrizations of ω and κ, this chapter illustrates a gradient optimization algorithm using the softmax parametrization. The softmax function arises frequently in neural networks, where it is used as a neural activation function [25] and ma-

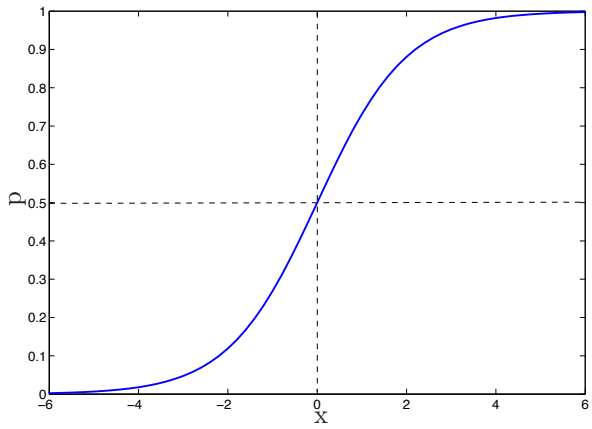


Figure 4.1: The logistic function with the parameter x can be used to continuously parametrize the Bernoulli trial by allowing the probability of success p of the trial.

chine learning models such as the generalized linear model [98]. It can be viewed as the multi-variate generalization of the logistic function

$$logistic(x) = \frac{1}{1 + \exp(-x)}, \quad x \in \mathbb{R}. \tag{4.8}$$

Equation (4.8) can be used to continuously parametrize a Bernoulli trial. Consider for example, a coin toss experiment with a biased coin. Let the probability of turning up heads in a single trial be given by p. Next, let p be parametrized using the logistic function in Equation (4.8), i.e.,

$$p(x) = logistic(x) \tag{4.9}$$

thereby making x a parameter in the original coin toss experiment. As x reaches $-\infty$ or $+\infty$, p approaches 0 or 1 in the limit, respectively. This results in a deterministic outcome for the Bernoulli trials. However, for finite values of x, $0 < p < 1$. For $x = 0$, the coin is fair, i.e., $p = 0.5$. The logistic parametrization of p can be seen in Figure 4.1.

This can be extended to experiments whose trials have a single outcome from a multitude of choices. Let the set of outcomes of a trial be given by $\{1, 2, \ldots, N\}$ and the probability that a single trial has outcome $j \in \{1, \ldots, N\}$ be given by p_j. Clearly,

$$\begin{aligned} p_j &\geqslant 0, \text{ and} \\ \textstyle\sum_{j=1}^{N} p_j &= 1 \end{aligned} \tag{4.10}$$

to satisfy the laws of probability. Next, introduce parameters $x_1, x_2, \ldots, x_N$, and let the probability

of a given outcome be given by

$$p_j(x_1, \ldots, x_N) = \frac{\exp(x_j)}{\sum_{k=1}^{N} \exp(x_k)} \tag{4.11}$$

It can be seen easily that the set of probabilities $p_j(x_1, \ldots, x_N)$ satisfy the probability laws in Equation (4.10) for any real values of $x_1, \ldots, x_n$. When $x_1 = x_1 = \cdots = x_N$, then all outcomes have equal probability of $\frac{1}{N}$. As $x_1, \ldots, x_n$ each vary in the interval $(-\infty, \infty)$, $\{p_1, \ldots, p_N\}$ each vary *continuously* in the open interval $(0, 1)$ w.r.t each x_i. Additionally, if for some k, $(x_k - x_l) \to \infty, \ \forall\ l \neq k$, then $p_k \to 1$. Similarly if $(x_k - x_l) \to -\infty, \ \forall l \neq k$, then $p_k \to 0$. In fact, the approach to 0 or 1 is exponentially quick, and therefore in numerical implementations, if one parameter grows quickly, the trials become deterministic.

The softmax parametrization of Equation (4.11) has been studied in detail for the classic POMDP reward maximization problem [1]. This choice is well suited for numerical algorithms as the softmax function is convex in its parameters and its derivative is easily computed. In addition, the fast approach to deterministic outcomes as some parameters grow faster than others, is a desirable property for engineering systems where predictable behavior is favored or even crucial. However, the softmax parametrization also has some drawbacks related to the initialization of the FSC, which are addressed in Section 4.6.

Let the I-state and action distribution be parametrized by

$$\Phi = \begin{bmatrix} \vec{\phi}_{g_1 o_1} \\ \vec{\phi}_{g_1 o_2} \\ \vdots \\ \vec{\phi}_{g_1 o_{|\mathcal{O}|}} \\ \vec{\phi}_{g_2 o_1} \\ \vdots \\ \vec{\phi}_{g_{|G|} o_{|\mathcal{O}|}} \end{bmatrix}_{|G||\mathcal{O}| \times |G||Act|} \tag{4.12}$$

where

$$\vec{\phi}_{g_i o_k} = \begin{bmatrix} \phi_{g_1 \alpha_1 | g_i o_k} & \phi_{g_1 \alpha_2 | g_i o_k} & \cdots & \phi_{g_1 \alpha_{|Act|} | g_i o_k} \\ \phi_{g_2 \alpha_1 | g_i o_k} & \cdots & \phi_{g_{|G|} \alpha_{|Act|} | g_i o_k} \end{bmatrix}_{1 \times |G||Act|} \tag{4.13}$$

is a row vector and $\phi_{g_j \alpha_n | g_i o_k}$ is a real number that controls the relative probability of making the internal state transition $g_i \to g_j$ *and* taking action α_n, having observed o_k. This probability can be

computed using the expression for soft-max function

$$\omega(g_j, \alpha_n|\Phi, g_i, o_k) = \frac{\mathcal{I}_{g_j\alpha_n|g_io_k} \exp(\phi_{g_j\alpha_n|g_io_k})}{\sum\limits_{g\in G}\sum\limits_{\alpha\in Act} \mathcal{I}_{g\alpha|g_io_k} \exp(\phi_{g\alpha|g_io_k})}. \tag{4.14}$$

The derivative of ω with respect to the parameters is

$$\frac{\partial\omega(g_j, \alpha_n|\Phi, g_i, o_k)}{\partial\phi_{g\alpha|g'o}} = \begin{cases} \omega(g_j, \alpha_n|\Phi, g_i, o_k)\Big(\mathbb{1}\big((g,\alpha) = (g_j, \alpha_n)\big) - \omega(g, \alpha|\Phi, g_i, o_k)\Big) \\ \qquad\qquad \text{if } g_i = g',\ o_k = o \\ 0 \qquad\quad \text{otherwise.} \end{cases} \tag{4.15}$$

Likewise, letting $\Theta = \begin{bmatrix}\theta_{g_1} & \theta_{g_2} \dots \theta_{g_{|G|}}\end{bmatrix}$ denote the second set of parameters, the initial I-state distribution can be described as

$$\kappa(g_k|\iota_{init}^{\mathcal{PM}^\varphi}, \Theta) = \frac{\mathcal{I}_{g_k} \exp(\theta_{g_k})}{\sum\limits_{g\in G} \mathcal{I}_g \exp(\theta_g)}, \tag{4.16}$$

and its gradient takes the form

$$\frac{\partial\kappa(g_k|\iota_{init}^{\mathcal{PM}^\varphi}, \Theta)}{\partial\theta_g} = \kappa(g_k|\iota_{init}^{\mathcal{PM}^\varphi}, \Theta)\Big(\mathbb{1}(g = g_k) - \kappa(g|\iota_{init}^{\mathcal{PM}^\varphi}, \Theta)\Big). \tag{4.17}$$

Proposition 4.1.1 *Let a structure $\mathcal{I}$, of an FSC, be admissible as given by Equations (4.1), (4.3) and (4.5). Then, any softmax parametrization (Equations (4.14) and (4.16)) given by real and finite values of parameters Φ and Θ preserves the structure. Let the set of all softmax parametrized FSCs that have structure $\mathcal{I}$, be denoted $\mathfrak{G}(\mathcal{I})$.*

Proof : Consider a particular I-state g_i of the FSC. Note that an admissible structure of the FSC requires that for each observation $o_k \in \mathcal{O}$, there at least one pair $(g_{next}, \alpha_{next}) \in G \times Act$ such that the FSC transitions to g_{next} and issues action α_{next}. This implies that expression for ω in Equation (4.14) can be written as

$$\omega(g_j, \alpha_n|\Phi, g_i, o_k) = \frac{\mathcal{I}_{g_j\alpha_n|g_ko} \exp(\phi_{g_j,\alpha_n|g_io_k})}{\left(\sum\limits_{(g,\alpha)\neq(g_{next},\alpha_{next})} \mathcal{I}_{g\alpha|g_io_k} \exp(\phi_{g\alpha|g_io_k})\right) + \exp(\phi_{g_{next}\alpha_{next}|g_io_k})}. \tag{4.18}$$

This implies that for finite $\phi_{g_{next}\alpha_{next}|g_io_k}$ the denominator is non-zero. Next, note that in the numerator the exponential is non-zero for all finite values of $\phi_{g_j\alpha_n|g_io_k}$. Thus, $\omega(g_j, \alpha_n|\Phi, g_i, o_k)$ is zero if and only if $\mathcal{I}_{g_j\alpha_n|g_io_k}$ is zero. This is true for every $g_i \in G$. This proves that any real finite parametrization of ω preserves $\mathcal{I}_\omega$. The proof for $\mathcal{I}_\kappa$ follows entirely similarly.

4.2 Maximizing Probability of LTL Satisfaction for an FSC of Fixed Structure

4.2.1 Vector Notation for Finite Sets

Let $\mathcal{S}$ be a finite total-ordered set with the binary relation $\preceq$ giving the order. Then define $\vec{\mathcal{S}}_{\preceq}$ as the column vector

$$\vec{\mathcal{S}} = \begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ s_{|\mathcal{S}|} \end{bmatrix}$$

such that $s_i \preceq s_j,\ \forall i \leqslant j \leqslant |\mathcal{S}|$. If $A \subseteq \mathcal{S}$ then $\vec{A}$ is a column vector in which the elements respect the ordering as well. Let $A \in \mathcal{S}$. Then, let $\vec{1}_{\mathcal{S}|A}$ denote the column vector of size $|\mathcal{S}| \times 1$ as follows

$$\vec{1}_{\mathcal{S}|A} = \begin{bmatrix} \mathbb{1}(s_1 \in A) \\ \mathbb{1}(s_2 \in A) \\ \vdots \\ \mathbb{1}(s_{|\mathcal{S}|} \in A) \end{bmatrix}_{|\mathcal{S}|\times 1}$$

where, as before, $s_i \preceq s_j,\ i \leqslant j \leqslant |\mathcal{S}|$. The elements of this vector assume a unit value in those places that correspond to states belonging to A, and zero if not.

4.2.2 Ordering Global States by Recurrent classes

Definition 4.2.1 (Total Order of a Set) *Let $\preceq \subseteq X \times X$ denote a binary relation. If the ordered tuple $(x, y) \in \preceq$, then the short hand notation is to write $x \preceq y$. $\preceq$ is called a* total order *over the set X if it satisfies the following properties properties. $\forall\ \ x, y, z \in X$,*

1. **Antisymmetry:** *If $x \preceq y$ and $y \preceq x$, then $x = y$.*
2. **Transitivity:** *If $x \preceq y$ and $y \preceq z$, then $x \preceq z$.*
3. **Totality:** *Either $x \preceq y$ or $y \preceq x$.*

In this and subsequent chapters, it will be frequently assumed that the global state space, $\mathcal{S} \times G = (\mathcal{S}^{model} \times Q) \times G$, is ordered by recurrent classes. From Definition 3.4.5, it is clear that under a fixed structure of the FSC, the (maximal) recurrent classes are unique and known. Let $RecSets^{\mathcal{G}(\mathcal{I})} = \{R_1, R_2, \ldots R_N\}$ be the recurrent sets. Let these sets have arbitrary, but fixed total

ordering $\preceq'$. That is, for any two R_i,R_j, either $R_i \preceq' R_j$ or $R_j \preceq' R_i$, and $\preceq'$ also satisfies the transitivity and antisymmetry conditions of Definition 4.2.1. Let each $R_k \in RecSets^{\mathcal{G}(\mathcal{I})}$ have total order $\preceq_k \subseteq R_k \times R_k$ over its member global states. Further, since some states in the global state space $\mathcal{S} \times G$ are transient, collect all transient states in the set $\mathcal{T}$ and endow it with an arbitrary, but fixed total ordering, $\preceq_{\mathcal{T}} \subseteq \mathcal{T} \times \mathcal{T}$. Finally define the following relation $\preceq$ on the global state space $\mathcal{S} \times G$ as follows. For $\forall [s,g], [s',g'] \in \mathcal{S} \times G$,

$$[s,g] \preceq \left[s',g'\right] \text{ if and only if } \begin{cases} [s,g], [s',g'] \in R_k \text{ and } [s,g] \preceq_k [s',g'], & \text{or,} \\ [s,g] \in R_k, [s',g'] \in R_l,\ k \neq l, \text{ and } R_k \preceq' R_l, & \text{or,} \\ [s,g] \in R_k \subseteq RecSets^{\mathcal{G}} \text{ and } [s',g'] \in T, & \text{or,} \\ [s,g], [s',g'] \in \mathcal{T} \text{ and } [s,g] \preceq_T [s',g']. \end{cases} \tag{4.19}$$

Proposition 4.2.2 *The relation $\preceq$ in Equation (4.19) defines a total order over the set of states $\mathcal{S} \times G$.*

Proof Sketch : See Appendix B.2.

When the transition probabilities of the controlled system are written in matrix form, then the above ordering results in a *canonical* block diagonal form

$$T^{\mathcal{PM}^{\varphi},\mathcal{G}} = \left[\begin{array}{cccc|c} T_{R_1} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & T_{R_2} & \dots & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \dots & T_{R_N} & \mathbf{0} \\ \hline T_{\mathcal{T}\to R_1} & T_{\mathcal{T}\to R_2} & \dots & T_{\mathcal{T}\to R_N} & T_{\mathcal{T}\to\mathcal{T}} \end{array}\right]_{|\mathcal{S}||G|\times|\mathcal{S}||G|} \tag{4.20}$$

where the matrices T_{R_k}, corresponding to the recurrent sets, are stochastic (each row sums to 1), and can be directly used to represent the restriction of the global Markov chain to R_k.

Similar to the canonical form in Equation (4.20), the same global state ordering in Equation (4.19) results in a block form for the initial distribution of the controlled system as follows

$$\vec{l}_{init}^{\mathcal{PM}^{\varphi},\mathcal{G}} = \left[\begin{array}{c} \vec{l}_{init}^{\mathcal{PM}^{\varphi},\mathcal{G}}(R_1) \\ \vec{l}_{init}^{\mathcal{PM}^{\varphi},\mathcal{G}}(R_2) \\ \vdots \\ \vec{l}_{init}^{\mathcal{PM}^{\varphi},\mathcal{G}}(R_N) \\ \hline \vec{l}_{init}^{\mathcal{PM}^{\varphi},\mathcal{G}}(\mathcal{T}) \end{array}\right]_{|\mathcal{S}||G|\times 1} . \tag{4.21}$$

4.2.3 Probability of Absorption in φ-feasible Recurrent Sets

Recall that for a given FSC $\mathcal{G} \in \mathfrak{G}(\mathcal{I})$ of fixed structure, the recurrent set $\varphi\text{-}RecSets^{\mathcal{G}}$, which is a subset of the global state space $\mathcal{S} \times G$, denotes the union of all recurrent sets that are φ-feasible, and is uniquely determined by the structure, $\mathcal{I}$. This section aims to compute the probability of absorption into this set, given the initial distribution of the product-POMDP, $\mathcal{PM}^{\varphi}$. It will be shown to be a function of parameters Φ and Θ and analytical expressions of the probability of absorption in terms of these parameters will be derived.

The probability of absorption into a recurrent set R_k for finite Markov chains is well known [71]. Let $\vec{1}_{M\times 1}$ denote a column vector of size M with all entries equal to 1. Then using the block decomposition in Equations (4.20) and (4.21), the following holds.

$$\begin{aligned}\Lambda(R_k) &= \Pr\left(\pi \to R_k | \iota_{init}^{\mathcal{PM}^{\varphi}}\right) \\ &= \underbrace{\left(\vec{\iota}_{init}^{\mathcal{PM}^{\varphi},\mathcal{G}}(R_k)\right)^T \vec{1}_{|R_k|\times 1}}_{\text{Term 1}} \\ &\quad + \underbrace{\left(\vec{\iota}_{init}^{\mathcal{PM}^{\varphi},\mathcal{G}}(\mathcal{T})\right)^T \left(I + T_{\mathcal{T}\to\mathcal{T}} + T_{\mathcal{T}\to\mathcal{T}}^2 + \dots\right) T_{\mathcal{T}\to R_k} \vec{1}_{|R_k|\times 1}}_{\text{Term 2}} .\end{aligned} \tag{4.22}$$

In the above equation, Term 1 is simply the probability that under the initial distribution, $\vec{\iota}_{init}^{\mathcal{PM}^{\varphi},\mathcal{G}}$, the initial global state $(s_0, g_0) \in \mathcal{S} \times G$ is in the recurrent set R_k. If this is so, any resulting path of the controlled system is guaranteed to remain in R_k forever. Next, Term 2 can be rewritten as

$$\text{Term 2} = \sum_{t=0}^{\infty} \left(\vec{\iota}_{init}^{\mathcal{PM}^{\varphi},\mathcal{G}}(\mathcal{T})\right)^T \left(T_{\mathcal{T}\to\mathcal{T}}\right)^t T_{\mathcal{T}\to R_k} \vec{1}_{|R_k|\times 1}. \tag{4.23}$$

For each t, the corresponding summand denotes the probability that the execution started and stayed in some transient state in $\mathcal{T}$, until exactly t-th time step before getting absorbed in R_k at time step $t+1$. The following lemma shows that the infinite sum in Equation (4.23) converges.

Lemma 4.2.3 *[71] : The limit*

$$\lim_{t\to\infty} \sum_{k=0}^{t} T_{\mathcal{T}\to\mathcal{T}}^k \tag{4.24}$$

exists and is equal to $(I - T_{\mathcal{T}\to\mathcal{T}})^{-1}$.

Equation (4.22) with Lemma 4.2.3 allows the probability of absorption in *any* φ-feasible set to be computed as

$$\begin{aligned}\Pr\left(\pi \to \varphi\text{-}RecSets^{\mathcal{G}} | \iota_{init}^{\mathcal{PM}^{\varphi}}\right) &= \sum_{R_k \subseteq \varphi\text{-}RecSets^{\mathcal{G}}} \Lambda(R_k) \\ &= \sum_{R_k \subseteq \varphi\text{-}RecSets^{\mathcal{G}}} \Pr\left(\pi \to R_k | \iota_{init}^{\mathcal{PM}^{\varphi}}\right),\end{aligned} \tag{4.25}$$

which gives the LTL satisfaction probability as a function of the parameters of $\mathcal{G}$.

4.2.3.1 Complexity and Efficient Approximation

Before going into the details of how Equation (4.22) is used to optimize the satisfaction probability of φ, it is worthwhile to look at the complexity and efficiency of computing the r.h.s. of that equation. One source of computational complexity arises from the need to compute the recurrent sets of the global Markov chain. The growth of this complexity is analyzed in Section 4.5.1. Here, the computational complexity arising due to the infinite sum in Term 2 of Equation (4.22) is presented. Lemma 4.2.3 offers one method of computing the r.h.s using the infinite sum $1 + T_{\mathcal{T}\to\mathcal{T}} + T_{\mathcal{T}\to\mathcal{T}}^2 + \ldots$, by inverting $(1 - T_{\mathcal{T}\to\mathcal{T}})$, which has complexity $\mathbf{O}(|\mathcal{T}|^3)$. A less computationally intensive method to compute Term 2 is by recognizing that this sum is finally multiplied by $T_{\mathcal{T}\to R_k}\vec{1}_{|R_k|\times 1}$, which computes to a column vector of size $|R_k| \times 1$. As suggested in [1], this allows for an iterative method with lower complexity as follows. Initialize variables v_0 and x_0

$$\begin{aligned} v_0 &= T_{\mathcal{T}\to R_k}\vec{1}_{|R_k|\times 1}, \\ x_0 &= \mathbf{0} \end{aligned} \tag{4.26}$$

Then iterate the equations

$$\begin{aligned} v_{n+1} &= T_{\mathcal{T}\to\mathcal{T}} v_n, \\ x_{n+1} &= x_n + v_n \end{aligned} \tag{4.27}$$

until convergence, i.e., for some tolerance $\varepsilon_x > 0$, $\exists\ N_x$ s.t. $||x_N - x_{N-1}||_\infty \leqslant \varepsilon_x \ \ \forall \ \ N \geqslant N_x$. The convergence of Equation (4.27) is guaranteed because it is well known that $T_{\mathcal{T}\to\mathcal{T}}^n \to 0$ as $n \to \infty$ [71]. Then, use the approximation

$$\widehat{\text{Term 2}} = \left(\vec{\iota}_{init}^{\mathcal{PM}^{\varphi},\mathcal{G}}\right)^T x_{N_x}. \tag{4.28}$$

This approximation method has the complexity $\mathbf{O}(|R_k|^2 N_x)$. In fact, if the underlying POMDP's transition distribution is sparse, then sparse matrix multiplication and addition can be used in the approximation method, whose practical complexity reduces to $\mathbf{O}(c|R_k|N_x)$ with $c \ll |R_k|$. The sparsity assumption may appear in many engineering examples such as robotic systems in which only a few other states are reachable from a particular state. The constant c is dependent on the sparsity level.

4.2.4 Gradient of Probability of Absorption

Equation (4.22) provides an expression for the probability of absorption: an analytical expression for its gradient with respect to Φ and Θ can also be derived. In Terms 1 and 2 of Equation (4.22), the expression $\iota_{init}^{\mathcal{PM}^\varphi,\mathcal{G}}$ is a function only of the parameters Θ via the initial FSC node distribution κ. In Term 2, the expressions $\lim_{t\to\infty}\sum_{k=0}^{t} T_{\mathcal{T}\to\mathcal{T}}^k$ and $T_{\mathcal{T}\to R_k}$ are a function only of the Φ parameters. It suffices to provide the derivative of Terms 1 and 2 w.r.t. Θ, and the derivative of Term 2 w.r.t. Φ. The rest of this section computes these quantities.

From Equation (3.1) in the definition of a global Markov chain

$$\iota_{init}^{\mathcal{PM}^\varphi,\mathcal{G}}([s,g]) = \iota_{init}^{\mathcal{PM}^\varphi}(s)\kappa(g|\iota_{init}^{\mathcal{PM}^\varphi},\Theta) \quad \Longrightarrow \quad \frac{\partial \iota_{init}^{\mathcal{PM}^\varphi,\mathcal{G}}([s,g])}{\partial\theta_i} = \iota_{init}^{\mathcal{PM}^\varphi}(s)\frac{\partial\kappa(g|\iota_{init}^{\mathcal{PM}^\varphi},\Theta)}{\partial\theta_i}, \tag{4.29}$$

where $\frac{\partial\kappa(g|\iota_{init}^{\mathcal{PM}^\varphi},\Theta)}{\partial\theta_i}$ can be computed using Equation (4.17).

Next, it is shown how to compute the gradient of a general entry in the matrix $T^{\mathcal{PM}^\varphi,\mathcal{G}}$. From Equation (3.2)

$$\begin{aligned} T^{\mathcal{PM}^\varphi,\mathcal{G}}([s',g']|[s,g],\Phi) &= \sum_{o\in\mathcal{O}}\sum_{\alpha\in Act} O(o|s)\omega(g',\alpha|g,o)T(s'|s,\alpha) \\ \Longrightarrow \quad \frac{\partial T^{\mathcal{PM}^\varphi,\mathcal{G}}([s',g']|[s,g],\Phi)}{\partial\phi_{\bar{g}\bar{\alpha}|\bar{g}\bar{o}}} &= \sum_{o\in\mathcal{O}}\sum_{\alpha\in Act} O(o|s)\tfrac{\partial\omega(g',\alpha|g,o)}{\partial\phi_{\bar{g}\bar{\alpha}|\bar{g}\bar{o}}}T(s'|s,\alpha), \end{aligned} \tag{4.30}$$

where $\frac{\partial\omega(g',\alpha|g,o)}{\partial\phi_{\bar{g}\bar{\alpha}|\bar{g}\bar{o}}}$ is computed using Equation (4.15).

Finally, the following shows how to compute the gradient of the infinite sum in Term 2. From Lemma 4.2.3,

$$\lim_{t\to\infty}\sum_{k=0}^{t} T_{\mathcal{T}\to\mathcal{T}}^k = (I - T_{\mathcal{T}\to\mathcal{T}})^{-1}. \tag{4.31}$$

This implies that,

$$\begin{aligned} \nabla_\Phi\left(\lim_{t\to\infty}\textstyle\sum_{k=0}^{t} T_{\mathcal{T}\to\mathcal{T}}^k\right) &= \nabla_\Phi\left((I - T_{\mathcal{T}\to\mathcal{T}})^{-1}\right) \\ &= -(I - T_{\mathcal{T}\to\mathcal{T}})^{-1}\nabla_\Phi(I - T_{\mathcal{T}\to\mathcal{T}})(I - T_{\mathcal{T}\to\mathcal{T}})^{-1} \\ &= +(I - T_{\mathcal{T}\to\mathcal{T}})^{-1}\nabla_\Phi T_{\mathcal{T}\to\mathcal{T}}(I - T_{\mathcal{T}\to\mathcal{T}})^{-1} \\ &= \left(I + T_{\mathcal{T}\to\mathcal{T}} + T_{\mathcal{T}\to\mathcal{T}}^2\dots\right)\nabla_\Phi T_{\mathcal{T}\to\mathcal{T}}\left(I + T_{\mathcal{T}\to\mathcal{T}} + T_{\mathcal{T}\to\mathcal{T}}^2\dots\right), \end{aligned} \tag{4.32}$$

where line 1 implies line 2 using linear algebra identities [112] and can be derived easily by differentiating both sides of the equation $(I - T_{\mathcal{T}\to\mathcal{T}})(I - T_{\mathcal{T}\to\mathcal{T}})^{-1} = I$. Thus the computation has been reduced to computing $\nabla_\Phi T_{\mathcal{T}\to\mathcal{T}}$ which is done using Equation (4.30).

In closing, the aggregate of these computations yield the gradient of the probability of satisfaction of φ when the structure of the FSC is fixed. For $\nabla \in \{\nabla_\Theta, \nabla_\Phi\}$

$$\begin{aligned}\nabla \Pr(\mathcal{PM} \vDash \varphi) &= \nabla \Pr\left(\pi \to \varphi\text{-}RecSets^{\mathcal{G}} | \iota_{init}^{\mathcal{PM}^{\varphi}}\right) \\ &= \sum_{R_k \subseteq \varphi\text{-}RecSets^{\mathcal{G}}} \nabla \Pr\left(\pi \to R_k | \iota_{init}^{\mathcal{PM}^{\varphi}}\right).\end{aligned} \tag{4.33}$$

From Equation (4.22)

$$\begin{aligned}\nabla_{\Theta} \Pr\left(\pi \to R_k | \iota_{init}^{\mathcal{PM}^{\varphi}}\right) &= \left(\nabla_{\Theta} \vec{\iota}_{init}^{\mathcal{PM}^{\varphi},\mathcal{G}}(R_k)\right)^T \vec{1}_{|R_k|\times 1} \\ + \left(\nabla_{\Theta} \vec{\iota}_{init}^{\mathcal{PM}^{\varphi},\mathcal{G}}(\mathcal{T})\right)^T &\left(I + T_{\mathcal{T}\to\mathcal{T}} + T_{\mathcal{T}\to\mathcal{T}}^2 + \ldots\right) T_{\mathcal{T}\to R_k} \vec{1}_{|R_k|\times 1} \quad ,\end{aligned} \tag{4.34}$$

and

$$\begin{aligned}\nabla_{\Phi} \Pr\left(\pi \to R_k | \iota_{init}^{\mathcal{PM}^{\varphi}}\right) &= \underbrace{\left(\vec{\iota}_{init}^{\mathcal{PM}^{\varphi},\mathcal{G}}(\mathcal{T})\right)^T \nabla_{\Phi}\left(I + T_{\mathcal{T}\to\mathcal{T}} + T_{\mathcal{T}\to\mathcal{T}}^2 + \ldots\right) T_{\mathcal{T}\to R_k} \vec{1}_{|R_k|\times 1}}_{\text{Grad Term 1}} \\ &+ \underbrace{\left(\vec{\iota}_{init}^{\mathcal{PM}^{\varphi},\mathcal{G}}(\mathcal{T})\right)^T \left(I + T_{\mathcal{T}\to\mathcal{T}} + T_{\mathcal{T}\to\mathcal{T}}^2 + \ldots\right) \nabla_{\Phi} T_{\mathcal{T}\to R_k} \vec{1}_{|R_k|\times 1}}_{\text{Grad Term 2}} .\end{aligned} \tag{4.35}$$

4.2.4.1 Complexity and Efficient Computation

For the gradients of Terms 1 and 2, one source of computational complexity are the infinite sums $\nabla_{\Phi}\left(I + T_{\mathcal{T}\to\mathcal{T}} + T_{\mathcal{T}\to\mathcal{T}}^2 + \ldots\right)$ and $\left(I + T_{\mathcal{T}\to\mathcal{T}} + T_{\mathcal{T}\to\mathcal{T}}^2 + \ldots\right)$ respectively, where computing each power term successively has $\mathbf{O}(|\mathcal{T}|^3)$ complexity. However, this computation can be reduced to quadratic complexity by using a similar trick as in Section 4.2.3.1.

First, in gradient of Term 2, note that the infinite sum is pre-multiplied by a row vector $\left(\vec{\iota}_{init}^{\mathcal{PM}^{\varphi},\mathcal{G}}\right)^T$. To compute this product the following iteration can be setup. Initialize variables v'_o and x'_0

$$\begin{aligned} v'_0 &= \left(\vec{\iota}_{init}^{\mathcal{PM}^{\varphi},\mathcal{G}}\right)^T, \quad \text{and} \\ x'_0 &= \mathbf{0}. \end{aligned} \tag{4.36}$$

Then, carry out the iteration

$$\begin{aligned} v'_{n+1} &= v'_n T_{\mathcal{T}\to\mathcal{T}}, \quad \text{and} \\ x'_{n+1} &= x'_n + v'_n \end{aligned} \tag{4.37}$$

until $||x'_{n+1} - x'_n||_{\infty} \leqslant \varepsilon_{x'}$, where $\varepsilon_{x'}$ is a given tolerance. Next, for the gradient of Term 1, rewrite the second term in Equation (4.35) using Equation (4.32)

$$\nabla_{\Phi}\text{Term 1} = \underbrace{\left(\iota_{init}^{\mathcal{PM}^{\varphi},\mathcal{G}}\right)^{T}\left(I + T_{\mathcal{T}\to\mathcal{T}} + T_{\mathcal{T}\to\mathcal{T}}^{2} + \dots\right)}_{\text{Term A}}\nabla_{\Phi}T_{\mathcal{T}\to\mathcal{T}}\underbrace{\left(I + T_{\mathcal{T}\to\mathcal{T}} + T_{\mathcal{T}\to\mathcal{T}}^{2} + \dots\right)T_{\mathcal{T}\to R_k}\vec{1}_{|R_k|\times 1}}_{\text{Term B}}. \tag{4.38}$$

Note that Terms A and B are the same quantities that are computed in the iterative schemes of Equations (4.36-4.37) and (4.26-4.27) respectively.

For the gradient of the absorption probability w.r.t. Φ, as shown in Equation (4.34), all that remains now is to establish the overall complexity of computing $\nabla_{\Phi}T$. This is can be quite an expensive operation, since the gradient must be taken for each element of T with respect to each $\phi \in \Phi$. In the worst case, the complexity is given by $\mathbf{O}(|\mathcal{S}|^2|G|^2|\Phi||Act||\mathcal{O}|)$. However, for systems that are described by sparse transition and observation functions, the practical complexity is $\mathbf{O}(c|\mathcal{S}||G||\Phi||Act|)$ with $c \ll |\mathcal{G}||G||\mathcal{O}|$.

For the gradient w.r.t Θ, the complexity of evaluating $\nabla_{\Theta}\iota_{init}^{\mathcal{PM}^{\varphi},\mathcal{G}}(R_k)$ is given by $\mathbf{O}(|\mathcal{S}|^2|\Theta|)$.

4.2.4.2 Gradient Based Optimization

In the preceding sections, the analytical expression for the gradient of satisfaction probability was derived. This gradient can be used in first order methods [150] to search or optimize the following objective

$$\begin{aligned} & \max_{\Phi,\Theta} \ \Pr(\mathcal{PM} \vDash \varphi) \\ \implies & \max_{\Phi,\Theta} \ \Pr\left(\pi \to \varphi\text{-}RecSets^{\mathcal{G}} | \iota_{init}^{\mathcal{PM}^{\varphi}}\right) \\ \implies & \max_{\Phi,\Theta} \sum_{R_k \subseteq \varphi\text{-}RecSets^{\mathcal{G}}} \Pr\left(\pi \to R_k | \iota_{init}^{\mathcal{PM}^{\varphi}}\right) \end{aligned} \tag{4.39}$$

over the parameters Φ and Θ.

4.3 Ensuring Non-Infinitesimal Frequency of Visiting $Repeat^{\mathcal{PM}^{\varphi}}$ States

Even though the soft-max parametrization ensures that the structure and hence the recurrent classes of the global Markov chain remains unchanged with parameters, in numerical algorithms it can lead to undesirable steady state behavior. To understand this, consider the global Markov chain in Figure 4.2. It is possible that the optimization in Equation (4.39) causes the $Repeat^{\mathcal{PM}^{\varphi}}$ states to be visited with infinitesimal frequency. While this characteristic will satisfy φ for very long runs, in real applications it is reasonable to prefer that $Repeat^{\mathcal{PM}^{\varphi}}$ is visited often in steady state.

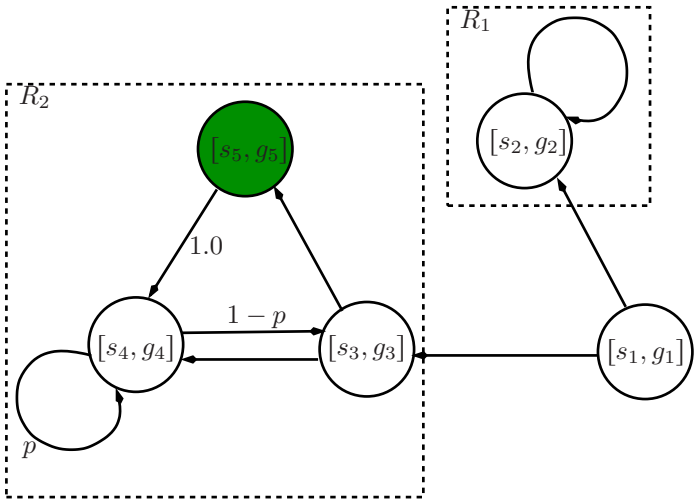


Figure 4.2: $Repeat^{\mathcal{PM}^\varphi}$ can be visited with vanishing frequency.
The projection of global state $[s_5, g_5]$ onto the product POMDP state is given by s. Assume that it is the only product state that belongs to a $Repeat^{\mathcal{PM}^\varphi}$ set. It is possible that maximizing probability of absorption into R_2 causes p to grow arbitrarily close to 1, causing state $[s_5, g_5]$ to be visited with infinitesimally small frequency in steady state.

A high frequency of visiting the set $Repeat^{\mathcal{PM}^\varphi}$ can be accomplished by looking at each φ-feasible recurrent set R_k and identifying the set that needs to be visited often, denoted $Good_k \subseteq R_k$. This is done in Algorithm 4.1 below.

Algorithm 4.1 Generate Set To Visit Frequently

Input: φ-feasible recurrent set R_k, Rabin acceptance pairs of product-POMDP $\Omega^{\mathcal{PM}^\varphi}$

$Good_k = \varnothing$
for all $(Repeat_i^{\mathcal{PM}^\varphi}, Avoid_i^{\mathcal{PM}^\varphi}) \in \Omega^{\mathcal{PM}^\varphi}$ **do**
  if $(Avoid_i^{\mathcal{PM}^\varphi} \times G) \cap R_k = \varnothing$ **then**
    $Good_k = Good_k \cup \left((Repeat_i^{\mathcal{PM}^\varphi} \times G) \cap R_k\right)$
  end if
end for
return $Good_k$

The algorithm simply identifies those Rabin acceptance pairs that are consistent with the recurrent set R_k, and then selects those states from R_k that project onto the $Repeat^{\mathcal{PM}^\varphi}$ part of the Rabin pair. Next, the goal is to ensure that at least some state(s) in the set $Good_k$ is(are) visited frequently in *steady state*. Recall that steady state implies that the path is already absorbed in R_k. The quantity of interest is given by the empirical or pathwise occupation measure from Equation (3.23) by setting $A \leftarrow Good_k$. Writing out the modified equation explicitly gives

$$\pi^{(t)}(Good_k|s_0) = \frac{1}{t}\sum_{k=1}^{t} \mathbb{1}\left([s_k, g_k] \in Good_k\right), \; t = 1, 2, \ldots \tag{4.40}$$

Then the *expectation* of the pathwise occupation measure is taken with the assumption that

paths are *already* absorbed in R_k, thus implying steady state behavior of the Markov chain. This can be done by additionally taking the expectation due to an initial distribution ι^{ss}_{init} whose support is in R_k. Additionally, the horizon is taken to be infinite to reflect long term steady state. Formally, this done by modifying the Equation (3.24) in the following way to compute the expected pathwise occupation measure as the horizon goes to ∞.

$$\begin{aligned}\lim_{t\to\infty}\mathbb{E}\left[\pi^{(t)}(Good_k)|\iota^{ss}_{init}\right] &= \lim_{t\to\infty}\mathbb{E}\left[\frac{1}{t}\sum_{k=1}^{t}\mathbb{1}\left([s_k,g_k]\in Good_k\right)|\iota^{ss}_{init}\right] \\ &= \sum_{[s,g]\in R_k}\iota^{ss}_{init}\Big([s,g]\Big)\left(\lim_{t\to\infty}T^{(t)}(Good_k|[s,g])\right), \end{aligned} \tag{4.41}$$

$$\text{where}\sum_{[s,g]\in R_k}\iota^{ss}_{init}\Big([s,g]\Big) = 1$$

ensures that $support(\iota^{ss}_{init})\subseteq R_k$.

Equation (4.41) can be rewritten using the vector and matrix representation of the above quantities as follows

$$\begin{aligned}\lim_{t\to\infty}\mathbb{E}\left[\pi^{(t)}(Good_k)|\iota^{ss}_{init}\right] &= (\vec{\iota}^{ss}_{init})^T\left(\lim_{t\to\infty}T^{(t)}\right)\vec{1}^{\mathcal{S}\times G}_{Good_k} \\ &= (\vec{\iota}^{ss}_{init})^T\,\Pi\vec{1}^{\mathcal{S}\times G}_{Good_k}, \end{aligned} \tag{4.42}$$

where line 1 leads to line 2 using the limiting matrix, Π, as introduced in Definition 3.4.12.

4.3.1 Equivalence to Expected Long Term Average Reward

Proposition 4.3.1 *: Consider the reward structure over the global state space*

$$r\left([s,g]\right) = \begin{cases} 1 & \text{if } [s,g]\in Good_k \\ 0 & \text{otherwise.}\end{cases} \tag{4.43}$$

Then the expected long term average reward is the same as the expected occupation measure of set $Good_k$, i.e.,

$$\eta_{av}(R_k) = \lim_{t\to\infty}\left[\frac{1}{t}\sum_{k=0}^{t}r_k\,|\iota^{ss}_{init}\right] = \lim_{t\to\infty}\mathbb{E}\left[\pi^{(t)}(Good_k)|\iota^{ss}_{init}\right] \tag{4.44}$$

where r_k is the reward obtained at time step k.

This is an important relationship, as the long term average reward and the computation of its gradient is studied extensively in the literature, especially for the case of the Markov chain being ergodic (or having a single recurrent class) [1, 10]. Our restriction on the support of the initial distribution ensures that the Markov chain evolves exclusively in a single recurrent class R_k, and therefore the methods described in these works can be directly utilized. The derivations of the gradients $\nabla_\Theta\eta_{av}(R_k)$ and $\nabla_\Phi\eta_{av}(R_k)$ are skipped, and the reader is referred to [1] for a detailed

description. The complexity of evaluating the gradient is summarized from [1] here to complete the view of computational burden of the gradient ascent methodology.

4.3.1.1 Complexity of Computing Gradient of $\eta_{av}(R_k)$

From [1], in the worst case, the complexity of computing $\nabla_\Phi \eta_{av}(R_k)$ is given by $\mathbf{O}(|\mathcal{S}|^2|G|^2|\Phi||Act||\mathcal{O}|)$ similar to the gradient of absorption probability. The reduced practical complexity for sparse transition and observation functions applies as well and is given by $\mathbf{O}(c|\mathcal{S}||G||\Phi||Act|)$ with $c \ll |\mathcal{G}||G||\mathcal{O}|$.

The gradient of $\eta_{av}(R_k)$ w.r.t. Θ is zero.

4.4 Trade Off between Absorption Probability and Visitation Frequency

Having a high frequency of visiting $Good_k$ is best viewed as a constraint on the optimization problem in Equation (4.39). However, in order to formally construct the constrained problem, it is no longer possible to look at the union of all φ-feasible recurrent sets at once. To understand this, consider the following events over the global Markov chain.

1. $p_k := \pi \to R_k,\ R_k \subseteq \varphi\text{-}RecSets^{\mathcal{G}}$: Path is absorbed in φ-feasible recurrent set R_k.
2. $p := \pi \to \bigcup_{R_k \subseteq \varphi\text{-}RecSets^{\mathcal{G}}} R_k$: Path is absorbed in some φ-feasible recurrent set R_k.
3. $q_k := \eta(R_k) \geqslant c_k > 0$: States $Good_k \subseteq R_k$ are visited with positive frequency.

Ideally, any algorithms should attempt to maximize the following objective.

$$\max_{\Phi,\Theta} \ \Pr\left[p \wedge \bigwedge_{R_k \subseteq \varphi\text{-}RecSets^{\mathcal{G}}} (p_k \implies q_k) \right], \tag{4.45}$$

in which q_k is a constraint that is selectively applied depending on which recurrent set the path is absorbed into. This is difficult to pose as a single optimization problem. Instead, one constrained optimization problem for each φ-feasible recurrent set must be solved as follows

$$\max_{\Phi,\Theta} \ \Pr[p_k \implies q_k]. \tag{4.46}$$

In terms of $\Lambda(R_k)$, which denotes the absorption probability (Equation (4.22)), and $\eta_{av}(R_k)$, which is the expected long term average reward (Equation (4.44)), this optimization can be written as

$$\begin{aligned} &\max_{\Phi,\Theta} && \Lambda(R_k) \\ &\text{subject to} && \eta_{av}(R_k) \geqslant c_k. \end{aligned} \tag{4.47}$$

The above optimization problem is still difficult to solve because the constraint is non-linear and it is unclear how to explore the feasible set for it. Instead, the constrained problem can be transformed into a unconstrained problem using a barrier function, such as the log barrier, to give

$$\max_{\Phi,\Theta} \quad \underbrace{\Lambda(R_k) + \frac{1}{c'_k} \log\left(\eta_{av}(R_k) - c_k\right)}_{\Gamma(R_k)}. \tag{4.48}$$

The preceding sections showed how to compute the gradient of $\Lambda(R_k)$ and $\eta_{av}(R_k)$. Hence, first order methods for unconstrained optimization can be applied. The case studies presented in Section 4.7 utilize adaptive step size gradient ascent. The parameters c_k, c'_k, need to be hand tuned, usually after studying the behavior of the system, the feasible set, and the need to have good steady state visitation frequency. Moreover, the objective is non-linear, and first order methods can only guarantee convergence to a local maximum.

The overall best FSC parametrization is obtained by taking the maximum over all φ-feasible recurrent sets. Let $\Gamma^*(R_k)$ be the (local) optimum value obtained from Equation (4.48) and the corresponding optimizing parameters are given by $\mathcal{G}^*(R_k) = \{\Phi^*(R_k), \Theta^*(R_k)\}$. Then the optimum value is taken to be

$$\max_{R_k \subseteq \varphi\text{-}RecSets^{\mathcal{G}}} \quad \Gamma^*(R_k). \tag{4.49}$$

The optimum controller is given by $\mathcal{G}^*(R_k^*) = \{\Phi^*(R_k^*), \Theta^*(R_k^*)\}$, where

$$R_k^* = \operatorname*{argmax}_{R_k \subseteq \varphi\text{-}RecSets^{\mathcal{G}}} \quad \Gamma^*(R_k). \tag{4.50}$$

4.5 Heuristic Search for FSC Structures with a φ-Feasible Recurrent Set

In this section it shown how, given a fixed size $|G|$, candidate FSCs that yield at least one φ-feasible recurrent set can be generated. This problem is hard [33] in itself – the hardness arising out of partial observability in which possibly unbounded sequences of actions and observations may be required to ensure that some states are never visited. However, the heuristic described in this section restricts the search over outcomes that can be inferred by a single, most recent, observation and action. Thus, the proposed method is incomplete, in which a solution may exist, but the algorithm is unable to find it. The details of this heuristic search is given in Algorithm 4.2.

In order to understand the algorithm, the reader is pointed to the example in Figure 4.3. It shows a part of a global Markov chain, such that the underlying product POMDP has only one Rabin acceptance pair. The global state, $[s_4, g_4]$, denoted in green in Figure 4.3, is the only global

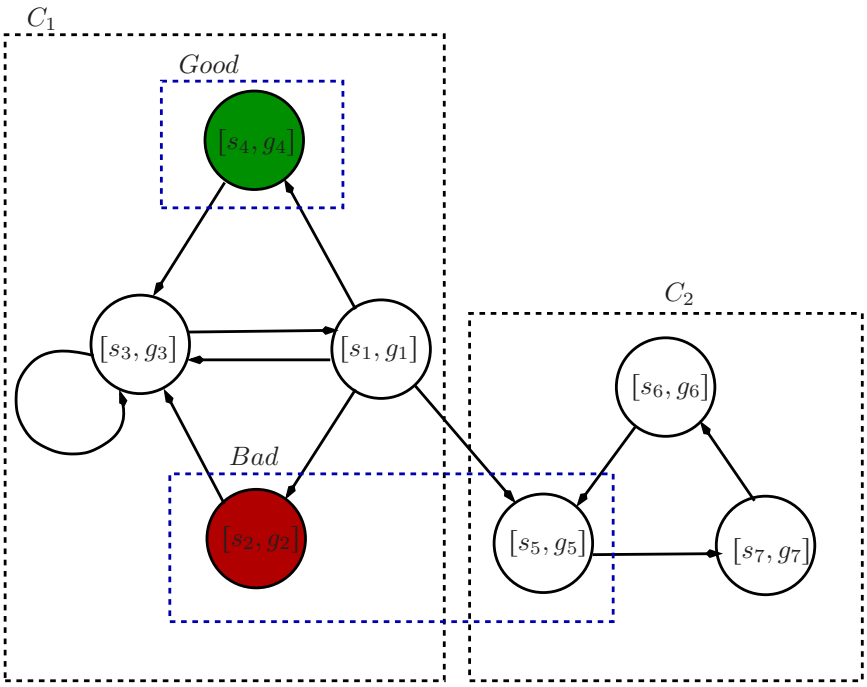


Figure 4.3: Generating Admissible Structures of FSC
There are two communicating classes C_1 and C_2. In steady state it must be ensured that the green node is recurrent, while the red node is never visited, i.e., $[s_2, g_2]$ must be disconnected from $[s_1, g_1]$. Since C_1 can lead to C_2, which is absorbing, the communication between $[s_1, g_1]$ and $[s_5, g_5]$ needs to be severed. Thus, in Algorithm 4.2, $Good = \{[s_4, g_4]\}$, and $Bad = \{[s_2, g_2], [s_5, g_5]\}$.

state whose projection $s_4 \in Repeat_1^{\mathcal{P}\mathcal{M}^\varphi}$. The global state, $[s_2, g_2]$, denoted in red, is such that $s_2 \in Avoid_1^{\mathcal{P}\mathcal{M}^\varphi}$. There are two communicating classes, C_1, and C_2, in the global Markov chain s.t. $C_1 \to C_2$, and C_2 is absorbing. Therefore the states in $Bad = \{[s_2, g_2], [s_5, g_5]\}$ need to be made unreachable in steady state, while ensuring that some state in $Good = \{[s_4, g_4]\}$ is recurrent. The former is done in steps 14-15 of Algorithm 4.2 by disallowing actions which lead to bad states under the latest observation. Recurrence of some state in *Good* is ensured in steps 19-20. This recurrence may not always be guaranteed because disconnecting states in *Bad* by removing actions may change the communication properties of other global states. The check for $\sum_{g_l \in G, \alpha \in Act} \mathcal{I}_\omega(g_l \alpha | g_k, o) > 0$ in steps 9 and 17 makes sure that the modification in $\mathcal{I}_\omega$ does not yield an inadmissible structure as defined in Equation (4.3).

Note, that no discussion about $\mathcal{I}_\kappa$ has been made in the context of feasibility. This is because setting $\mathcal{I}_\kappa(g) = 1$, $\forall g \in G$, is sufficient and this choice does not affect the φ-feasibility.

4.5.1 Complexity

Algorithm 4.2, presents two main sources of computational complexity. First is the computation of strongly connected components. For a graph $\mathbb{G}$, these components can be found with effort

Algorithm 4.2 Generate Candidate FSCs

Input: Fixed FSC I-States G, structure $\mathcal{I}_\omega$, product-POMDP transition probabilities T^φ

Candidate FSC structures $\mathbb{I} = \varnothing$
Construct the directed graph $\mathbb{G}$ of the global Markov chain under $\mathcal{I}_\omega$
Find communicating classes $\mathcal{C} = \{C_1, \ldots, C_N\}$ by computing the strongly connected components $\mathcal{C} = sccs(\mathbb{G})$.
for all $C \in \mathcal{C}$ **and** $(Repeat_r^{\mathcal{PM}^\varphi}, Avoid_r^{\mathcal{PM}^\varphi}) \in \Omega^{\mathcal{PM}^\varphi}$ **do**
  $Bad_r = \{\mathfrak{s}' \in C_d \neq C \text{ s. t. } \exists \mathfrak{s} \in C \text{ and } \mathfrak{s} \to \mathfrak{s}'\}$
  $Bad_r = Bad_r \cup \left(C \cap (Avoid_r^{\mathcal{PM}^\varphi} \times G)\right)$
  $Good_r = C \cap (Repeat_r^{\mathcal{PM}^\varphi} \times G)$
  Initialize $\mathcal{I}_\omega(g_l \alpha | g_k o) = 1$, $\phi_{g_l \alpha | g_k o} = 0$, $\forall \alpha \in Act$, $o \in \mathcal{O}$, $g_k, g_l \in G$
  while $\sum_{g_l \in G, \alpha \in Act} \mathcal{I}_\omega(g_l \alpha | g_k o) > 0$, $\forall o \in \mathcal{O}$, $g_k \in G$ **and** $Bad \neq \varnothing$ **do**
    Pick $\mathfrak{s}' = [\langle s_j, q_n \rangle, g_l] \in Bad_{k,r}$
    for all $\mathfrak{s} = [\langle s_i, q_m \rangle, g_k] \in R_k$ **do**
      for all $\alpha \in Act$ **do**
        $\omega(g_l, \alpha | \Phi, g_k, o) = \frac{\mathcal{I}_\omega(g_l, \alpha | g_k, o)}{\sum_{g'} \sum_{\alpha'} \mathcal{I}_\omega(g_l, \alpha' | g_k, o)}$
        if $O(o|s_i)\omega(g_l, \alpha | \phi, g_k, o) T^\varphi(\langle s_j, q_n \rangle | \langle s_i, q_m \rangle, \alpha) > 0$ **then**
          $\forall g_k, g_l \in G$, $\mathcal{I}_{g_l \alpha | g_k o} \leftarrow 0$
        end if
      end for
    end for
    $Bad_r \leftarrow Bad_r \backslash \{\mathfrak{s}'\}$
  end while
  if $\sum_{g_l \in G, \alpha \in Act} \mathcal{I}_\omega(g_l, \alpha | g_k, o) > 0, \forall o \in \mathcal{O}, g_k \in G$ **then**
    Construct the directed graph $\mathbb{G}'$ of the global Markov chain under modified $\mathcal{I}_\omega$
    if $\exists \mathfrak{s} \in Good_r$ s.t. $\mathfrak{s}$ is recurrent under $\mathbb{G}$ **then**
      $\mathbb{I} \leftarrow \mathbb{I} \cup \{\mathcal{I}_\omega\}$
    end if
  end if
end for
return $\mathbb{I}$

Output: The set of admissible structures $\mathbb{I} = \{\mathcal{I}_\omega | \text{Resulting FSC has } \varphi\text{-feasible recurrent sets}\}$

proportional to $\mathbf{O}(|V|+|E|)$, where V are the set of nodes and E are the set of edges. For the global Markov chain, the number of nodes equals $|\mathcal{S}||G|$, while the worst case number of edges is given by $|\mathcal{S}|^2|G|^2$. Taking the higher of the two, this gives steps 2 and 18 a complexity of $\mathbf{O}(|\mathcal{S}|^2|G|^2)$. On the other hand the nested loops in steps 4-16, have the worst case complexity of $\mathbf{O}(|\mathcal{S}|^2|G|^2|Act||\mathcal{O}|)$. Therefore, the overall complexity is the later expression $\mathbf{O}(|\mathcal{S}|^2|G|^2|Act||\mathcal{O}|)$.

4.6 Initialization of Θ and Φ

The last algorithmic aspect that should be mentioned is that once a feasible structure has been found for the optimization problem in Equation (4.48), Θ and Φ need to be initialized. Giving equal probability to all I-state transitions and actions, reflected in identical initialization of parameters, may seem like the best choice, since that assumes least amount of prior information. However, as described in [1], this may lead to uniformly zero gradients. While the cited work talks exclusively about the gradient of $\eta_{av}(R_k)$ w.r.t. Φ, it is equally applicable to the gradients w.r.t. Θ and also to the gradient of $\Lambda(R_k)$. This arises because the quantities whose gradients are being computed are expectations over infinite paths. Uniform parametrization of the controller may not be able to distinguish between probability measures of paths over long executions, thus giving zero gradient with respect to the parameters. This can be addressed by randomly initializing the parameters Θ and Φ.

In [1], the author also talks about the problem of soft-max saturation in which the gradient vanishes because the softmax saturates when some parameters are large as compared to others. The author recommends random initialization of parameters to small values, usually in $[-1, 1]$.

4.7 Case Studies

This section collects numerical examples that demonstrate the salient features of the policy gradient methodology presented in this chapter.

4.7.1 Case Study I - Repeated Reachability with Safety

System Model: This case study uses the system model described in Example 2.2.2. For convenience, the graphical representation is repeated in Figure 4.4(c) and it will be called POMDP-World. Figure 4.4(a) shows the deterministic version of the world, called Det-World, in which each action has a single deterministic outcome and full observability is assumed, that is, the robot knows its cell location perfectly. Part (b) of the figure shows a *fully observable* probabilistic world, called MDP-World. In this case the actions move right R, move left L, move up U and move down D, have probabilistic outcomes as in the case of the POMDP-World. The stop action X deterministically

causes the robot to remain in its current cell. These three models will allow comparison of some issues that arise from the use of gradient ascent methodology to design a controller.

LTL Specification: The LTL specification of interest for this case study is given by the formula

$$\varphi_1 = \square\diamondsuit a \wedge \square\diamondsuit b \wedge \square\neg c. \tag{4.51}$$

The intuitive notion is that the grid world in Figure 4.4 represents a segmentation or finite abstraction of a corridor. The red cell represents a stairwell that a wheeled robot is unsafe in, while it must regularly tend to both the green cells (perhaps to service instruments) indefinitely. Thus the specification tells the robot to ensure that it must visit both cells labeled a and b infinitely often, while ensuring that it never visits cell c. The former requirement is a liveness formula, while the latter is a safety formula (Section 2.1.2). The DRA for this formula is shown in Figure 4.5.

Results:

Feasibility

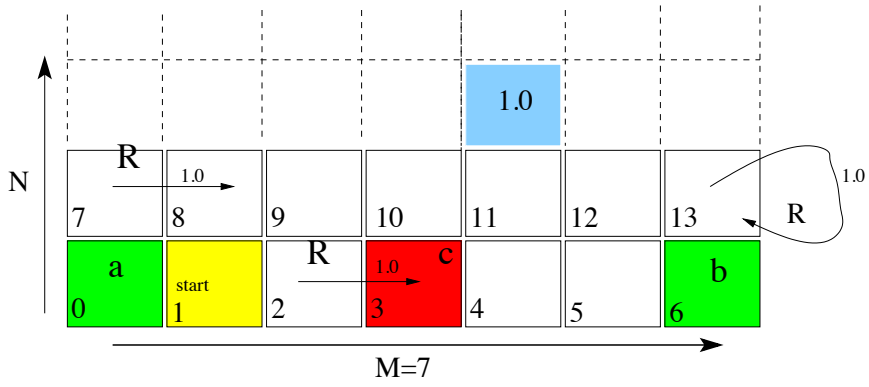
First, consider the difference in feasibility of the specification over the three different types of system model. It can be seen that the formula φ_1 is feasible for Det-World when the corridor width $N \geqslant 2$, while for the case of MDP-World and POMDP-World the system becomes feasible for $N \geqslant 3$. It is easy to see why $N = 2$ in the probabilistic models is infeasible: the robot can only move into cell 10 from the left by issuing a right signal R from cell 2 or 9, in which case it may end up in the red cell 3 due to probabilistic outcomes. The specification becomes feasible for $N = 3$ for both MDP-World and POMDP-World. For the MDP world, the robot needs to first ensure it has moved to cell 16 and then issue a move right signal R. Similarly, to move left from the right half of the grid world, if the robot issues move action L from cell 18, it is guaranteed to stay away from red cell 3. For the POMDP-World however, there is the additional burden for the robot to infer with full confidence that it is in cell 16 before issuing a move right action, and similarly guarantee location in cell 18 before issuing a move left action. This issue is visited for a slightly different grid world model again in Section 4.7.3.

Quality of Trajectory

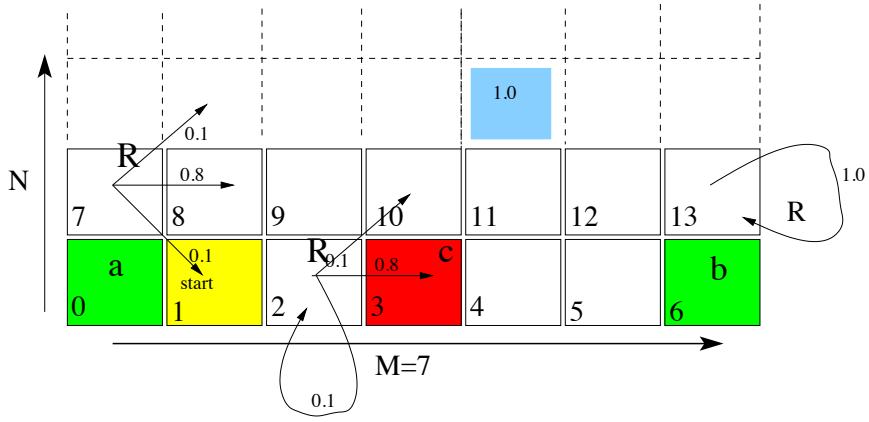
The performance of the policy gradient ascent algorithm starting from an initial feasible controller for different sizes of the FSC is presented. The experiment was conducted for $N = 4$. Recall that the objective function for the policy gradient ascent is given by Equation (4.48), repeated here

$$\max_{\Phi,\Theta} \quad \underbrace{\Lambda(R_k) + \frac{1}{c'_k}\log\left(\eta_{av}(R_k) - c_k\right)}_{\Gamma(R_k)}, \tag{4.52}$$

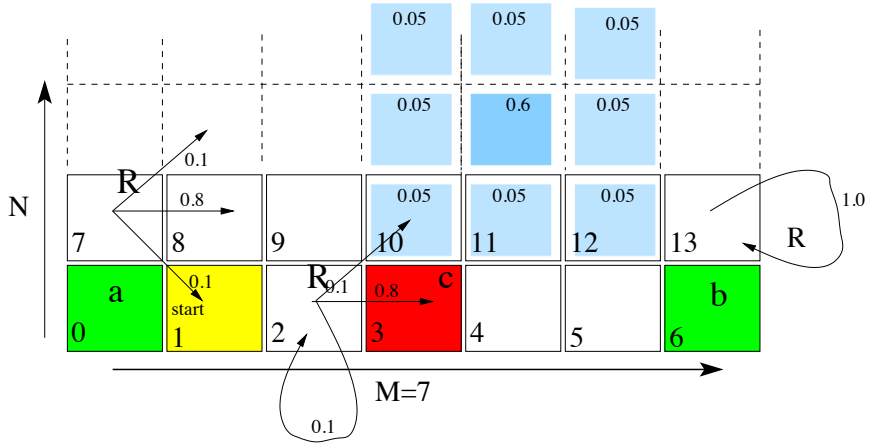
where the quantity $\Lambda(R_k)$ denotes the probability of absorption into a φ_1-feasible recurrent set and $\eta_{av}(R_k)$ indicates the frequency of "good events". For our example, a good event occurs every time



(a) Det-World



(b) MDP-World



(c) POMDP-World

Figure 4.4: System Models for Case Study I. (a) A planar grid world in which a robot moves deterministically, i.e., each action has a single outcome. Outcomes for moving right ($\alpha = R$) shown above. The actions for moving left L, up U, and down D respectively, are similarly defined. In addition there is a fifth action *Stop*, denoted X, which causes the robot to remain in its current cell. The robot additionally knows its location (cell number) without ambiguity. (b) A probabilistic world, in which the robot actions R, L,, U and D have probabilistic outcomes, while X results in the robot remaining in its current cell deterministically. The difference in behavior in different types of cells (interior,edge, or corner cells) should be noted. Action L, U, and D are symmetrically defined. The robot can still locate itself without ambiguity, thereby making the domain an MDP. (c) Partially observable probabilistic grid world. The robot moves probabilistically when commanded, with same values as the MDP. However, it can no longer measure its location (cell) accurately. When the robot is in the dark blue cell, the probabilities of getting a measurement which suggests a location in a neighboring cell are indicated by the neighboring light blue cells as shown.

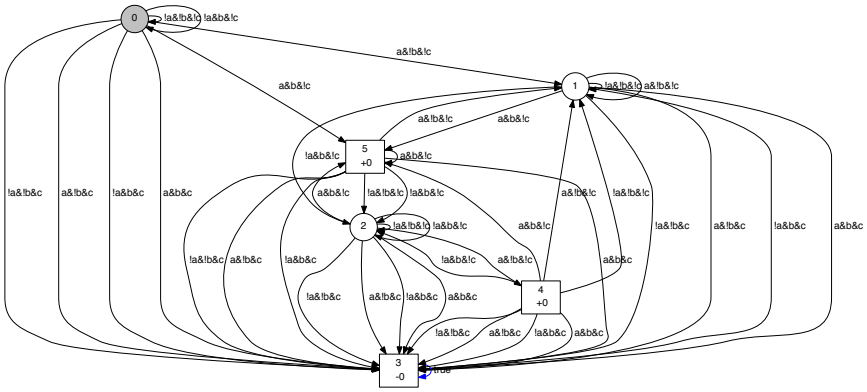


Figure 4.5: DRA for the LTL specification $\varphi_1 = \square\diamondsuit a \wedge \square\diamondsuit b \wedge \neg c$. There is one accepting condition indexed by +/-0. State 3, labeled by -0 belongs to the $Avoid_1$ set and therefore must be avoided during steady state. However, the state numbered 3 is a sink and therefore to maximize the probability of absorption into a φ_1-feasible set, visits to this state must be avoided during transience as well. It so happens that state 5, which belongs to the $Repeat_1$, is in fact not reachable when the product is taken with the model. Therefore, it must be ensured that state 4 is visited infinitely often.

both a and b cells are visited since the last good event. This means that a sequence a, a, a, a, b counts as 1 good event and the process that indicates good events resets.

It was found that all feasible controllers trivially ensure $\Lambda(R_k) = 1$, so that the optimization simply serves to improve the frequency of good events. The dependence of η_{av} on the number of FSC I-states is shown in Figure 4.6. Note that, the gradient ascent algorithm does not indicate how to automatically grow the number of I-states at a local maxima, so each trial was seeded by the number of I-states shown in the figure. Note the large increase in the value of η_{av} every time an I-state is added for small values of $|G|$. This fast increase can be interpreted as follows. The FSC's performance will increase if it can keep track of whether it is trying to go to cell b or to cell a. This fact already points to the need for at least two I-states to realize good performance. In addition, utilizing more I-states allows for optimization over longer observation/action histories, thus making up for partial observability. Another perspective is that for small numbers of I-states, the controller tries to achieve its goal of recurrence by leveraging stochasticity: by producing random sequence of actions, some sub-sequences are bound to make the state visit a or b. As the controller gains more internal states, the control becomes more deliberate, taking better account of the system dynamics and observations.

4.7.2 Case Study II - Stability with Safety

This case study demonstrates that the probability of LTL satisfaction can be negatively effected by partial observability. Control design and execution results for the POMDP-World will be compared with that of fully observable MDP cases.

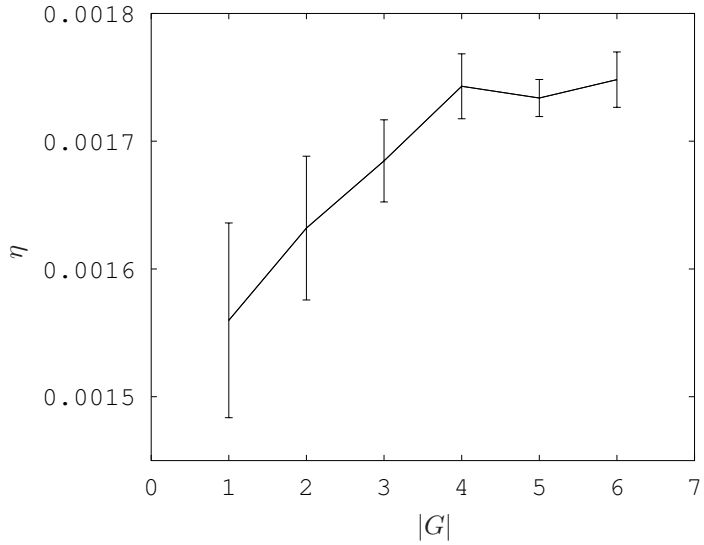


Figure 4.6: Dependence of expected steady state average reward η on size of FSC, for POMDP-World, N=4. Since Φ, Θ are initialized randomly, mean and standard deviation for 5 samples each is shown. The hand tuned parameters in Equation (4.48), were taken to be $c_1' = 1, c_1 = 0$.

System Model

For this case study, consider a different grid world, as shown in Figure 4.7. The motion and observation models are the same as in Case Study I. Only the size and cell labelings are different.

LTL Specification

In this case study the LTL formula of concern is given by

$$\varphi_2 = (\diamondsuit \square a \vee \diamondsuit \square b) \wedge \square \neg c. \tag{4.53}$$

In other words, the specification dictates that the robot should navigate to either green cell 36 or green cell 41, and stay there. But the robot must avoid all the red cells (cells 14 and 24). Recall that the requirement in the parentheses is a stability requirement while the requirement $\square \neg c$ is a safety formula as defined in Section 2.1.2.

Results

First note that the full formula φ_2 can be satisfied with probability 1 (almost sure satisfaction) for both MDP and POMDP versions. For the MDP case, visiting either cell 36 or cell 41 is acceptable. For POMDP-World the resulting FSC accomplishes almost sure satisfaction by choosing to make proposition b true. Thus the robot navigates to green cell 41. This is because an attempt to navigate to cell 36 can cause the robot to visit either cell 14 or 24 depending on the path it takes. Thus, for both MDP and POMDP if the gradient ascent algorithm is used, at optimality the probability of LTL satisfaction was found to be $\Lambda = 1$. In addition, once the robot reaches cell 41, it must ensure it stays there forever, thus requiring $\eta_{av} = 1$, which is the maximum value that η can take. The

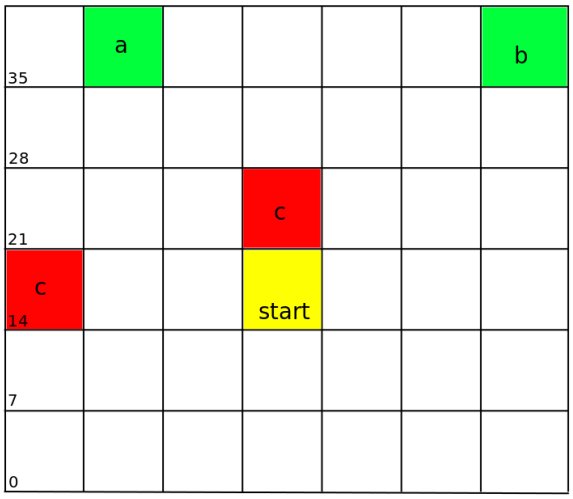


Figure 4.7: System Model for Case Study II

	$\Lambda(\varphi_2(a))$	$\Lambda(\varphi_2(b))$
MDP	1.000	1.000
POMDP	0.974	1.000

Table 4.1: Results for GW-B under φ_2

algorithm converged to the optimal values for both Λ and η. However, there is a difference between the gradient ascent algorithm and the typical solution methodology for MDPs. To understand this, first split φ_2 into two formulas:

$$\begin{aligned} \varphi_2(a) &= \Diamond\Box a \wedge \Box\neg c \text{ and} \\ \varphi_2(b) &= \Diamond\Box b \wedge \Box\neg c \text{ with} \\ \varphi_2 &= \varphi_2(a) \vee \varphi_2(b) \end{aligned} \tag{4.54}$$

That is, φ_2 is a disjunction of two smaller formulas. When the gradient ascent algorithm was run *separately* for these two formulas, the absorption probability obtained are given in Table 4.1. For the POMDP case, the gradient ascent algorithm picks φ_2 and visiting cell 41 precisely because it has larger Λ of the two. When the gradient ascent algorithm is applied to the MDP case, in which both formulas have the same probability of satisfaction, the algorithm does not pick a favorite. It finds two controllers of equal satisfaction probabilities, and the tie needs to be broken by some rule. This is because two candidate φ-$RecSets^{\mathcal{G}}$ are generated, and an FSC is optimized for each of them separately. This is different from the typical MDP solution methodology [8, 40], where a *shortest path problem* is solved in one shot to the union of all candidate φ-feasible recurrent sets. For the initial start state as shown in Figure 4.7, the shortest path algorithm would always steer the robot to cell 36, even when the formula φ_2 is satisfied with probability 1 by navigating to either cell a or cell b.

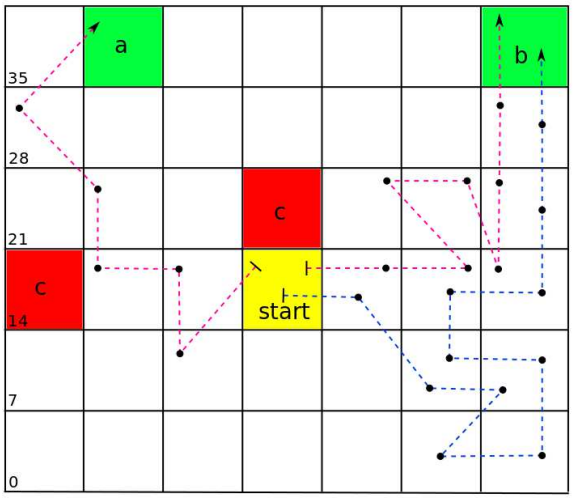


Figure 4.8: Sample trajectories under optimal controller(s) for MDP (pink) and POMDP (blue).

Sample trajectories resulting from numerical simulations for both MDP and POMDP examples are shown in Figure 4.8. As mentioned before, for MDP-World, two controllers with identical performance that reach cells a and b respectively, were found. Pink dashed lines show a sample trajectory for both such controllers. Running the gradient ascent algorithm on the POMDP-World yielded two controllers corresponding to $\varphi_2(a)$ and $\varphi_2(b)$ as well. However, the latter gave better optimal value for the objective. A sample trajectory for the POMDP using the best controller is also shown in the same figure via a blue dashed line.

4.7.3 Case Study III - Initial Feasible Controller

System Model and LTL Specification

In this case study, the models Det-World, Prob-World and POMDP-World from Case Study I are studied again under the LTL formula φ_1, which requires repeatedly reaching green cells 0 and 6 while avoiding red cell 3. However, there is a small difference in the motion model. The actions move up (U) and move down (D), are now deterministic: the robot follows the motion model from Det-World of Figure 4.4(a). The limitation of the heuristic in Algorithm 4.2 used to find the initial feasible controller can be seen when applied to varying corridor widths in the grid world models. Table 4.2 shows the cases for which the heuristic succeeds in finding an initial feasible controller.

Results

It was found that for $N = 3$, even though there exist feasible FSCs for the POMDP-World, the heuristic is unable to find them. This failure occurs because the heuristic prunes actions based only on a single, most recent observation. Effectively, the controller considers only the previous step in its execution. However, in order to localize itself to the top edge of the grid world for $N = 3$, from where it can move right or left safely, it needs access to its actions and observations from past time

Corridor Width	Det-World		Prob-World		POMDP-World	
	Feasible	Found	Feasible	Found	Feasible	Found
$N = 1$	No	No	No	No	No	No
$N = 2$	Yes	Yes	No	No	No	No
$N = 3$	Yes	Yes	Yes	Yes	Yes	No
$N \geqslant 4$	Yes	Yes	Yes	Yes	Yes	Yes

Table 4.2: Finding the Initial Feasible Controller by Algorithm 4.2. For $N = 3$, even though there exist feasible FSCs for the POMDP-World, the heuristic is unable to find them.

steps as well.

4.8 Concluding Remarks

This chapter showed how the problem of maximizing the satisfaction of an LTL formula for a POMDP can be solved for a fixed structure of the FSC. The idea was to identify recurrent sets that are φ-feasible, and then maximize the probability of absorption into some φ-feasible set. The φ-feasibility was framed as a constraint, and was finally incorporated into the objective using the log barrier function, so that gradient optimization methods could be applied. The gradient itself was shown to be computable and analytical expressions were derived for the same. A suboptimal heuristic was then proposed to find the $\varphi-$feasible recurrent sets in the first place. Several case studies that highlight the various aspects of the algorithm were shown.

Chapter 5

Reward Design for LTL Satisfaction

In the previous chapter it was shown how to maximize the LTL satisfaction probability for an FSC $\mathcal{G}$ of a given structure, $\mathcal{I}$. This was done by parametrizing the distributions ω which determine the probability of making I-states transitions and issuing actions, and κ, which chooses the initial I-state of the FSC given the product-POMDP initial distribution. The optimization was to the parameters that determine ω and κ. However, the general problem of maximizing LTL formulas over a POMDP encompasses searching over all possible FSC structures, which includes the FSC size $|G|$, the allowed transitions and actions from each I-state of the FSC, $\mathcal{I}_\omega$, and the allowed initial nodes of I-States $\mathcal{I}_\kappa$.

This chapter outlines how to simultaneously set up an *any time* optimization algorithm to search both over the quality and structure of the FSC. However, the structure of the FSC $\mathcal{I}$ will not be examined explicitly, except for the number of I-states utilized by the controller. Instead, the structure can be inferred from the positive probability I-state transitions, actions, and initial I-states, in ω and κ. In addition, the distributions $\omega(g_l, \alpha | g_k, \emptyset)$ and $\kappa(g)$ are no longer dependent on other parameters such as Φ and Θ as in the previous chapter, but are found directly from a constrained optimization problem, which if feasible, ensures that they obey the laws of probability.

The reader is reminded of the goal of the optimization in Equation (4.48) from the previous chapter: Maximize the probability of absorption into a φ-feasible recurrent set, while ensuring that in steady state, the states in the corresponding $Repeat_i^{\mathcal{P}\mathcal{M}^\varphi}$ are visited with finite frequency once absorbed. An example of robotic tasks that require repeated (but not necessarily periodic) behavior is the case of service or surveillance robots that must repeatedly visit designated items or locations that are not co-located. Another example is the use of robots on assembly lines, where a robot is required to perform one of several possible tasks that are determined on the fly.

5.1 Reward Design for LTL Satisfaction

In this chapter, a *different* optimization goal will be considered. Instead of maximizing the probability of absorption into a φ-feasible recurrent set directly, this chapter tackles a reward design problem with the following goals:

- During the transient phase of the global Markov chain execution, the global state is absorbed into a φ-feasible recurrent set *quickly*.
- During the steady state phase of the global Markov chain execution, the state visits the states in $Repeat_{\mathfrak{r}}^{\mathcal{P}\mathcal{M}^{\varphi}}$ *frequently*.

These design goals are motivated by real world engineering examples, e.g. with physical robots, energy management systems, etc., where the occurrence of "good" events in quick succession can be crucial. Some concrete examples are: (a) Finishing a set of tasks before the robot battery runs low, in which case some sensors or actuators may start to function below par; (b) A robot needs to traverse a debris field quickly to reach an emergency site for response and recovery; (c) A sentry robot that needs to surveil each of several locations indefinitely, should ideally navigate quickly to the next location in a round robin scheme.

5.1.1 Incentivizing Frequent Visits to $Repeat_{\mathfrak{r}}^{\mathcal{P}\mathcal{M}^{\varphi}}$

In classical POMDP planning problems, in which the agent collects rewards as it visits different states, the desire to quickly accumulate useful goals is incentivized by weighing the rewards collected at later time steps less. Formally, this incentivization process is encoded via a discounted reward scheme introduced in Definition 2.2.8. There exist temporal logics that allow explicit verification/design for known finite time horizon [8, 21]. But, it may be hard to gauge what horizon is feasible for a given POMDP and LTL formula *a-priori*. In such scenarios, a discounted reward scheme, which does not effect feasibility, thus offers a viable solution.

In light of the above, a goal that incentivizes frequent occurrences of "good" events will be set up as follows. Consider, a particular Rabin acceptance pair $(Repeat_{\mathfrak{r}}^{\mathcal{P}\mathcal{M}^{\varphi}}, Avoid_{\mathfrak{r}}^{\mathcal{P}\mathcal{M}^{\varphi}})$ in the product-POMDP. The aim is to visit some states in $Repeat_{\mathfrak{r}}^{\mathcal{P}\mathcal{M}^{\varphi}}$ in the product-POMDP often. For this, assign a reward scheme, called the "repeat" reward scheme, as follows.

$$r_{\mathfrak{r}}^{\beta}(s) = \begin{cases} 1 & \text{if } s \in Repeat_{\mathfrak{r}}^{\mathcal{P}\mathcal{M}^{\varphi}} \\ 0 & \text{otherwise.} \end{cases} \tag{5.1}$$

The difference from the reward scheme in Section 4.3.1 is that we are no longer looking for a particular recurrent set in the global state space. Equation (5.1) assigns rewards in the *Product-POMDP* state

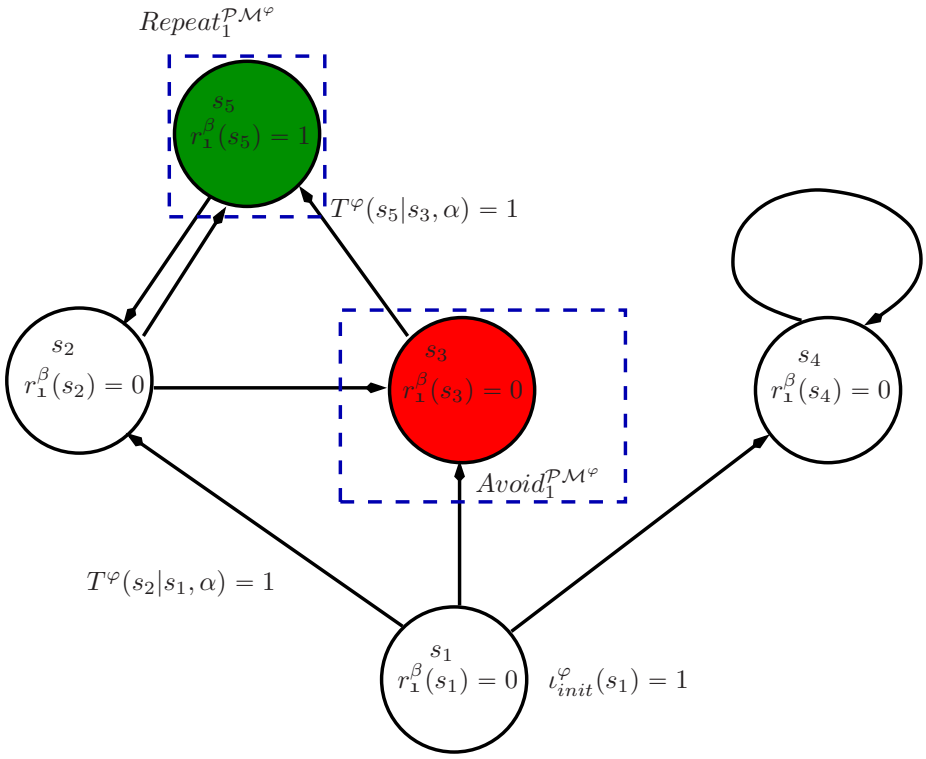


Figure 5.1: Assigning rewards for visiting $Repeat^{\mathcal{PM}^\varphi}$ frequently. The above shows the state space of product-POMDP. The edges are dependent on a particular action, $\alpha \in Act$, chosen arbitrarily here. There is only one Rabin pair $(Repeat_1^{\mathcal{PM}^\varphi}, Avoid_1^{\mathcal{PM}^\varphi})$. In order to incentivize visiting $Repeat_1^{\mathcal{PM}^\varphi}$, the state $s_5 \in \mathcal{S}$ is assigned a reward of 1, while all other states get assigned a reward of 0.

space and are not concerned with the recurrent sets generated by a controller. This reward scheme is represented in Figure 5.1.

Next, the discounted reward problem of interest is formally set up as:

$$\eta_\beta(\mathfrak{r}) = \lim_{T\to\infty} \mathbb{E}\left[\sum_{t=0}^{T} \beta^t r_\mathfrak{r}^\beta(s_t) \,|\iota_{init}^\varphi\right], \quad 0 < \beta < 1, \tag{5.2}$$

where β is again the discount factor. In Equation (5.2), while the objective incentivizes early visits to states in $Repeat_\mathfrak{r}^{\mathcal{PM}^\varphi}$ to accrue the maximum reward, it has two drawbacks:

1. The objective becomes exponentially less dependent on visitations to $Repeat_\mathfrak{r}^{\mathcal{PM}^\varphi}$ at later time steps. The rate of this decay is given by β. Thus, these visits are incentivized to be frequent during the initial several time steps.

2. Due to partial observability, the transition from transience to recurrent (steady state) phase of the global Markov chain cannot be reliably detected during a given execution. This poses the difficulty of precluding visits to $Avoid_\mathfrak{r}^{\mathcal{PM}^\varphi}$ when steady state is reached.

In order to tackle the first problem, if a stationary policy that is independent of the initial

distribution of the product-POMDP can be found, then the expected visitation frequency can be expected to remain the same for later time steps, including during steady state, when the global Markov chain evolves in a recurrent set.

A suboptimal solution to tackle the second problem is discussed in the remainder of this chapter.

5.1.2 Computing the Probability of Visiting $Avoid^{\mathcal{P}\mathcal{M}^{\varphi}}$ in Steady State

In this section a method for computing the likelihood or probability of visiting any state in $Avoid^{\mathcal{P}\mathcal{M}^{\varphi}}$ is developed. If this quantity can be computed, then the discounted reward criterion can be optimized under the constraint that this probability is zero, or extremely low. In order to compute the probability of visiting $Avoid^{\mathcal{P}\mathcal{M}^{\varphi}}$ regardless of the phase (transient or steady state) of execution of the global Markov chain, first consider a modified Product-POMDP as follows. For $\forall \alpha \in Act$ do

$$T^{\varphi}_{mod}(s_k|s_j,\alpha) = \begin{cases} 0 & \text{if } s_j \neq s_k \text{ and } s_j \in Avoid^{\mathcal{P}\mathcal{M}^{\varphi}}_{\mathfrak{r}} \\ 1 & \text{if } s_j = s_k, \text{ and } s_j \in Avoid^{\mathcal{P}\mathcal{M}^{\varphi}}_{\mathfrak{r}} \\ T^{\varphi}(s_k|s_j,\alpha) & \text{otherwise.} \end{cases} \tag{5.3}$$

This choice of transition rule has the effect of making all states in $Avoid^{\mathcal{P}\mathcal{M}^{\varphi}}_{\mathfrak{r}}$ *sinks*.

Then, assign a different, "avoid" reward scheme

$$r^{av}_{\mathfrak{r}}(s) = \begin{cases} 1 & \text{if } s \in Avoid^{\mathcal{P}\mathcal{M}^{\varphi}}_{\mathfrak{r}} \\ 0 & \text{otherwise.} \end{cases} \tag{5.4}$$

Figure 5.2 shows the modification in transition distribution and the new reward assignment for an arbitrary $\alpha \in Act$.

Finally, under a given FSC, $\mathcal{G}$, consider the expected long term average reward

$$\eta_{av}(\mathfrak{r}) = \lim_{T\to\infty} \frac{1}{T} \mathbb{E}_{mod}\left[\sum_{t=0}^{T} r^{av}_{\mathfrak{k}}(s_t) \,\middle|\, \iota^{\mathcal{P}\mathcal{M}^{\varphi}}_{init} \right] \tag{5.5}$$

where the expectation is using the global Markov chain arising from the modified transition distribution T^{φ}_{mod} as defined in Equation (5.3)

Lemma 5.1.1

$$\Pr\left[\pi \to (Avoid^{\mathcal{P}\mathcal{M}^{\varphi}}_{\mathfrak{r}} \times G) \,\middle|\, \iota^{\varphi,\mathcal{G}}_{init} \right] = \eta_{av}(\mathfrak{r}) \tag{5.6}$$

where $\pi \in Paths(\mathcal{M}^{\mathcal{P}\mathcal{M}^{\varphi},\mathcal{G}})$ *is a path in the global Markov chain, which would arise from the execution of the original* unmodified *Product-POMDP.*

Proof See Appendix B.1.

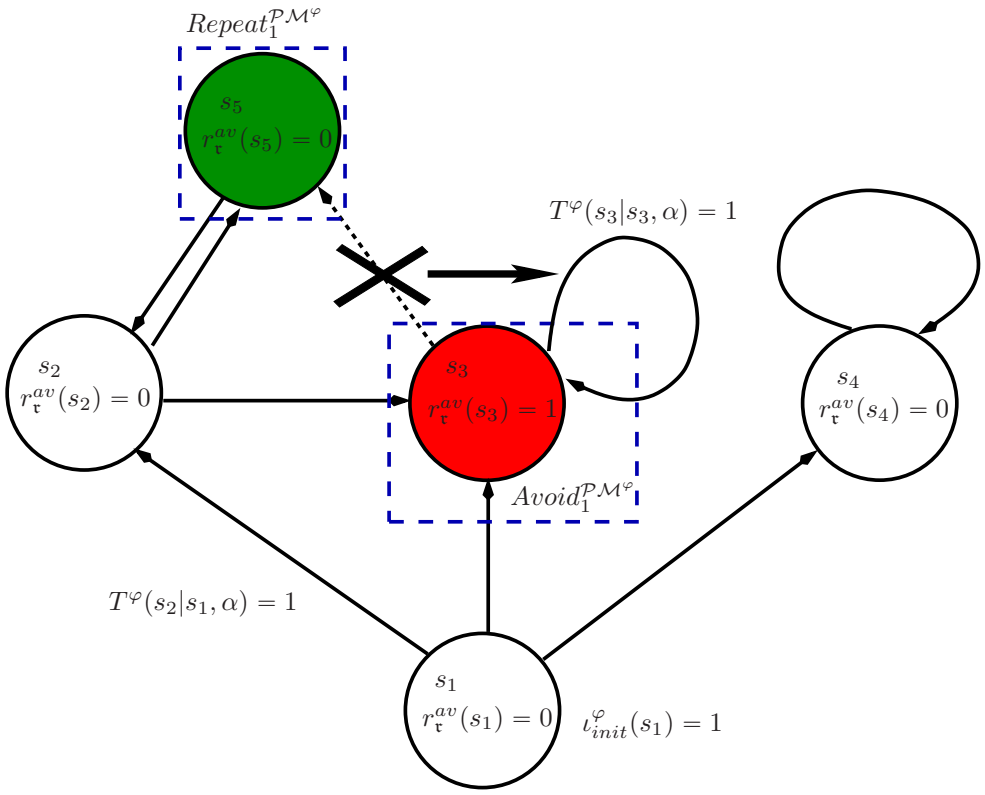


Figure 5.2: Modifying T^φ for steady state φ-feasibility. The diagram shows the state space of the Product-POMDP, $\mathcal{PM}^\varphi$. There is only one Rabin pair $(Repeat_1^{\mathcal{PM}^\varphi}, Avoid_1^{\mathcal{PM}^\varphi})$ in this depiction. For each action $\alpha \in Act$, all states in $Avoid_1^{\mathcal{PM}^\varphi}$ are made sinks or recurrent classes by themselves.

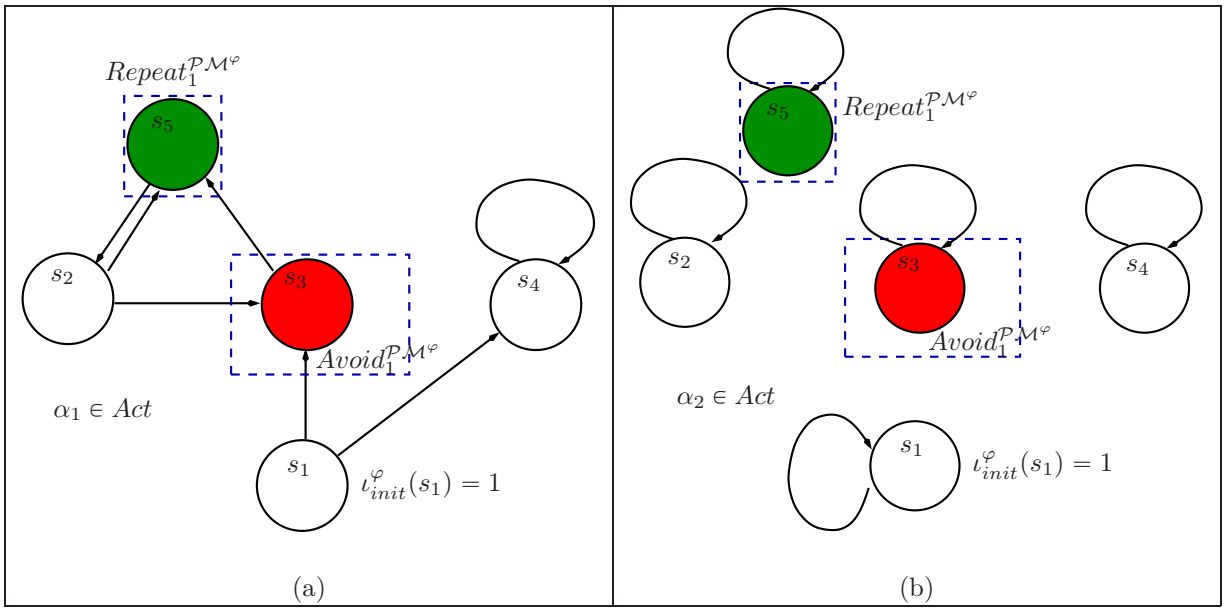


Figure 5.3: Example where visiting $Avoid^{\mathcal{PM}^\varphi}$ is required to reach $Repeat^{\mathcal{PM}^\varphi}$. In this example, consider a 5 state product-POMDP which has only two actions, α_1 and α_2. The transitions for each is shown above in (a) and (b) respectively. There is only one Rabin acceptance pair, shown by the green state which must be visited infinitely often, and the red state which cannot be visited in steady state. The initial distribution is concentrated in state s_1. A deterministic controller that issues the sequence of actions $\alpha_1\alpha_1\alpha_2\alpha_2\alpha_2\ldots$ has positive probability of satisfying the LTL specification. However, if the controller is constrained to have zero probability of visiting the red node, no controller with positive satisfaction exists.

Lemma 5.1.1 provides a simple and tractable way of computing the probability of visiting $Avoid_{\mathfrak{r}}$. Note the conditional dependence on ι_{init}^{φ} in Equation (5.5). Recall that to satisfy an LTL formula, it is only required to guarantee that the probability of visiting and avoid state is zero in *steady state*. Requiring this probability to be zero during the transient period may render many problems infeasible where admissible controllers exist. One such example is explained in Figure 5.3.

Unfortunately, for partially observable processes, all of the formulation presented so far does not provide a way to know if a *particular* path has entered steady state behavior. Recall that steady state behavior in a Markov chain corresponds to the state entering a recurrent set. At most it is possible to know the *probability* of being in steady state by taking the sum of all beliefs over recurrent states.

Next, consider that the controller has access to an oracle that can declare the end of the transient period during which visits to $Avoid_{\mathfrak{k}}^{\mathcal{PM}^\varphi}$ may be allowed. Conversely, the oracle allows knowledge of when the state enters a sub-Markov chain, in which the state is guaranteed to never visit $Avoid_{\mathfrak{r}}$. Of course, the controller doesn't actually have access to this oracle, but perhaps a product-POMDP and reward assignment can be *designed* such that the controller *implicitly incorporates* the function of the oracle.

First, a conservative way to approximate this oracle function is described, which will be shown to preclude some controllers which could be valid otherwise.

5.1.3 Partitioned FSC and Steady State Detecting Global Markov Chain

Suppose that the FSC I-states, G, are differentiated *a-priori* into two sets G^{tr} and G^{ss} such that

$$G = G^{tr} \cup G^{ss}, \quad \text{and} \quad G^{tr} \cap G^{ss} = \varnothing, \tag{5.7}$$

where, superscripts tr and ss stand for "transient" and "steady state" respectively. The idea is to use this partitioning of the FSC I-states to indicate whether the execution of the global Markov chain can ensure zero probability of any future visits to $Avoid^{\varphi}_{\mathfrak{k}}$. This is explained further as follows.

Let the global state at any time step t be given by $[s_t, g_t]$. The aim of this procedure is to create a global Markov chain using the Product-POMDP that has the following property

$$\Pr\left[[s_{t'}, g_{t'}] \in Avoid^{\mathcal{PM}^{\varphi}}_{\mathfrak{k}} \times G \,\middle|\, \exists t \leqslant t', \text{ s.t. } [s_t, g_t] \in Repeat^{\mathcal{PM}^{\varphi}}_{\mathfrak{k}} \times G^{ss}\right] = 0. \tag{5.8}$$

In other words, let the product-POMDP visit a state in $Repeat^{\mathcal{PM}^{\varphi}}_{\mathfrak{k}}$, when the FSC is executing in steady state, i.e., $g_t \in G^{ss}$. Then, it must be ensured that the probability for the product-POMDP to visit $Avoid^{\mathcal{PM}^{\varphi}}_{\mathfrak{k}}$ at anytime in the future is zero. This requirement can be achieved in three steps.

First, the FSC is constrained so that it cannot transition from a state in G^{ss} to any other state in G^{tr}. Formally, $\forall \alpha \in Act, o \in \mathcal{O}$,

$$\omega(g', \alpha | g, o) = 0, \quad g \in G^{ss}, g' \in G^{tr}. \tag{5.9}$$

This constraint ensures that the controller is forced to transition to steady state only *once* during an execution, mimicking the fact that for each infinite path in the Markov chain, the transition to a recurrent set occurs only once.

Second, the method of evaluating the transition distribution of the global Markov chain is based on the following definition.

Definition 5.1.2 (Steady State Detecting Global Markov Chain) *The* steady state detecting global Markov chain *(hereafter shortened* ssd-global Markov chain*) is given by its transition*

distribution function

$$T_{ssd}^{\mathcal{PM}^{\varphi},\mathcal{G}}\left([s',g']\,|[s,g]\right) = \begin{cases} \sum_{o\in\mathcal{O}}\sum_{\alpha\in Act} O(o|s)\omega(g',\alpha|g,o)T^{\varphi}(s'|s,\alpha) & \text{if } g\in G^{tr}, g'\in G^{ss}, \\ \sum_{o\in\mathcal{O}}\sum_{\alpha\in Act} O(o|s)\omega(g',\alpha|g,o)T^{\varphi}_{mod}(s'|s,\alpha) & \text{if } g,g'\in G^{ss} \\ 0 & \text{if } g\in G^{ss}, g'\in G^{tr}, \\ & \textit{due to Equation (5.9).} \end{cases} \tag{5.10}$$

where the use of *modified* transition function in line two from Equation (5.3) should be noted. This corresponds to the modification in which all states in $Avoid_{\mathfrak{r}}^{\mathcal{PM}^{\varphi}}$ were made sinks.

The utility of the ssd-global Markov chain is visualized in Figures 5.4 and 5.5. The two figures show how this construction allows for steady state behavior to preclude visits to $Avoid_{\mathfrak{r}}^{\mathcal{PM}^{\varphi}}$, while still allowing the execution to visit these states in the transient phase.

Third, in addition to the two reward schemes in Equations (5.1) and (5.4), assign rewards to the I-states as well. For all $g\in G$,

$$r_{\mathfrak{r}}^{G}(g) = \begin{cases} 1 & \text{if } g\in G^{ss} \\ 0 & \text{if } g\in G^{tr} \end{cases} \tag{5.11}$$

5.1.4 Posing the Problem as an Optimization Problem

Using the above various definitions of rewards from Equations (5.1), (5.4) and (5.11), for an FSC of fixed size and partitioning $G=\{G^{tr},G^{s}\}$, the following conservative optimization criterion is derived.

Conservative Optimization Criterion

$$\begin{aligned} \max_{\omega,\kappa} \quad & \eta_{\beta}(\mathfrak{r}) \\ \text{subject to} \quad & \eta_{av}^{ssd}(\mathfrak{r}) = 0 \\ & \omega(g',\alpha|g,o) = 0 \qquad g'\in G^{tr}, g\in G^{ss} \\ & \sum_{(g',\alpha)\in G\times Act}\omega(g',\alpha|g,o) = 1 \quad \forall g\in G, o\in\mathcal{O} \\ & \omega(g',\alpha|g,o) = 1 \qquad \forall g,g'\in G, o\in\mathcal{O}, \alpha\in Act \\ & \sum_{g\in G}\kappa(g) = 1. \end{aligned} \tag{5.12}$$

In the Equation (5.12), frequent visits to $Repeat_{\mathfrak{r}}^{\mathcal{PM}^{\varphi}}$ are incentivized by maximizing

$$\eta_{\beta}(\mathfrak{r}) = \lim_{T\to\infty}\mathbb{E}\left[\sum_{t=0}^{T}\beta^{t}r_{\mathfrak{r}}^{\beta}(s_t)r_{\mathfrak{r}}^{G}\,|\iota_{init}^{\varphi}\right], \quad 0<\beta<1, \tag{5.13}$$

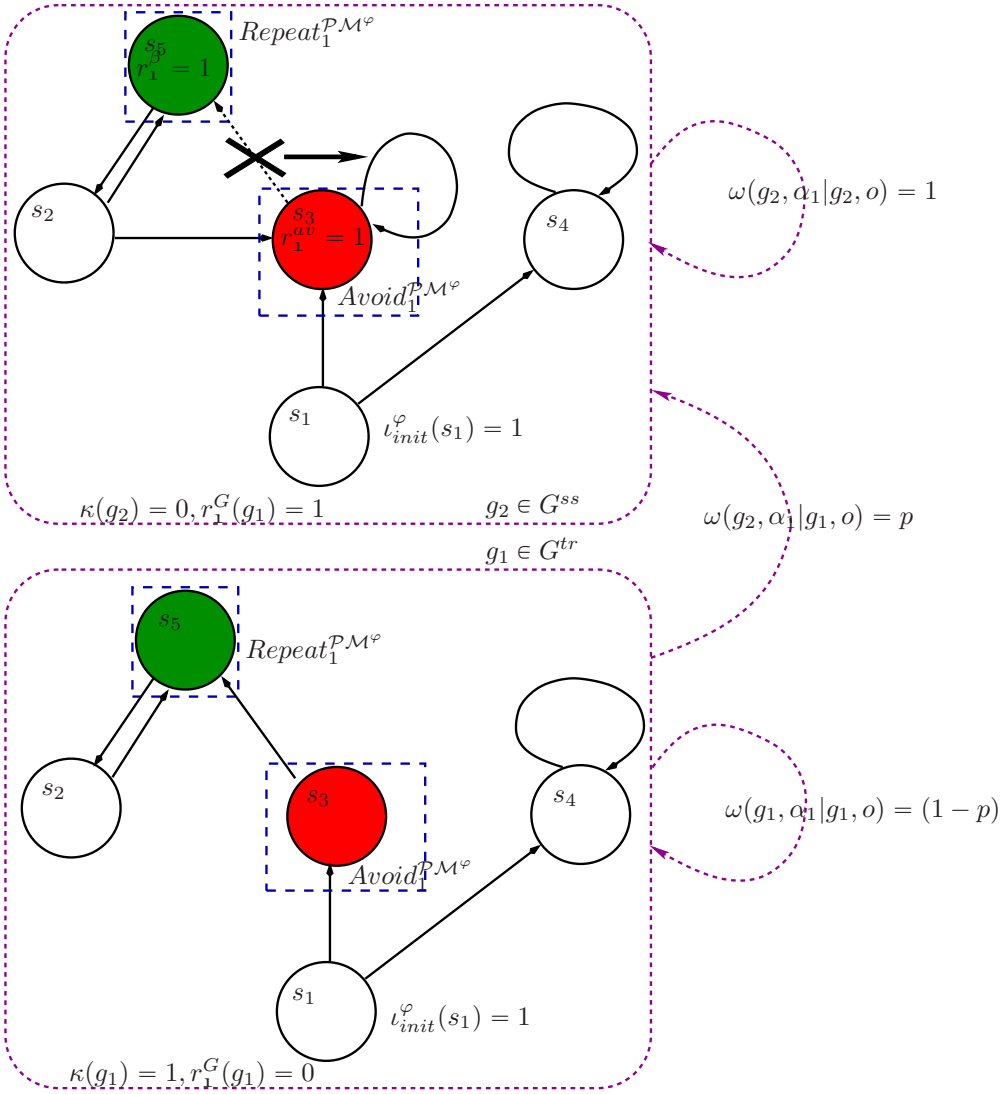


Figure 5.4: Steady state detecting global Markov chain. Consider the example from Figure 5.3. This figure shows how the ssd-global Markov chain might look if only one action α_1 is allowed in the FSC. The FSC is assumed to have two states $g_1 \in G^{tr}$ and $g_2 \in G^{ss}$ indicated by the large dotted squares. Since the global Markov chain's state space is a cartesian product of the Product-POMDP and the FSC states, each I-state is shown to contain all the Product-POMDP states. The FSC transitions probabilities are shown in dotted lines. While the black arrows denote between the Product-POMDP states show the actions that are allowed in the corresponding FSC state. The FSC starts in g_1. Clearly under this policy, the probability of reaching s_5 is positive when $0 < p < 1$, since $s_3 \in Avoid_1^{\mathcal{PM}^\varphi}$ is allowed to be visited when controller state is $g_1 \in G^{tr}$. However, even after the FSC has transitioned to the I-state $g_2 \in G^{ss}$ and the product state reaches $s_5 \in Repeat_1^{\mathcal{PM}^\varphi}$, there is still a finite probability of visiting s_3, thereby violating the constraint in Equation (5.8.)

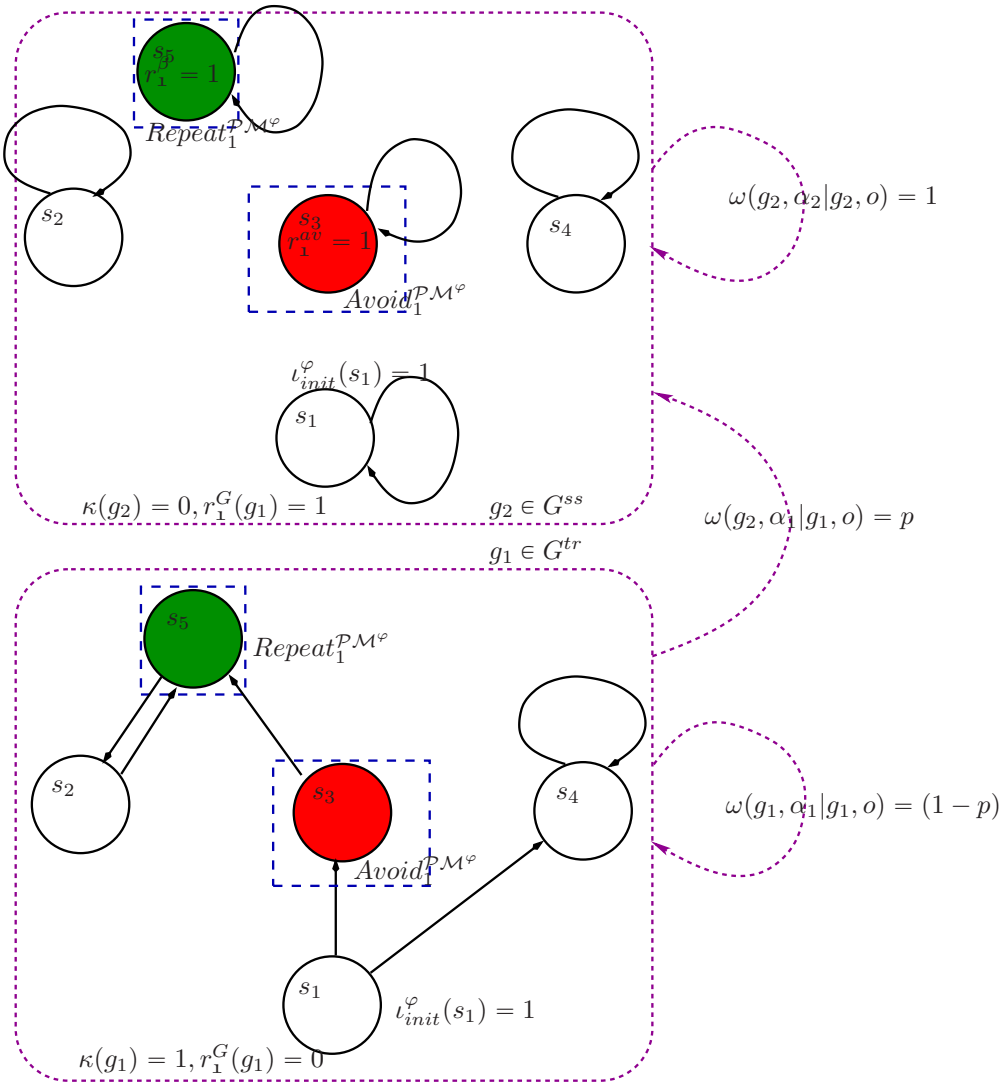


Figure 5.5: Steady state detecting global Markov chain. Consider the same example as in previous figure. The FSC again has two I-states $g_1 \in G^{tr}$ and $g_2 \in G^{ss}$. However, under this possible, only α_1 is allowed in g_1 and only $alpha_2$ is allowed in g_2. This FSC again allows visit to $s_3 \in Avoid_1^{\mathcal{PM}^\varphi}$ when in g_1. There is a finite probability of reaching $[s_5, g_2]$, with $s_5 \in Repeat_1^{\mathcal{PM}^\varphi}$ if $0 < p < 1$. Also under this policy, if the FSC I-state is g_2 and the product state reaches $s_5 \in Repeat_1^{\mathcal{PM}^\varphi}$, then the probability of visiting s_3 at any time in the future is zero, thus satisfying the constraint in Equation (5.8).

and steady state visits to $Avoid_{\mathfrak{r}}^{\mathcal{PM}^{\varphi}}$ are forbidden the first constraint

$$\eta_{av}^{ssd}(\mathfrak{r}) = \lim_{T\to\infty} \frac{1}{T} \mathbb{E}_{ssd} \left[\sum_{t=0}^{T} r(s_t) r_{\mathfrak{r}}^{G} \, | \iota_{init}^{ss} \right] = 0 \tag{5.14}$$

in Equation (5.12), where the new quantity, ι_{init}^{ss}, introduced in Equation (5.14), will be explained shortly. The other constraints relate to the I-state partitioning that has already been introduced and the enforcement of laws of probability for admissible FSCs. Note that the expectation needed to compute η_{av}^{ssd} uses the ssd-global Markov chain transition distribution from Equation (5.10), but that of η_{β} uses the unmodified Markov chain transition distribution. The product terms $r_{\mathfrak{r}}^{\beta} r_{\mathfrak{r}}^{G}$ and $r_{\mathfrak{r}}^{\beta} r_{\mathfrak{r}}^{G}$ ensure that only those visits to $Repeat_{\mathfrak{r}}^{\mathcal{PM}^{\varphi}}$ are rewarded when the controller I-state lies in G^{ss}, implying that it can now guarantee no more visits to $Avoid_{\mathfrak{r}}^{\mathcal{PM}^{\varphi}}$. This can be seen in Figures 5.4 and 5.5 as well.

Next, define ι_{init}^{ss} as a distribution over the ssd-global Markov chain states as follows.

$$\iota_{init}^{ss}([s,g]) = \begin{cases} \frac{1}{|G^{ss}||Repeat_{\mathfrak{r}}^{\mathcal{PM}^{\varphi}}|} & \text{if } s \in Repeat_{\mathfrak{r}}^{\mathcal{PM}^{\varphi}}, g \in G^{ss} \\ 0 & \text{otherwise.} \end{cases} \tag{5.15}$$

The effect of choosing this initial condition implies that in steady state,

$$\forall [s,g] \in (Repeat_{\mathfrak{r}}^{\mathcal{PM}^{\varphi}} \times G^{ss}), \quad [s,g] \nrightarrow (Avoid_{\mathfrak{r}}^{\mathcal{PM}^{\varphi}} \times G). \tag{5.16}$$

Compare this to the statement in Equation (5.8), which can be re-written as

$$\Pr\left[\pi \to [s,g] \in (Repeat_{\mathfrak{r}}^{\mathcal{PM}^{\varphi}} \times G)\right] > 0 \implies [s,g] \nrightarrow (Avoid_{\mathfrak{r}}^{\mathcal{PM}^{\varphi}} \times G). \tag{5.17}$$

This later statement requires this condition to hold for only those states in $(Repeat_{\mathfrak{r}}^{\mathcal{PM}^{\varphi}} \times G)$ under the current ω and κ. If some repeat state is not visited by the controller during steady state, then our proposed choice of ι_{init}^{ss} adds additional feasibility constraints, which may severely reduce the obtainable reward, $\eta_{\mathfrak{r}}^{\beta}$, and in the worst case, render the problem infeasible. This phenomenon can be understood from the example in Figure 5.6. This is the reason that the optimization problem stated in Equation (5.12) is called a Conservative Optimization Criterion. While being suboptimal, this criterion has some signficant advantages. As will be outlined in the next chapter, the Conservative Optimization Criterion can be framed as a policy iteration algorithm, in which each policy improvement step can be solved efficiently. Moreover the policy improvement step also yields a way to add I-states to the controller. This is useful to escape the local maxima encountered during optimization of the total reward $\eta_{\mathfrak{r}}^{\beta}$. The intuition behind the addition of FSC I-states is that they allow the generation and differentiation of many new observation and action sequences. This implies

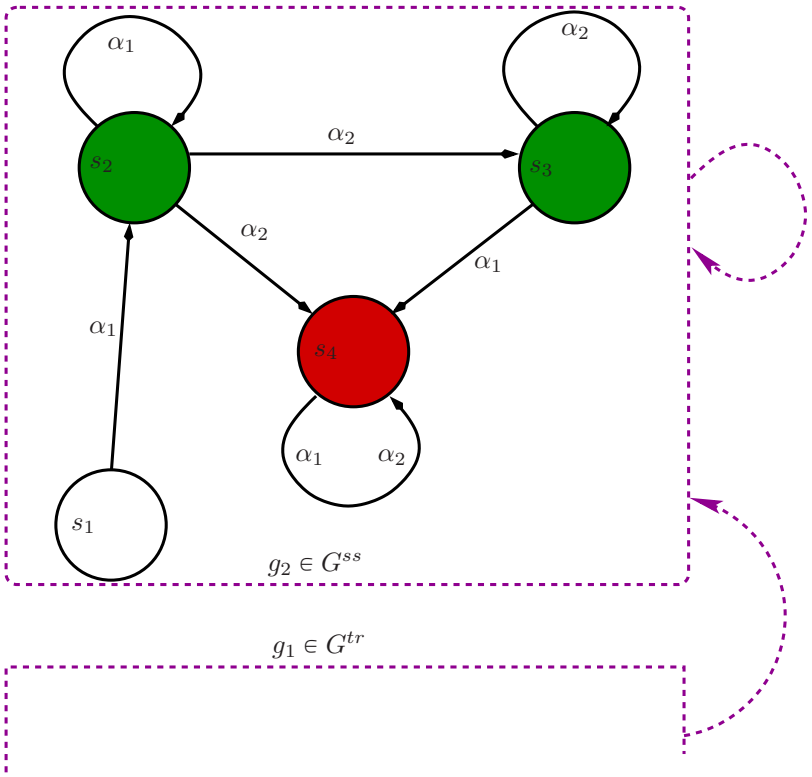


Figure 5.6: Reduced Feasibility arising from Conservative Optimization Criterion. Consider an example of a product-POMDP in which each action results in deterministic state transition as shown by the solid black arrows. $Repeat_1^{\mathcal{P}^{\mathcal{M}^\varphi}} = \{s_2, s_3\}$, and $Avoid_1^{\mathcal{P}^{\mathcal{M}^\varphi}} = \{s_4\}$. There are two I-states in the FSC, with $g_1 \in G^{tr}$, and $g_2 \in G^{ss}$. Note that $\omega(g_2, \alpha_1 | g_1, o) = 1$ disconnects s_4 and s_3 from a repeat state s_2, which is additionally guaranteed to be visited infinitely often, thus making a feasible controller. However, the condition over η_{av}^{ssd} requires that s_4 be disconnected from the repeat set s_3 as well, even though s_3 need not be visited at all. This prevents issuing action α_1 completely, and no controller can be found.

that many new paths in the global Markov chain can be explored for improving the optimization objective. The entirety of the next chapter is spent describing this methodology.

5.2 Concluding Remarks

This chapter proposed two global state space reward schemes that can be utilized to compute the two conditions necessary for LTL satisfaction – visiting particular states quickly and frequently, especially in steady state, and guaranteed transience of certain other states. A novel partitioning of the FSC internal states was then introduced in order to express these two conditions as an optimization objective and a constraint on the optimization problem respectively. In the next chapter, a novel any-time policy iteration algorithm will be proposed to find FSCs with increasing values for the objective of this optimization problem.

Chapter 6

Policy Iteration for Reward Maximization Problem

In the previous chapter, a reward maximization scheme was introduced that allowed the controller to collect a positive reward whenever states in $Repeat_{\mathfrak{r}}^{\varphi,\mathcal{G}}$ sets were visited while being constrained to not visit any state in $Avoid_{\mathfrak{r}}^{\varphi,\mathcal{G}}$ states during steady state execution. In this chapter, dynamic programming techniques will be used to solve the associated optimization problem. The methodology will allow for simultaneous search over both the structure and quality of the FSC. The chapter is organized as follows. First a well studied and crucial equation for Markov chains, namely the Poisson Equation, is studied. Next, a review of stochastic dynamic programming is provided in the context of the expected long term discounted reward. A brief overview of two methods of solving dynamic programming problems, namely value iteration and policy iteration, is also provided. The average case dynamic program is shown to be related deeply to the Poisson Equation. Then, the Constrained Optimization Criterion from the last chapter is optimized by adapting a recent advancement in policy iteration that bounds the growth of the FSC size. The policy iteration must be started with an initial φ-feasible FSC. The chapter concludes by setting up an optimization problem to search for such feasible FSCs in order to seed the policy iteration.

6.1 The Multi-chain Poisson Equation for Markov Chains

Instead of describing the Poisson Equation and related results for Markov chains over general state spaces [94, 100], the discussion in this section is restricted to time homogenous, discrete time, finite state space Markov chains. The main focus is the ssd-global Markov chain of Definition 5.1.2, which can differentiate whether states in $Avoid_{\mathfrak{r}}^{\varphi,\mathcal{G}}$ are allowed to be visited. Recall that the ssd-global Markov chain was generated by partitioning the FSC I-states into transient and steady state sets, G^{tr}, and G^{ss}. The transition probabilities $T_{ssd}^{\mathcal{PM}^{\varphi},\mathcal{G}}$ were then computed using Equation (5.10). In addition, recall the average reward function

$$r^{av}\left([s,g]\right) = r^{av}_{\mathfrak{r}}(s) r^{G}_{\mathfrak{r}}(g). \tag{6.1}$$

In keeping with the vector notation, $\vec{r}^{av}$ will denote the vectorized representation of r^{av} for some ordering of the global state space $\mathcal{S} \times G$.

Definition 6.1.1 (Poisson Equation [56]) *The Poisson Equation (P.E.) for $T^{\mathcal{PM}^{\varphi},\mathcal{G}}_{ssd}$ is*

$$\text{(a)} \quad \vec{\mathfrak{g}} = T^{\mathcal{PM}^{\varphi},\mathcal{G}}_{ssd} \vec{\mathfrak{g}} \quad \text{and (b)} \quad \vec{\mathfrak{g}} + \vec{\mathfrak{h}} - T^{\mathcal{PM}^{\varphi},\mathcal{G}}_{ssd} \vec{\mathfrak{h}} = \vec{r}^{av}. \tag{6.2}$$

where the matrix representation for $T^{\mathcal{PM}^{\varphi},\mathcal{G}}_{ssd}$, defined in Equation (5.10), has been used.

In a general treatment of the Poisson Equation for Markov chains, r^{av} can be replaced with any measurable function, $f : \mathcal{S} \times G \to \mathbb{R}$, and is called a *charge*. If Equation (6.2) holds, then the pair $(\vec{\mathfrak{g}}, \vec{\mathfrak{h}})$ is called a solution to the P.E. with charge r^{av}.

The Poisson Equation arises in many discrete and continuous time Markov processes, especially in stochastic optimal control problems. In order to gain some intuition about the Poisson Equation, it is useful to first consider the case when a Markov chain has a single recurrent class and possibly some transient states. In this case Poisson equation solves the long term average cost criterion for given initial state $\mathfrak{s}_0$, which is given by

$$\eta_{av} = \lim_{T \to \infty} \mathbb{E}\left[\frac{1}{T} \sum_{t=0}^{T} r^{av}(t) \middle| \mathfrak{s}_0 \right]. \tag{6.3}$$

for the reward $r^{av}(t)$. In fact, the value for the scalar η_{av} can be obtained by solving a slightly different version of the Poisson Equation (6.2) given by

$$\eta_{av} + \vec{\mathfrak{h}} - T^{\mathcal{PM}^{\varphi},\mathcal{G}}_{ssd} \vec{\mathfrak{h}} = \vec{r}^{av}. \tag{6.4}$$

Note that Equation (6.4) is obtained from Equation (6.2b) by replacing the vector $\vec{\mathfrak{g}}$ by the scalar η_{av}. For a finite Markov chain with a single recurrent class, this equation has a unique solution for η_{av}. Next, consider that the Markov chain under consideration has multiple recurrent classes, $R_1, R_2, \ldots R_N$ and the set of transient states T. Recall the block represention of the transition matrix as described in Proposition (4.2.2) is given by

$$T^{\mathcal{PM}^{\varphi},\mathcal{G}}_{ssd} = \left[\begin{array}{cccc|c} T_{R_1} & \mathbf{0} & \ldots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & T_{R_2} & \ldots & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \ldots & T_{R_N} & \mathbf{0} \\ \hline T_{\mathcal{T}\to R_1} & T_{\mathcal{T}\to R_2} & \ldots & T_{\mathcal{T}\to R_N} & T_{\mathcal{T}\to\mathcal{T}} \end{array}\right]. \tag{6.5}$$

For each R_i, the scalar form of the Poisson Equation (6.4) can be solved separately for the restriction of the Markov chain to $R_i \cup T$ to yield $different$ unique long term avarage rewards $\eta_{av}^{R_i}$ for initial probability distribution belonging to $R_i \cup T$. If $T = \varnothing$, then these equations can be stacked to yield the original Poisson Equation (6.2) by letting $\vec{\mathfrak{g}}(R_i) = \eta_{av}^{R_i}$ elementwise, and

$$\vec{\mathfrak{g}} = \begin{bmatrix} \vec{\mathfrak{g}}(R_1) \\ \vec{\mathfrak{g}}(R_2) \\ \vdots \\ \vec{\mathfrak{g}}(R_N) \end{bmatrix}. \tag{6.6}$$

However, when $T \neq \varnothing$ and initial state of the Markov chain lies in T, then the average reward criterion of Equation (6.3), may not have a unique solution because, the state may be absorbed in one of several recurrent classes R_i. However, a probabilistic interpretation may be taken in which the average reward accounts the probability of absorption into the different R_i in the computation of the average cost given the intial distribution $\iota_{init}(\mathfrak{s})$. The multi-chain Poisson Equation as introduced in Equation (6.2) is used when such a probabilistic interpretation is taken. Further discussion of the Poisson Equation in the context of using Dynamic Programming for solving it is provided in Section (6.3.2).

A good development on the Poisson Equation can be found in [56, 94]. Other texts that study the conditions for existence and uniqueness of solutions to the Poisson Equation can be found in [57, 101, 123, 126, 149].

In the context of this thesis, where the resulting closed loop global Markov chain has finite state space, it is well known that a solution for the P.E. always exists. This is stated formally in the following lemma.

Lemma 6.1.2 *[56] (a) For a finite state space Markov chain given by its transition matrix* $T_{ssd}^{\mathcal{PM}^{\varphi},\mathcal{G}}$*, and charge* r^{av}*, a solution pair* $(\vec{\mathfrak{g}}, \vec{\mathfrak{h}})$ *to the Poisson Equation always exists.*

(b) Moreover, $\vec{\mathfrak{g}}$ *is* unique *and is given by*

$$\vec{\mathfrak{g}} = \Pi_{ssd} \vec{r}^{av} \tag{6.7}$$

where Π_{ssd} *is the limiting matrix introduced in Definition 3.4.12.*

(c) The solution $\vec{\mathfrak{g}}$ *in Equation (6.7), when paired with the following* $\vec{\mathfrak{h}}$*, solves the P.E:*

$$\vec{\mathfrak{h}} = H \vec{r}^{av} \tag{6.8}$$

where H is called the deviation matrix *and is given by*

$$H = \underbrace{(I - T_{ssd}^{\mathcal{PM}^{\varphi},\mathcal{G}} + \Pi_{ssd})^{-1}}_{\textit{fundamental matrix, } Z}(I - \Pi_{ssd}). \tag{6.9}$$

(d) $\vec{\mathfrak{h}}$ is not unique. If $(\vec{\mathfrak{g}}, \vec{\mathfrak{h}})$ is a solution then $\forall k \geqslant 0$, $(\mathfrak{g}, \vec{\mathfrak{h}} + \Pi_{ssd}\vec{\mathfrak{h}})$ is also a solution.

The Poisson equation is important because the quantity $\mathfrak{g}$ can be used to compute the probability of visiting the set $Avoid_{\mathfrak{r}}^{\mathcal{PM}^{\varphi},\mathcal{G}}$ for the ssd-global Markov chain in the following theorem. This is crucial because it can then be used to enforce the constraint $\eta_{av}^{ssd} = 0$ in the optimization criterion of Equation (5.12).

Theorem 6.1.3 *The probability of ssd-global Markov chain of visiting $(Avoid_{\mathfrak{r}}^{\mathcal{PM}^{\varphi}} \times G^{ss})$ for an initial distribution, $\iota'_{init} \in M_{\mathcal{S}\times G}$ is given by*

$$\Pr\left[\pi \to (Avoid_{\mathfrak{r}}^{\mathcal{PM}^{\varphi}} \times G^{ss}) \,|\iota'_{init}\right] = \vec{\iota'}_{init}^{T}\vec{\mathfrak{g}}. \tag{6.10}$$

Proof Note that under $T_{ssd}^{\mathcal{PM}^{\varphi},\mathcal{G}}$, each state in $(Avoid_{\mathfrak{r}}^{\mathcal{PM}^{\varphi}} \times G^{ss})$ is a sink by construction and therefore recurrent. Applying Lemma 5.1.1 gives

$$\begin{aligned}
\Pr\left[\pi \to (Avoid_{\mathfrak{r}}^{\mathcal{PM}^{\varphi}} \times G^{ss}) \,|\iota'_{init}\right] &= \lim_{T\to\infty} \tfrac{1}{T}\mathbb{E}\left[\sum_{t=0}^{T} r^{av}([s_t, g_t]) \,|\iota'_{init}\right] \\
&= \vec{\iota'}_{init}^{T}\ \Pi_{ssd}\ \vec{\mathbb{1}}^{\mathcal{S}\times G}_{(Avoid_{\mathfrak{r}}^{\mathcal{PM}^{\varphi}}\times G)} \\
&= \vec{\iota'}_{init}^{T}\ \Pi_{ssd}\ \vec{r}^{av} \\
&= \vec{\iota'}_{init}^{T}\ \vec{\mathfrak{g}},
\end{aligned} \tag{6.11}$$

where line 1 implies line 2 due to Equation (4.44), and line 3 follows from the fact that $\vec{r}^{av}$ can be re-written as an indicator vector $\vec{r}^{av} = \vec{\mathbb{1}}^{\mathcal{S}\times G}_{(Avoid_{\mathfrak{r}}^{\mathcal{PM}^{\varphi}}\times G^{ss})}$.

Theorem 6.1.3 will be used later in this chapter to enforce the constraint, $\eta_{av}^{ssd}(\mathfrak{r}) = 0$, during the optimization procedure for the conservative optimization criterion of Equation (5.12).

6.2 Dynamic Programming Basics

This section briefly reviews stochastic dynamic programming in discrete time domains. Dynamic programming is a specific methodology which computes a policy for a sequential decision process to optimize reward functions of certain types. The earliest formalization of the such problems is credited to Bellman [11, 12]. Several texts provide a general development of the topic [15, 16], and provide many key results for the stochastic domain.

Stochastic dynamic programming arises for dynamical systems operating in a stochastic environment whose evolution can be described by an equation of the form

$$x_{t+1} = f(x_t, \alpha_t, w_t), \tag{6.12}$$

where x_t is the state of the system at time step t, α_t are exogenous inputs that can be applied by an agent and w_t is a disturbance from some probability space. The discussion in this section is restricted to the case of finite state and action spaces, and also to a finite probability space for the disturbance. It is required that disturbance w_t have a conditional distribution of the form $p(w_t|x_t, \alpha_t)$. Also of concern is the reward obtained by the agent, given by a function $r(x_t, \alpha_t, w_t)$. Next, consider a *policy* $\mu = (\mu_0, \mu_1, \dots)$ where at step t, an action α_t is chosen according to μ_t. For stochastic systems, the policy μ_t may require the entire execution history to successfully pick the action, i.e.,

$$\alpha_t = \mu_t(x_0, \alpha_0, \dots, \alpha_{t-1}, x_t). \tag{6.13}$$

Restricting the discussion to the discounted long term reward and a known initial state of the system, the objective of the agent is to maximize the following expected long term discounted reward

$$\eta^{\beta}(x_0) = \sup_{\mu} \mathbb{E} \sum_{t=0}^{\infty} \beta^t r(x_t, \mu_t, w_t) \quad 0 \leqslant \beta < 1, \tag{6.14}$$

under the constraint that x_t evolves using Equation (6.12). For this particular choice of objective it is well known that a stationary Markov policy is sufficient. Formally, this means that $\alpha_t = \mu_t(x_t) = \mu(x_t)$.

The dynamic programming algorithm for the preceding algorithm is given by the iteration

$$\begin{aligned} V_0^{\beta}(x) &= 0 \\ V_{k+1}^{\beta}(x) &= \sup_{\alpha} \mathbb{E}_w \left[r(x, \alpha, w) + \beta V_k^{\beta} \left[f(x, \alpha, w) \right] \right]. \end{aligned} \tag{6.15}$$

For a known initial starting state x_0, the optimal value $\eta^{\beta *}(x_0)$ is given by the limit

$$\eta^{\beta *}(x_0) = \lim_{k \to \infty} V_k^{\beta}(x_0). \tag{6.16}$$

Since it is known that a stationary optimal policy exists in the case of infinite horizon discounted reward over a finite system model, it follows that this policy satisfies the *Bellman Optimality Equation* given by

$$V^{\beta *}(x) = \sup_{\alpha} \mathbb{E}_w \left[r(x, \alpha, w) + \beta V^{\beta *} \left[f(x, \alpha, w) \right] \right] \quad \forall x, \alpha, w. \tag{6.17}$$

6.2.1 Dynamic Programming Variants

Equations (6.15) and (6.16) are leveraged to perform dynamic programming in several popular ways in the literature. Restricting the scope of the discussion to a Markov decision process in which the

system evolution is given by a conditional probability distribution $x_{k+1} \sim T(x_{k+1}|x_k, \alpha)$, and the reward is given by $r(x, \alpha)$, the expectation in Equation (6.15) can be explicitly computed as

$$\begin{aligned} V_0(x) &= 0 \\ V_{k+1}(x) &= \sup_\alpha \sum_{x'} T(x'|x,\alpha)\left[r(x,\alpha) + \beta V_k^\beta(x')\right]. \end{aligned} \tag{6.18}$$

This iterative method constitutes what is known as a *value iteration* methodology for dynamic programming [12, 13]. At each iteration step k, $V_k^\beta(x)$ is called the *value function* of the state x. It denotes the expected long term discounted reward the agent would collect if the initial state $x_{t=0} = x$. When the iteration has converged, the policy is computed using

$$\mu(x) = \operatorname*{argmax}_\alpha \sum_{x'} T(x'|x,\alpha)\left[r(x,\alpha) + \beta V_k^\beta(x')\right]. \tag{6.19}$$

It is also well known that for any given policy μ, the value function $V_\mu^\beta(x)$ satisfies the following system of equations

$$V_\mu^\beta(x) = \sum_{x'} T(x'|x,\alpha)\left[r(x,\alpha_k,w_k) + \beta V_k^\beta(x')\right], \tag{6.20}$$

called the *Bellman Equation*. While it is possible to solve the above equation using exact methods, in most cases the Bellman equation is solved by iteration: the r.h.s of Equation (6.20) is repeatedly applied on successive values V^β until convergence. The Bellman equation is utilized in another variant of dynamic programming, namely *policy iteration*, or Howard's method [62], outlined in Algorithm 6.1.

Algorithm 6.1 Policy Iteration for Markov Decision Process

1: $iter \leftarrow 0$
2: Choose an initial policy μ_{iter}.
3: $V_{iter}^\beta(x) \leftarrow 0 \ \forall x$
4: **repeat**
5: $\quad iter \leftarrow iter + 1$
6: $\quad$ **Policy Improvement:** Improve the policy

$$\mu_{iter}(x) = \operatorname*{argmax}_\alpha \sum_{x'} T(x'|x,\alpha)\left[r(x,\alpha) + \beta V_k^\beta(x')\right] \tag{6.21}$$

7: $\quad$ **Policy Evaluation:** Solve the Bellman Equation (6.20) to get $V_{iter}^\beta(x) \ \forall x$.
8: **until** $V_{iter}^\beta(x) - V_{iter-1}^\beta(x) \leqslant \varepsilon^\beta \ \forall x$

So far, the discussion of stochastic dynamic programming in this section has only focused on the discounted reward criterion. However, the Bellman Optimality equation, the Bellman Equation, and value and policy iteration techniques can be derived for the expected long term average reward criterion (Definition 2.2.9) as well. However, in the general setting of an arbitrary reward function and infinite state space, the existence of an optimal solution for the average case is not guaranteed

[81, 108]. However, for the set of problems of interest in this thesis, the global Markov chain is a discrete time system that evolves over finite state space, in which case the average reward does have an optimum. Additionally, as will be seen in Section 6.5, the optimal solution for the average case is not required for the algorithm proposed herein. Only the *evaluation* of the average reward value function under a given FSC is required to guarantee LTL satisfaction. Therefore, the Bellman Equation for the average reward case is sufficient for this work. In the succeeding section, the relevant dynamic programming equations for both discounted and average rewards are summarized for the specific case of POMDPs controlled by FSCs.

6.3 Summary of Dynamic Programming Facts

For the case of POMDPs controlled by FSCs, it is useful to think of the dynamic program in the global state space $\mathcal{S} \times G$. The value function is in fact defined over this global state space. In addition, the policy iteration techniques also need to be carried out in the global state space.

6.3.1 Value Function of Discounted Reward Criterion

For a given FSC $\mathcal{G}$, and the unmodified product-POMDP, the value function V^β is the expected discounted sum of rewards under $\mathcal{G}$, and can be computed by solving a set of linear equations:

$$V^\beta\left([s_i, g_k]\right) = r^\beta\left([s_i, g_i]\right) + \beta \sum_{\substack{o\in\mathcal{O},\alpha\in Act \\ g_k\in G, s_j\in\mathcal{S}}} O(o|s_i)\omega(g_l, \alpha|g_k, o)T^\varphi(s_j|s_i, \alpha)V^\beta\left([s_j, g_l]\right). \tag{6.22}$$

For the global Markov chain, the above can be written in vector notation as follows

$$\vec{V}^\beta = \vec{r}^\beta + \beta T^{\mathcal{PM}^\varphi}\ \vec{V}^\beta. \tag{6.23}$$

The above system of equations is also called the Bellman Equation for the discounted reward criterion. Sometimes it is useful to look at the value function of the POMDP states for a given I-state g, of the FSC. Note the use of the vector notation

$$\vec{V}_g^\beta = \begin{bmatrix} V^\beta\left([s_1, g]\right) \\ V^\beta\left([s_2, g]\right) \\ \vdots \\ V^\beta\left([s_{|\mathcal{S}|}, g]\right) \end{bmatrix}. \tag{6.24}$$

Given a distribution or belief, $\vec{b}$, over the the product states, a particular *I-state's* value at the

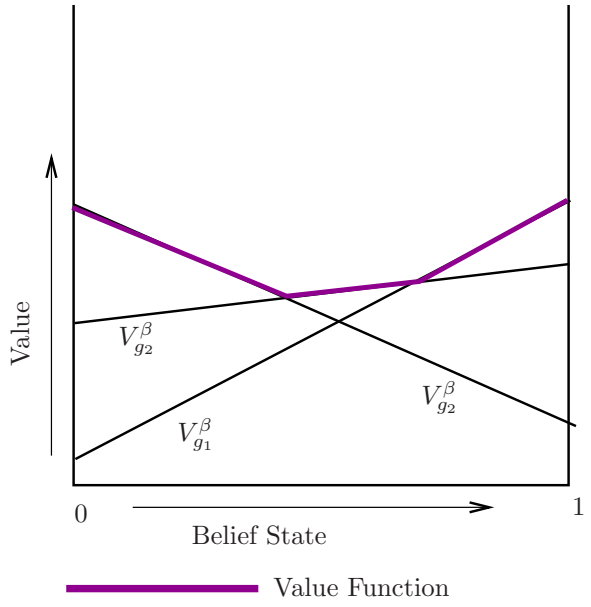


Figure 6.1: Value Function for a two state POMDP. The value of each I-state is a linear function of the belief state. The value function of the FSC is given by the upper surface (pointwise maximum) of each I-state's value.

belief is the expectation

$$V_g^{\beta}(b) = \vec{b}^T \vec{V}_g^{\beta} \quad . \tag{6.25}$$

If ι_{init}^{φ} is the initial distribution of the product-POMDP then, the best FSC I-state can be selected as

$$\kappa(g|\iota_{init}^{\varphi}) = \begin{cases} 1 & \text{if } g = \underset{g'}{\text{argmax}} \quad V_{g'}^{\beta}(\iota_{init}^{\varphi}) \\ 0 & \text{otherwise.} \end{cases} \tag{6.26}$$

In other words, the FSC is started in the I-state with maximum expected value for the belief.

Definition 6.3.1 (Value Function) *The* value function *gives the value at any belief b using the following*

$$V^{\beta}(b) = \max_{g \in G} V_g^{\beta}(b). \tag{6.27}$$

Clearly, Equation (6.25) shows that the value of a particular I-state is a linear function of the belief state. The value function itself is piece wise linear by taking the pointwise maximum of all the I-state values at each belief state. For a POMDP with only two states, the belief state can be represented by the belief in any one of the states and can be represented along a single axis. The value function for such a system can be seen in Figure 6.1, which will be useful to consider under policy iteration in forthcoming sections.

Computational Complexity and Efficient Approximation

Solving a system of linear equations by direct methods is $\mathbf{O}(n^3)$ where n is the number of equations. Equation (6.23) represents $|\mathcal{S}||G|$ equations. However, a basic Richardson iteration can be applied. One starts with an arbitrary value of V^β, typically 0, and repeatedly applies the r.h.s of the above equation to get better approximations. That is,

$$\begin{aligned} \vec{V}^{\beta,(0)} &= 0 \\ \vec{V}^{\beta,(t+1)} &= \vec{r}^{\beta} + \beta T^{\mathcal{PM}^{\varphi}} \vec{V}^{\beta,(t)} \\ &\qquad \text{until } ||\vec{V}^{\beta,(t+1)} - \vec{V}^{\beta,(t)}||_\infty < \varepsilon_\beta \end{aligned} \tag{6.28}$$

The Richardson iteration for this linear system of equations is known to converge. That is, for any $\varepsilon > 0$, $\exists p = p(\varepsilon)$ such that

$$||V^\beta - V^{\beta,(p')}|| \leqslant \varepsilon, \quad \forall p' \geqslant p \tag{6.29}$$

Several other iteration schemes also exist. A summary of these and criteria for choosing between them can be found in [128].

During each iteration, the maximum number of operations required are $\mathbf{O}(|\mathcal{S}|^2|G|^2)$, however if the ssd-global Markov chain can be represented as a sparse matrix, then the complexity is linear.

6.3.2 Value Function of Average Reward Criterion:

For a given FSC $\mathcal{G}$, the value function V^β is the expected discounted sum of rewards under $\mathcal{G}$, and can be computed by solving a set of linear equations:

$$V^{av}([s_i, g_k]) = -\rho^{av}([s_i, g_i]) + r^{av}([s_i, g_i]) + \sum_{\substack{o\in\mathcal{O},\alpha\in Act \\ g_k\in G, s_j\in\mathcal{S}}} O(o|s_i)\omega(g_l, \alpha|g_k, o)T(s_j|s_i, \alpha)V^{av}([s_j, g_l]). \tag{6.30}$$

Writing the above equation in vector notation for the ssd-global Markov chain gives

$$\vec{V}^{av} = (-\vec{\rho}^{av} + \vec{r}^{av}) + T_{ssd}^{\mathcal{PM}^{\varphi}} \vec{V}^{av}. \tag{6.31}$$

The above system of equations constitutes the Bellman Equation for the average reward criterion. Note that this is the same as the second part of the Poisson Equation (6.2(b)), by substituting $\vec{\mathfrak{g}} = \vec{\rho}^{av}$.

Computational complexity

Since the value function of the average reward criterion is identical to the Poisson Equation, the following considers the complexity of solving the Poisson Equation. Again, the exact methods of solving the linear system of equations is cubic is number of equations, which is $2|\mathcal{S}||G|$ in Equation (6.2) with as many number of variables, which comprise of both $\vec{V}^{av}$ and $\vec{\mathfrak{g}}$. Unfortunately, arbitrary initialization may not guarantee proper convergence of the variables when using typical Richardson iteration schemes. However, $\vec{\mathfrak{g}} = \Pi_{ssd}\vec{r}^{av}$ is unique for the Poisson equation, and can be pre-computed independently of V^{av}. The pre-computation of $\vec{\mathfrak{g}}$ allows for solving a linear system of equations in the unknown variables V^{av} using Equation (6.2b) exclusively. The term $\vec{\mathfrak{g}} = \Pi_{ssd}\vec{r}^{av}$ can be computed in multiple ways. First is the iterative method that leverages the fact that Π_{ssd}, whose computation involves the Cesaro sum as defined in Equation 3.25, is post multiplied by the vector $\vec{r}^{av}$. Therefore the trick employed in Section 4.2.4.1 can be applied until $\vec{\mathfrak{g}}$ converges to within a tolerance $\varepsilon_{\Pi_{ssd}} > 0$. However, limiting matrix computation for finite state space Markov chains can also be carried out by first partitioning the global state space into recurrent classes and transient states and computing the stationary distribution for each class separately [71] using numerical methods or eigen analysis. However, as will be shown later in this chapter, direct computation of the full $\vec{\mathfrak{g}}$ and $\vec{V}^{av}$ vectors will not be required frequently in the algorithm proposed in this chapter. The Poisson equation will be directly input into the optimization software as constraints in order to compute the values for the unknown vectors $\vec{\mathfrak{g}}$ and $\vec{V}^{av}$.

6.3.3 Bellman Optimality Equation / DP Backup Equation - Discounted Case

When the discounted case does not have constraints other than probability constraints on ω and κ, then at optimality the discounted value function satisfies the **Bellman Optimality Equation**, which is also known as the **DP Backup Equation**:

$$V^{\beta}(b) = \max_{\alpha \in Act} r^{\beta}(b) + \beta \sum_{o \in \mathcal{O}} \Pr(o|b) V^{\beta}(b_o^{\alpha}) \tag{6.32}$$

where

$$\Pr(o|b) = \sum_{s \in \mathcal{S}} O(o|s)b(s), \tag{6.33}$$

$$b_o^{\alpha}(s') = \sum_{s} T(s'|s,\alpha) \frac{O(o|s)b(s)}{\sum_{o' \in \mathcal{O}} O(o'|s)b(s)} \tag{6.34}$$

and $V^{\beta}(b_o^{\alpha})$ is computed using Equations (6.27) and (6.25). The r.h.s. of the DP Backup can be applied to any value function. The effect is an improvement (if possible) at every belief state. This can be seen in Figure 6.2. However, DP backup is difficult to use directly as it must be computed

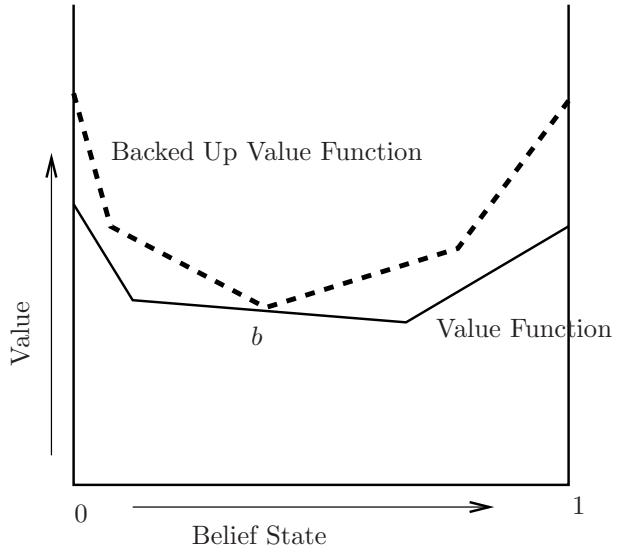


Figure 6.2: Effect of DP Backup Equation. The solid line shows a (piece-wise linear) value function V. The effect of applying the DP Backup Equation results in pointwise improvement of the value function (dashed line). However not all belief state may admit improvement as they could already be optimal. b is such a point and is called a *tangent belief state*.

at each belief state in the belief space, which is uncountably infinite. The stochastic bounded policy iteration algorithm, described in the remainder of this chapter, circumvents this by using a two pronged approach: (a) setup an efficient optimization problem to find the best ω for an FSC of a given size $|G|$; and (b) add a small, bounded number of I-states to the FSC to escape the local maxima as they are encountered.

6.4 Policy Iteration for FSCs

Policy iteration incrementally improves a controller by alternating between two steps: Policy Evaluation and Policy Improvement, until convergence to an optimal policy. For the discounted reward criterion, policy evaluation amounts to solving Equation (6.23). During policy improvement, a dynamic programming update using the DP Backup Equation is used. This results in the addition, merging, and pruning of I-states of the FSC.

In [53], the authors proposed a policy iteration algorithm to find deterministic controllers in which the I-state transition and choice of actions are unique. In each improvement step some I-states that are dominated by other I-states are pruned away, while other I-states can be added. While their work is guaranteed to converge to an ϵ-optimal controller, up to $|Act||G|^{|\mathcal{O}|}$ I-states can be added in each improvement step under the proposed algorithm. These additions can quickly increase the controller to an intractable size. In [118] a methodology called the Bounded Policy Iteration is proposed, in which the FSC is allowed to be stochastic. The next section outlines this methodology

before showing how it can be adapted for solving the Conservative Optimization Criterion given by Equation (5.12).

6.4.1 Bounded Stochastic Policy Iteration

Of concern is the problem of maximizing the expected long term discounted reward criterion over a general POMDP. The state transition probabilities are given by $T(s'|s,\alpha)$, and observation probabilities by $O(o|s)$. Most of this section follows from [118] and [54]. These authors showed that:

1. Allowing stochastic I-state transitions and action selection (i.e., FSC I-state transitions and actions sampled from distributions) enables improvement of the policy without having to add more I-states.

2. If the policy cannot be improved, then the algorithm has reached a local maximum. Specifically, there are some belief states at which no choice of ω for the current size of the FSC allows the value function to be improved. In such a case, a small number of I-states can be added that improve the policy at precisely those belief states, thus escaping the local maximum.

Both of these steps together constitute the **Policy Improvement** step of the policy iteration Algorithm 6.1.

Definition 6.4.1 (Tangent Belief State) *A belief state b is called a* tangent belief state*, if $V^{\beta}(b)$ touches the DP Backup of $V^{\beta}(b)$ from below. Since $V^{\beta}(b)$ must equal V_g^{β} for some g, we also say that the I-state g is tangent to the backed up value function V^{β} at b. Tangency can be seen in Figure 6.2.*

Equipped with this definition, the two steps involved in policy improvement can be carried out as follows.

Improving I-States by Solving a Linear Program

And I-state g is said to be *improved* if the tunable parameters associated to that state can be adjusted so that $\vec{V}_g^{\beta}$ is increased. This step tries to improve each I-state in a round robin fashion by keeping the other I-states the same. The improvement is posed as a linear program (LP) as follows:
I-state Improvement LP: For the I-state g, the following LP is constructed over the unknowns

ϵ, $\omega(g', \alpha|g, o)$, $\forall g', \alpha, o$.

$$\max_{\epsilon, \omega(g', \alpha|g, o)} \quad \epsilon$$

subject to

Improvement constraints:

$$V^{\beta}([s, g]) + \epsilon \leqslant r^{\beta}(s) + \beta \sum_{s', g', \alpha, o} O(o|s)\omega(g', \alpha|g, o)T(s'|s, \alpha)V^{\beta}([s', g']) \quad \forall s \tag{6.35}$$

Probability constraints:

$$\begin{aligned} \sum_{g', \alpha} \omega(g', \alpha|g, o) &= 1 \quad \forall o \\ \omega(g', \alpha|g, o) &\geqslant 0 \quad \forall g', \alpha, o \end{aligned}$$

The linear program searches for ω values that improve the I-state value vector $\vec{V}_g^{\beta}$ by maximizing the parameters ϵ. If an improvement is found, i.e., $\epsilon > 0$, the parameters of the I-state are updated by the corresponding maximizing ω. The value vector $\vec{V}_g^{\beta}$ may also be updated before proceeding to the next I-state in a round robin fashion.

In [118], the authors show the following interpretation of this optimization: it implicitly considers the value vectors V_g^{β} of the backed up value. A positive ϵ implies that the LP found a convex combination of the value vectors of the backed up function that dominates the current value of the I-state at every belief state. This is explained further in Figure 6.3, which is adapted from [54]. A key point is that the new value vector of the improved I-state is parallel to its current value, and the improved value becomes *tangent* to the backed up value function.

Escaping Local Maxima by Adding I-States

Eventually no I-state can be improved with further iterations, i.e., $\forall g \in G$, the corresponding LP yields an optimal value of $\epsilon = 0$. This is shown in Figure 6.4.

Theorem 6.4.2 *[118] Policy Iteration has reached a local maximum if and only if V_g is tangent to the backed up value function for all $g \in G$.*

In order to escape local maxima, the controller can add more I-states to its structure. Here the tangency criterion becomes useful. First note that the dual variables corresponding to the Improvement Constraints in the LP provides the tangent belief state(s) when $\epsilon = 0$. In some cases, a value vector may be tangent to the backed up value function not just at a single point, but along a line segment. Regardless, at a local maximum, each of the $|G|$ linear programs yield some tangent belief states. Most implementations of LP solvers solve the dual variables simultaneously and so these tangent beliefs are readily available as a by-product of the optimization process introduced above. Algorithm 6.2 shows how to use the tangent beliefs to escape the local maximum.

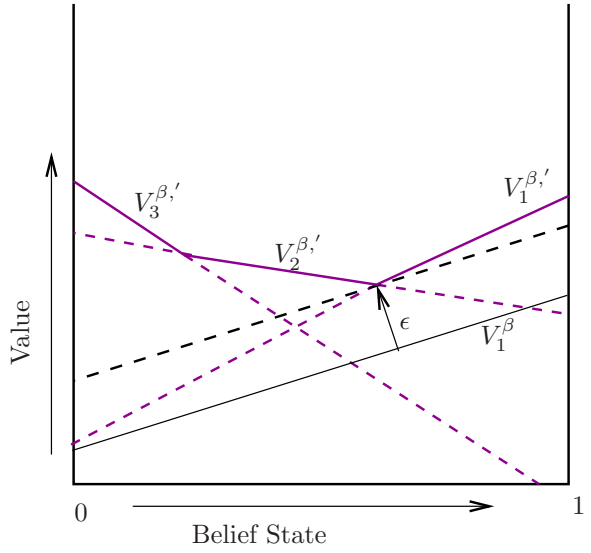


Figure 6.3: Graphical depiction of the effect of the I-state Improvement LP. This figure shows how the I-state improvement LP works. Let the LP be solved for the I-state whose value vector is V_1^{β}. The solid purple line shows the *backed up* value function. Current value function is not shown here. The backed up value vectors $V_1^{\beta,'}$ and $V_2^{\beta,'}$ are such that their convex combination (black dashed line) dominates the value vector V_1^{β} by $\epsilon > 0$. The parameters of the I-state g_1 are therefore replaced by corresponding maximizing parameters so that its value moves upwards by ϵ. Note that the improved value vector given by $V_1^{\beta} + \epsilon$ is tangent to the backed up value function.

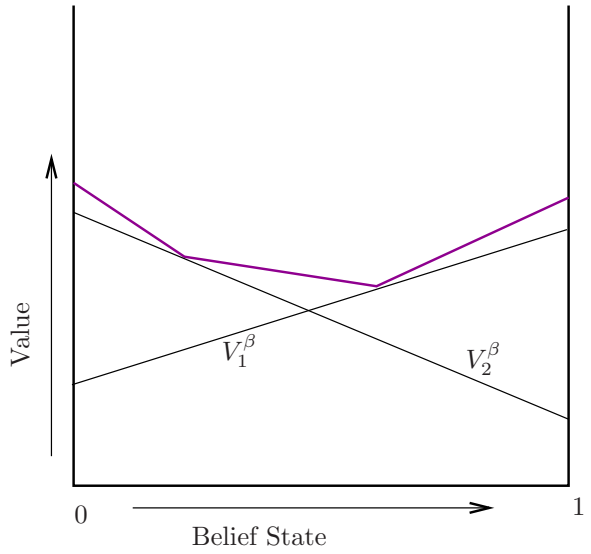


Figure 6.4: Policy Iteration Local Maximum. All current value vectors (solid lines) are tangent to the backed up value function (solid magenta). No improvement of any I-state is possible.

Algorithm 6.2 Bounded PI: Adding I-States to Escape Local Maxima

Input: Set B of tangent beliefs from policy improvement LPs for each I-state, N_{new} the maximum number of I-states to add.

$N_{added} \leftarrow 0$.
repeat
  Pick $b \in B$, $B = B \backslash \{b\}$.
  $Fwd = \varnothing$
  for all $(\alpha, o) \in (Act \times \mathcal{O})$ **do**
    if $Pr(o|b) = \sum_{s \in \mathcal{S}} b(s) O(o|s) > 0$ **then**
      Look ahead one step to compute forwarded beliefs

$$b_{o,\alpha}(s') = \sum_{s} T(s'|s,\alpha) \frac{O(o|s)b(s)}{\sum_{o' \in \mathcal{O}} O(o'|s)b(s)}. \tag{6.36}$$

      $Fwd \leftarrow Fwd \cup \{b_{o,\alpha}\}$
    end if
  end for
  for all $b_{fwd} \in Fwd$ **do**
    Apply the r.h.s. of DP Backup Equation to b_{fwd}

$$V^{\beta,backedup}(b_{fwd}) = \max_{\alpha \in Act} \left\{ r^{\beta}(b_{fwd}) + \beta \sum_{o \in \mathcal{O}} \Pr(o|b_{fwd}) \left(\max_{g \in G} b_{fwd}^{o,\alpha}(s) V_g^{\beta}(s) \right) \right\} \tag{6.37}$$

    where, $b_{fwd}^{o,\alpha}$ is computed for reach product state $s' \in \mathcal{S}$ as follows

$$b_{fwd}^{o,\alpha}(s') = \sum_{s} T(s'|s,\alpha) \frac{O(o|s)b_{fwd}(s)}{\sum_{o' \in \mathcal{O}} O(o'|s)b_{fwd}(s)}. \tag{6.38}$$

    Note the maximizing action α^* and I-state g^*.
    if $V^{\beta,backedup}(b_{fwd}) > V^{\beta}(b_{fwd})$ **then**
      Add new deterministic I-state g_{new} such that $\omega(g_{new}|g^*, \alpha^*, o) = 1 \ \forall o \in \mathcal{O}$.
      $N_{added} \leftarrow N_{added} + 1$
    end if
    if $N_{added} \geqslant N_{new}$ **then**
      return
    end if
  end for
until $B = \varnothing$.

The algorithm can be understood as follows. The tangent beliefs are those at which the DP backup results in no improvement of the value function beyond the current value. However, instead of improving the value at the tangent belief, the algorithm tries to improve the value of some belief that can be reached from the tangent belief in one step. These forwarded beliefs are computed in Steps 4-10 of Algorithm 6.2. Next, an attempt is made to improve these forwarded beliefs by DP backup (Step 12). If some action α^* and successor I-state g^* can in fact improve the value, then a new I-state is added which deterministically leads to this action and successor I-state (Steps 13-14). Note that at the end of the algorithm, the newly added I-states, g_{new} have no incoming edges, i.e., no pre-existing I-states transition to g_{new}. However, when the other I-states are improved in subsequent policy improvement steps, they generate transitions to any g_{new} added. This new I-state is then improves the value of the original tangent belief.

6.5 Applying Bounded Policy Iteration to LTL Reward Maximization

This section, shows how the bounded policy iteration methodology proposed in the previous section can be extended to the case of the Conservative Optimization Criterion, which is repeated here for convenience:

$$\begin{array}{lll} \max\limits_{\omega,\kappa} & \eta_\beta(\mathfrak{r}) & \\ \text{subject to} & \eta_{av}^{ssd}(\mathfrak{r}) = 0 & \\ & \omega(g',\alpha|g,o) = 0 & g' \in G^{tr}, g \in G^{ss} \\ & \sum\limits_{(g',\alpha)\in G\times Act} \omega(g',\alpha|g,o) = 1 & \forall g \in G, o \in \mathcal{O} \\ & \omega(g',\alpha|g,o) = 1 & \forall g,g' \in G, o \in \mathcal{O}, \alpha \in Act \\ & \sum\limits_{g\in G} \kappa(g) = 1. & \end{array} \tag{6.39}$$

Algorithm 6.3 outlines the main steps in the bounded policy iteration for the Conservative Optimization Criterion. Again, there are two distinct parts of the policy iteration. First, policy evaluation in which V^β is computed whenever some parameters of the controller changes (Steps 2, 10 and 20). The actual optimization algorithm to accomplish this step is found in Section 6.5.1. Second, after evaluating the current value function, an improvement is carried out either by changing the parameters of existing nodes, or if no new parameters can improve any node, then a fixed number of nodes are added to escape the local maxima (Steps 14-17). This is described in Section 6.5.3.

The two parts of policy improvement, namely the optimization to improve a given node, and addition of new nodes to escape local maxima are explained in detail in the succeeding sections.

Algorithm 6.3 Bounded Policy Iteration For Conservative Optimization Criterion

Input: (a) An initial feasible FSC, $\mathcal{G}$ with I-states $G = \{G^{tr}, G^{ss}\}$, such that $\eta_{av}^{ssd}(\mathfrak{r}) = 0$. (b) Maximum size of FSC N_{max}. (c) $N_{new} \leqslant N_{max}$ number of I-states

1: $improved \leftarrow True$
2: Compute the value vectors, $\vec{V}^{\beta}$ of the discounted reward criterion η_{β} as in Equation (6.23), or efficient approximation in Section 6.3.1.
3: **while** $|G| \leqslant N_{max}$ **and** $improved = True$ **do**
4: $improved \leftarrow False$
5: **for all** I-states $g \in G$ **do**
6: Set up the Constrained Improvement LP as in Section 6.5.1.
7: **if** Improvement LP results in optimal $\epsilon > 0$ **then**
8: Replace the parameters for I-state g
9: $improved \leftarrow True$
10: Compute the value vectors, $\vec{V}_g^{\beta}$ of the discounted reward criterion η_{β} as in Equation (6.23), or efficient approximation in Section 6.3.1.
11: **end if**
12: **end for**
13: **if** $improved = False$ **and** $|G| < N_{max}$ **then**
14: $n_{added} \leftarrow 0$
15: $N'_{new} \leftarrow \min(N_{new}, N_{max} - |G|)$
16: Try to add N'_{new} I-state(s) to $\mathcal{G}$ according to constrained DP backup in Section 6.5.3.
17: $n_{added} \leftarrow$ actual number of I-states added in previous step.
18: **if** $n_{added} > 0$ **then**
19: $improved \leftarrow True$
20: Compute the value vectors, $\vec{V}_g^{\beta}$ of the discounted reward criterion η_{β} as in Equation (6.23), or efficient approximation in Section 6.3.1.
21: **end if**
22: **end if**
23: **end while**

Output: $\mathcal{G}$

6.5.1 Node Improvement

The first observation is that the search over κ can be dropped. This simplification occurs because the initial node is chosen by computing the best valued node for the initial belief, i.e.,

$$\begin{aligned} \kappa(g_{init}) &= 1, \text{ where,} \\ g_{init} &= \underset{g}{\operatorname{argmax}} \left(\vec{\iota}^{\varphi}_{init}\right)^T \vec{V}^{\beta}_{g}. \end{aligned} \tag{6.40}$$

Once this initial node has been selected, the above objective only differs from the typical discounted reward maximization problem in the previous sections because of the presence of the new constraint

$$\eta^{ssd}_{av}(\mathfrak{r}) = 0, \tag{6.41}$$

which must be incorporated into the optimization algorithm. Using Theorem 6.1.3, the above constraint can be rewritten as

$$\eta^{ssd}_{av}(\mathfrak{r}) = 0 \iff \left(\vec{\iota}^{ssd}_{init}\right)^T \vec{\mathfrak{g}} = 0, \tag{6.42}$$

where $\vec{\mathfrak{g}}$ uniquely solves the Poisson Equation (6.2). This allows the node improvement to be written as a bilinear program. Again, one node g is improved at a time while holding all other nodes constant as follows.

I-state Improvement Bilinear Program:

$$\max_{\epsilon,\omega(g',\alpha|g,o),\vec{\mathfrak{g}},\vec{V}_{av}} \quad \epsilon$$

subject to

Improvement Constraints:

$$V^{\beta}([s,g]) + \epsilon \;\leqslant\; r^{\beta}(s) + \beta \sum_{s',g',\alpha,o} O(o|s)\omega(g',\alpha|g,o)T^{\mathcal{PM}^{\varphi}}(s'|s,\alpha)V^{\beta}([s',g']) \quad \forall s$$

Poisson Equation (if $g \in G^{ss}$):

$$\vec{V}^{av} + \vec{\mathfrak{g}} \;=\; \vec{r}^{av} + T_{mod}^{\mathcal{PM}^{\varphi}}\vec{V}^{av}$$
$$\vec{\mathfrak{g}} = T_{mod}^{\mathcal{PM}^{\varphi}}\vec{\mathfrak{g}}$$

Feasibility Constraints (if $g \in G^{ss}$):

$$\left(\vec{l}_{init,g}^{ss}\right)^{T}\vec{\mathfrak{g}} \;=\; 0$$

FSC Structure Constraints (if $g \in G^{ss}$):

$$\omega(g',\alpha|g,o) \;=\; 0 \;\text{ if } g' \in G^{tr}$$

Probability Constraints:

$$\sum_{g',\alpha}\omega(g',\alpha|g,o) \;=\; 1 \quad \forall o$$
$$\omega(g',\alpha|g,o) \;\geqslant\; 0 \quad \forall g',\alpha,o$$

(6.43)

Note that a node in G^{tr} does not have to guarantee that Product-POMDP states are not allowed to visit $Avoid_{\mathfrak{r}}^{\mathcal{PM}^{\varphi}}$ and hence the extra Poisson Equation and Feasibility Constraints that appear above need only be applied to I-state $g \in G^{ss}$. Further, the FSC structure constraints ensure that once the execution has transitioned to steady state, the I-states in G^{tr} can no longer be visited. They are in fact a reduction in the number of unknown variables. Next, the Poisson Equation constraints introduce bilinearity in the optimization. This is because the term $T_{mod}^{\mathcal{PM}^{\varphi}}$, which is linear in $\omega(g',\alpha|g,o)$, is multiplied by the unknowns $\vec{V}^{av}$ and $\vec{\mathfrak{g}}$ in the two sets of constraints that form the Poisson Equation.

6.5.2 Convex Relaxation of Bilinear Terms

Bilinear problems are in general hard to solve [27], unless they are equivalent to positive semidefinite or second order cone programs, which make the problem convex. Neither of these convexity assumptions hold for the bilinear constraints in Equation (6.43). However, several convex relaxation schemes exist for bilinear problems. Two of the most popular methods of relaxing the problem are to use the Reformulation-Linearization Technique (RLT) [132, 134] or the more recent SDP relaxation techniques used in [133, 136]. A good overview of these two methods can be found in [124]. This thesis utilizes a linear relaxation resulting from the RLT, which is summarized below, to obtain a possibly sub-optimal solution at each improvement step.

While RLT can be applied to a wide range of problems including discrete combinatorial problems, it is introduced here for the case of Quadratically Constrained Quadratic Problems (QCQPs) over unknowns $x \in \mathbb{R}^n$, $y \in \mathbb{R}^m$. The notation follows closely from [124]. A QCQP can be written as

$$\begin{aligned}
&\max \;\; x^T Q_o x + a_o^T x + b_o^T y \\
&\text{s.t.} \\
&\qquad x^T Q_k x + a_k^T x + b_k^T y \leqslant c_k \;\; \text{for } k = 1, 2, \ldots, p\,. \\
&\qquad l_{x_i} \leqslant x_i \leqslant u_{x_i} \qquad\qquad \text{for} i = 1, 2, \ldots, n \\
&\qquad l_{y_j} \leqslant y_j \leqslant u_{y_j} \qquad\qquad \text{for} j = 1, 2, \ldots, m
\end{aligned} \tag{6.44}$$

RLT is carried out as follows, for each x_i, x_j such that the product term $x_i x_j$ is non zero in either the objective or the constraints, a new variable X_{ij} is introduced, which replaces the product $x_i x_j$ in the problem. In addition, the bounds $l_{x_i}, l_{x_j}, u_{x_i}, u_{x_j}$ are utilized to produce four new linear constraints

$$\begin{aligned}
X_{ij} - l_{x_i} x_j - l_{x_j} x_i &\geqslant -l_{x_i} l_{x_j} \\
X_{ij} - u_{x_i} x_j - u_{x_j} x_i &\geqslant -u_{x_i} u_{x_j} \\
X_{ij} - l_{x_i} x_j - u_{x_j} x_i &\leqslant -l_{x_i} u_{x_j} \\
X_{ij} - u_{x_i} x_j - l_{x_j} x_i &\leqslant -u_{x_i} l_{x_j}.
\end{aligned} \tag{6.45}$$

The above constraints are the McCormick convex envelopes [97]. For bilinear programming with bounded variables, the McCormick convex envelopes are successively used in algorithms such as branch and bound [83] to successively obtain tighter relaxations to obtain globally optimal solutions. An efficient solver that incorporates this methodology is [89].

The following shows how to relax the bilinear constraints appearing in the Poisson Equation in the I-state Improvement Bilinear Program in Equation 6.43. Note first that while the Poisson Equation spans all global states, i.e., there are $\mathcal{S} \times |G|$ for each of the two sets of Poisson constraints, only the equations pertaining to the current I-state under consideration has bilinear terms. To see this, rewrite the Poisson constraints in block form,

$$\begin{bmatrix} \vec{V}^{av}_{g_1} \\ \vdots \\ \vec{V}^{av}_{g} \\ \vdots \\ \vec{V}^{av}_{g_{|G|}} \end{bmatrix} + \begin{bmatrix} \vec{\mathfrak{g}}_{g_1} \\ \vdots \\ \vec{\mathfrak{g}}_{g} \\ \vdots \\ \vec{\mathfrak{g}}_{g_{|G|}} \end{bmatrix} = \begin{bmatrix} \vec{r}^{av}_{g_1} \\ \vdots \\ \vec{r}^{av}_{g} \\ \vdots \\ \vec{r}^{av}_{g_{|G|}} \end{bmatrix} + \begin{bmatrix} T^{\mathcal{PM}^{\varphi}}_{mod,g_1} \\ \vdots \\ T^{\mathcal{PM}^{\varphi}}_{mod,g} \\ \vdots \\ T^{\mathcal{PM}^{\varphi}}_{mod,g_{|G|}} \end{bmatrix} \begin{bmatrix} \vec{V}^{av}_{g_1} \\ \vdots \\ \vec{V}^{av}_{g} \\ \vdots \\ \vec{V}^{av}_{g_{|G|}} \end{bmatrix} \text{ and}$$

$$\begin{bmatrix} \vec{\mathfrak{g}}_{g_1} \\ \vdots \\ \vec{\mathfrak{g}}_{g} \\ \vdots \\ \vec{\mathfrak{g}}_{g_{|G|}} \end{bmatrix} = \begin{bmatrix} T^{\mathcal{PM}^{\varphi}}_{mod,g_1} \\ \vdots \\ T^{\mathcal{PM}^{\varphi}}_{mod,g} \\ \vdots \\ T^{\mathcal{PM}^{\varphi}}_{mod,g_{|G|}} \end{bmatrix} \begin{bmatrix} \vec{\mathfrak{g}}_{g_1} \\ \vdots \\ \vec{\mathfrak{g}}_{g} \\ \vdots \\ \vec{\mathfrak{g}}_{g_{|G|}} \end{bmatrix}, \tag{6.46}$$

it can be seen that the bilinearity arises because the rows (in red) of $T^{\mathcal{PM}^{\varphi}}_{mod}$ which are linear terms of the unknowns $\omega(.,.|g,.)$ are multiplied by $\vec{V}^{av}$ and $\vec{\mathfrak{g}}$ respectively. The other rows of $T^{\mathcal{PM}^{\varphi}}_{mod}$ are not functions of the unknowns and their values are used from the values in the previous policy evaluation step. The total number of bilinear terms in both sets of equations is given by $2 \times |\mathcal{S}||\mathcal{O}||G||Act|$. Moreover, applying the convex relaxation requires that all terms appearing in bilinear products must have finite bounds. For the unknowns $\omega(.,.|g,.)$, $\vec{\mathfrak{g}}$ and $\vec{V}^{av}$ these bounds are given by

$$\begin{array}{rcccl} \vec{0} & \leqslant & \omega(.,.|g,.) & \leqslant & \vec{1} \\ \vec{0} & \leqslant & \vec{\mathfrak{g}} & \leqslant & \vec{1} \\ -\vec{M}_1 & \leqslant & \vec{V}^{av} & \leqslant & \vec{M}_2, \end{array} \tag{6.47}$$

where M_1, M_2 are large positive constants that are manually selected. This is because the feasibility set for $\vec{V}^{av}$ is dependent on the eigen values of $I - T^{\mathcal{PM}^{\varphi}}_{mod,g_{|}G|}$ [94], which is difficult to represent in terms of the optimization variables. During numerical implementation, this issue was not found to adversely effect the solution quality. This may be due to the fact that $\vec{V}^{av}$ does not appear in either the objective or in the feasibility constraints of Equation (6.43). In fact, for the choice of ω only constrains the value $\vec{\mathfrak{g}}$, whereas given ω and $\vec{\mathfrak{g}}$ a feasible value of $\vec{V}^{av}$ can always be found. A rigorous analysis of the effect of the choice for the bounds of $\vec{V}^{av}$ remains to be carried out.

6.5.3 Addition of I-States to Escape Local Maxima

When no I-state Improvement LP yields $\epsilon > 0$, a local maxima for the Bounded Policy Iteration (BPI) has been reached. The dual variables corresponding to the Improvement constraints in Equation (6.43) again give those belief states that are tangent to the backed up value function. The process for adding I-states will again involve forwarding the tangent beliefs one step and then checking if the value of those forwarded beliefs can be improved. However an additional check for recurrence

constraints will have to made if the involved I-state belongs to the G^{ss} states of the FSC controller. In addition, if an I-state is added to the FSC, it must also be assigned to either G^{tr} or G^{ss}, because the next policy evaluation iteration depends on the I-state partitioning in the computation of $T^{\mathcal{PM}^{\varphi}}_{mod}$. The procedure for adding I-states is provided in Algorithm 6.4.

Algorithm 6.4 can be understood as follows. Assume that a tangent belief b for some I-state g. Similar to Algorithm 6.2, instead of directly improving the value of the tangent belief, the algorithm tries to improve the value of forwarded beliefs reachable in one step from the tangent beliefs. This is given in Step 4 of Algorithm 6.4. Recall from Section 6.4.1 that when a new I-state is added, its successor states are chosen from the existing I-states. A similar approach is used in Algorithm 6.4. However, a new node may be added to either G^{tr} or G^{ss} depending on the I-state that generated the original tangent belief. Recall that I-states in G^{ss} have two additional constraints. First, no state in G^{ss} can transition to any state in G^{tr}. This is enforced by limiting the successor state candidates in Steps 6-9. Secondly, for improving a node in G^{ss}, the allowed actions and transitions must satisfy the Poisson Equation constraints of Equation (6.43). This further reduces or prunes the possible successor candidates in Step 10, which is elaborated as a separate procedure in Algorithm 6.5. The rest of the procedure is identical to Algorithm 6.2, except for Step 20, in which any newly added I-state is placed in the correct partition of G^{tr} or G^{ss}.

Algorithm 6.5 prevents any new I-states to choose a pair of action and successor I-state that may violate the Feasibility Constraints of Equation (6.43). In order to carry out this procedure, a phantom I-state, $g_{phantom} \in G^{ss}$ is temporarily added to the current FSC for a pair $(g, \alpha) \in candidates$. Next, the modified transition distribution $T^{\mathcal{PM}^{\varphi}}_{mod,phantom}$ is computed using Equation (5.3), and the Poisson Equation is solved to obtain a new $\vec{\mathfrak{g}}$ which can be used to verify the Feasibility Constraint. If this constraint is violated. i.e., then (g, α) is removed from the set $candidates$. Note that the algorithm works on a copy of the original FSC, and the solution of the Poisson Equation computed at the last Policy Evaluation step. The addition of $g_{phantom}$, and recomputation of the Poisson Equation is only used within Algorithm 6.5.

6.6 Finding an Initial Feasible Controller

So far, it has not been shown how an initial feasible controller may be found to begin the policy iteration. A feasible FSC is one which produces at least one φ-feasible recurrent set (Definition 3.5.3). This problem can be posed as a bilinear program. Assume a size $|G|$ and partitioning $G = \{G^{tr}, G^{ss}\}$ of the FSC has been chosen such that $|G^{tr}| > 0$ and $|G^{ss}| > 0$. Next, consider the Poisson Equation for the ssd-global Markov chain, in which the states in $Avoid^{\mathcal{PM}^{\varphi}}_{\mathfrak{r}} \times G^{ss}$ are sinks. However, instead of the charge of the Poisson Equation being r^{av}, consider the charge r^{β} in which the states in $Repeat^{\mathcal{PM}^{\varphi}}_{\mathfrak{r}} \times G^{ss}$ are rewarded. This equation is given by

Algorithm 6.4 Adding I-states to Escape Local Maxima of Constrained Optimization Criterion

Input: (a) Set B of tangent beliefs for each I-state. (b) A function $node : B \rightarrow G$ identifying the I-state which yields each tangent belief. (c) N_{new} the maximum number of I-states to add.

1: $N_{added} \leftarrow 0$.
2: **repeat**
3: Pick $b \in B$, $B \leftarrow B \backslash \{b\}, g \leftarrow node(b)$.
4: Compute the set of forwarded beliefs, Fwd, as in Steps 4-10 of Algorithm 6.2.
5: **for all** $b_{fwd} \in Fwd$ **do**
6: **if** $g \in G^{tr}$ **then**
7: $candidates \leftarrow G \times Act$.
8: **else**
9: $candidates \leftarrow G^{ss} \times Act$.
10: $candidates \leftarrow$ PruneCandidates($candidates$, b_{fwd}, $\vec{V}^{av}$, $\vec{\mathfrak{g}}$) using Algorithm 6.5.
11: **end if**
12: **if** $candidates \leftarrow \varnothing$ **then**
13: Go to step 5.
14: **end if**
15: Apply the r.h.s. of DP Backup Equation to b_{fwd}

$$V^{\beta,backedup}(b_{fwd}) = \max_{(g,\alpha) \in candidates} \left\{ r^{\beta}(b_{fwd}) + \beta \sum_{o \in \mathcal{O}} \Pr(o|b_{fwd}) \left(b^{o,\alpha}_{fwd}(s) V^{\beta}_{g}(s) \right) \right\} \quad (6.48)$$

where, $b^{o,\alpha}_{fwd}$ is computed for each product state $s' \in \mathcal{S}$ as follows

$$b^{o,\alpha}_{fwd}(s') = \sum_{s} T(s'|s,\alpha) \frac{O(o|s) b_{fwd}(s)}{\sum_{o' \in \mathcal{O}} O(o'|s) b_{fwd}(s)}. \quad (6.49)$$

16: Note the maximizing action α^* and I-state g^*.
17: **end for**
18: **if** $V^{\beta,backedup}(b_{fwd}) > V^{\beta}(b_{fwd})$ **then**
19: Add new deterministic I-state g_{new} such that $\omega(g_{new}|g^*, \alpha^*, o) = 1 \ \forall o \in \mathcal{O}$.
20: Assign g_{new} to correct FSC partition as follows:

$$g_{new} \in \begin{cases} G^{tr} & \text{if } g \in G^{tr} \\ G^{ss} & \text{otherwise.} \end{cases} \quad (6.50)$$

21: $N_{added} \leftarrow N_{added} + 1$.
22: **end if**
23: **if** $N_{added} \geqslant N_{new}$ **then**
24: **return**
25: **end if**
26: **until** $B = \varnothing$.

Algorithm 6.5 Pruning candidate successor I-states and actions to satisfy recurrence constraints.

Input: Set of candidate successor states and actions $candidates \subseteq G^{ss} \times Act$.

1: **for all** $(g, \alpha) \in candidates$ **do**
2: Add new state $g_{phantom}$ to G^{ss} to create a larger FSC where,

$$\omega(g, a | g_{phantom}, o) = 1 \quad \forall o \in \mathcal{O}. \tag{6.51}$$

3: Compute $T_{mod}^{\mathcal{PM}^{\varphi}}$ and $\vec{\iota}_{init}^{ss}$ for the new larger global ssd Markov chain.
4: Solve Poisson Equation for the new larger global Markov chain to obtain solutions $\vec{\mathfrak{g}}$, $\vec{V}^{av}$.
5: **if** Any Feasibility Constraints in Equation (6.43) are violated under the larger FSC **then**
6: $candidate \leftarrow candidates \backslash \{(g, \alpha)\}$.
7: **end if**
8: **end for**
9: **return** *candidates*

$$\begin{aligned} \vec{\mathfrak{g}}_{feas} &= T_{mod}^{\mathcal{PM}^{\varphi}} \vec{\mathfrak{g}}_{feas}, \text{ and} \\ \vec{V}_{feas}^{av} + \vec{\mathfrak{g}}_{feas} &= \vec{r}^{\beta} + T_{mod}^{\mathcal{PM}^{\varphi}} \vec{V}_{feas}^{av} \end{aligned} \tag{6.52}$$

Then, it can be shown that some state in $Repeat_{\mathfrak{r}}^{\mathcal{PM}^{\varphi}} \times G^{ss}$ is recurrent and can be reached from the initial distribution with positive probability if and only if $\exists g \in G^{ss}$ such that

$$\left(\vec{\iota}_{init}^{\mathcal{PM}^{\varphi}}\right)^{T} \vec{\mathfrak{g}}_{feas,g} > 0. \tag{6.53}$$

However, the constraint of never visiting the avoid states still applies. These procedures and constraints can be collected together in the following bilinear maximization problem.

$$\max_{\omega,\vec{V}^{av},\vec{V}^{av}_{feas},\vec{\mathfrak{g}},\vec{\mathfrak{g}}_{feas}} \left(\vec{\iota}^{\mathcal{PM}^{\varphi}}_{init}\right)^{T} \vec{\mathfrak{g}}_{feas,g}$$

subject to

Poisson Equation 1:

$$\begin{aligned} \vec{V}^{av} + \vec{\mathfrak{g}} &= \vec{r}^{av} + T^{\mathcal{PM}^{\varphi}}_{mod} \vec{V}^{av} \\ \vec{\mathfrak{g}} &= T^{\mathcal{PM}^{\varphi}}_{mod} \vec{\mathfrak{g}} \end{aligned}$$

Poisson Equation 2:

$$\begin{aligned} \vec{V}^{av}_{feas} + \vec{\mathfrak{g}}_{beta} &= \vec{r}^{\beta} + T^{\mathcal{PM}^{\varphi}}_{mod} \vec{V}^{av}_{feas} \\ \vec{\mathfrak{g}}_{feas} &= T^{\mathcal{PM}^{\varphi}}_{mod} \vec{\mathfrak{g}}_{feas} \end{aligned} \tag{6.54}$$

Feasibility constraints ($\forall g \in G^{ss}$)

$$\left(\vec{\iota}^{ss}_{init,g}\right)^{T} \vec{\mathfrak{g}} = 0$$

FSC Structure Constratins:

$$\omega(g', \alpha | g, o) = 0 \text{ if } g \in G^{tr} \text{ and } g \in G^{ss}$$

Probability constraints:

$$\begin{aligned} \sum_{g',\alpha} \omega(g', \alpha | g, o) &= 1 \quad \forall o \\ \omega(g', \alpha | g, o) &\geqslant 0 \quad \forall g', \alpha, o \end{aligned}$$

Any positive value of the objective $\left(\vec{\iota}^{\mathcal{PM}^{\varphi}}_{init}\right)^{T} \vec{\mathfrak{g}}_{feas,g}$ gives a feasible controller, and therefore the optimization need not be carried out to optimality. If the problem is infeasible, then states in G^{ss} can be successively added to search for a positive objective.

6.7 Case Studies

In this section, case studies for the bounded policy iteration algorithm described in Section 6.5 are shown. The first example demonstrates the effectiveness of the algorithm to optimize the transient phase of the controlled system, while the second example illustrates the effectiveness in improving the steady state behavior of the controlled system. The case studies use the grid world system models used in the case studies of Chapter 4, whose graphical representation are repeated in Figure 6.5 for convenience.

6.7.1 Case Study I - Stability with Safety

LTL Specification: The LTL specification is given by

$$\varphi_2 = \diamondsuit \square b \wedge \square \neg c. \tag{6.55}$$

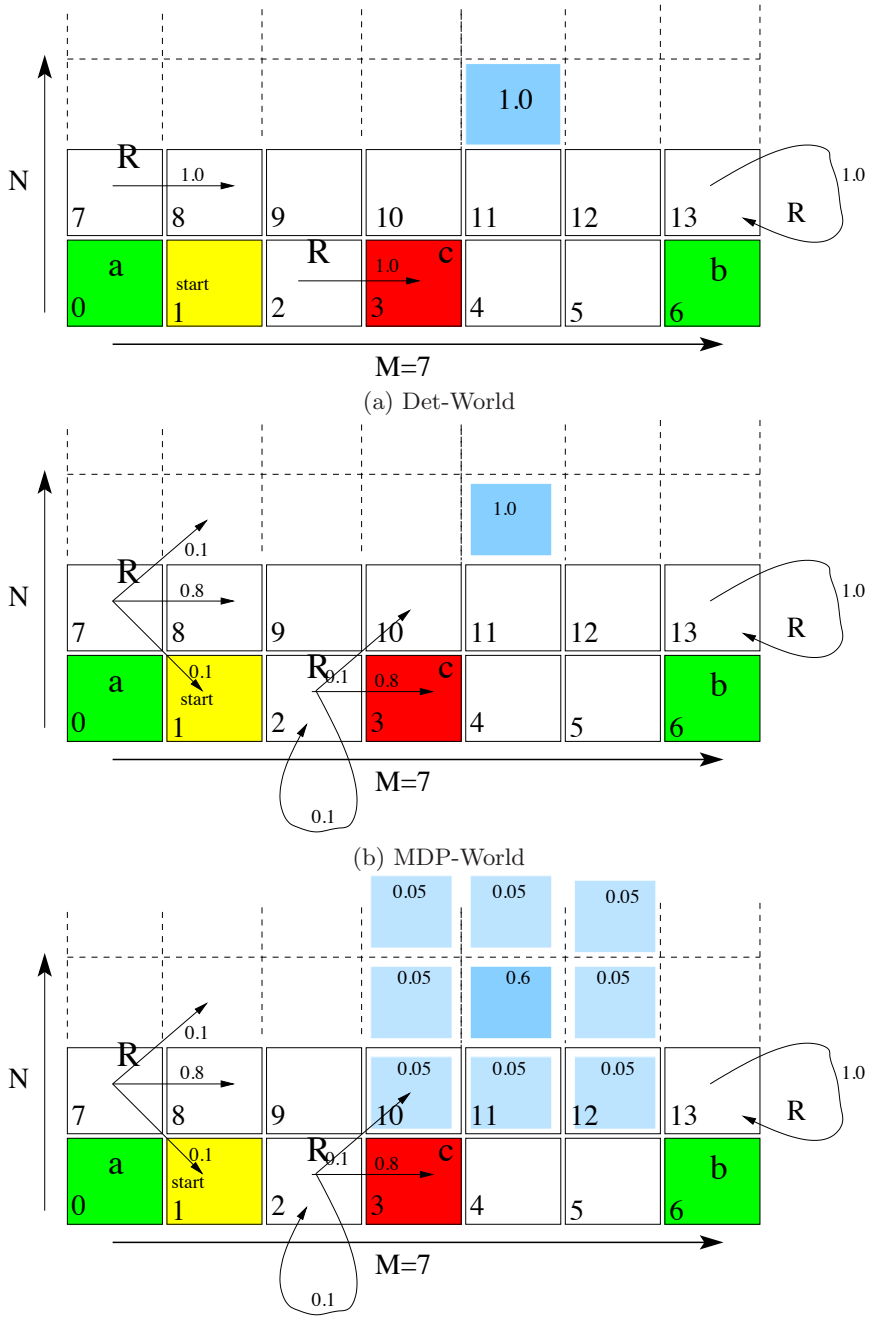


Figure 6.5: System models for Policy Iteration case studies. (a) Det-World (b) MDP-World (c) POMDP-World. A description of these systems can be found in Figure 4.4.

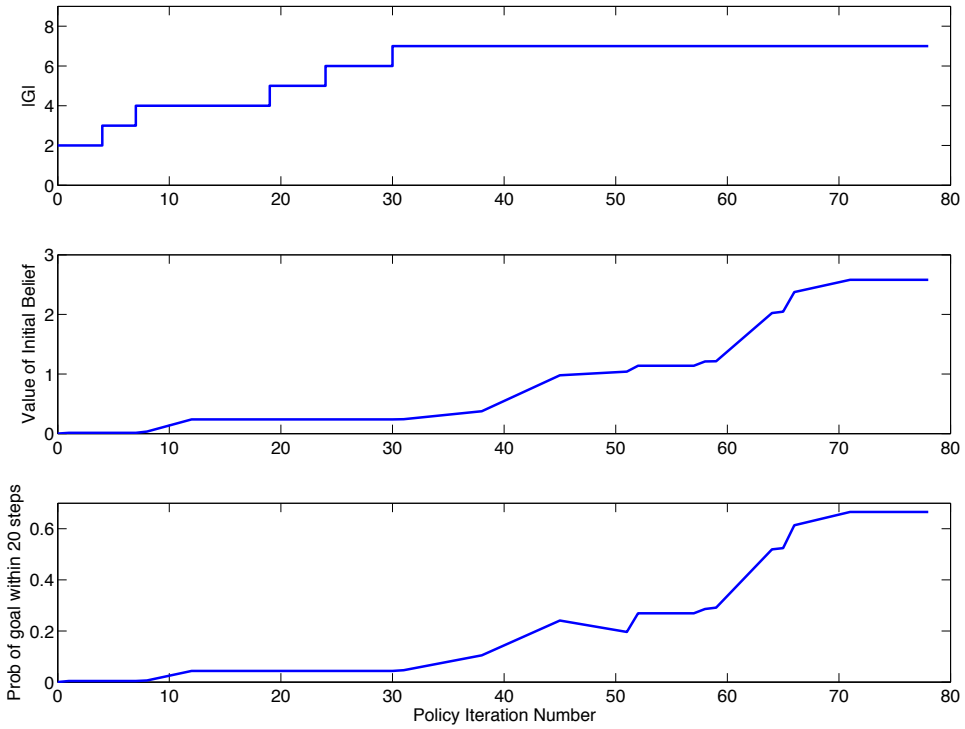


Figure 6.6: Transient behavior optimization using Bounded Policy Iteration. The x-axis denotes the number of policy improvement steps carried out. (a) The growth of the FSC size. (b) The value of the initial belief increases monotonically with each iteration. This value denotes the expected long term discounted reward for the given initial belief. (c) Since the goal is to reach cell 6, this sub-figure shows the increase in probability of reaching the goal state within 20 time steps as the FSC is optimized.

where b and c, shown in Figure 6.5 are requirements for the robot to navigate to cell 6, and stay there, while avoiding cell 3, respectively.

Results: The difficulty in this specification is that the robot must localize itself to the top edge of the corridor before moving rightward to cell 6. Note that a random walk performed by the robot is feasible: there is a finite probability that actions chosen randomly will lead the robot to cell 6 without visiting cell 3. The FSC used to seed the BPI algorithm was chosen to have uniform distribution for I-state transitions and actions. This is another advantage as compared to the gradient ascent algorithm, in which initial parametrization for the FSC would lead to zero gradient as mentioned in Section 4.6. Figure 6.6 shows the result of the BPI in detail. It can be seen that the value of the initial belief increases monotically with successive policy improvement steps, which includes both the optimization of Equation (6.43) and the addition of I-states to escape local maxima, as discussed in Section 6.5.3.

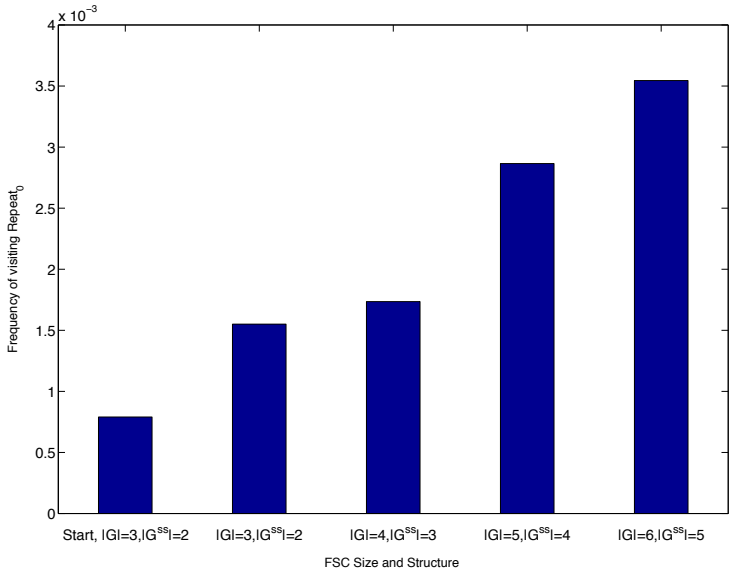


Figure 6.7: Effect of Bounded Policy Iteration on steady state behavior. Bounded Policy Iteration applied to the specification φ_1. The above graph shows the improvement in steady state behavior as the size of the FSC increases. Only states in G^{ss} were allowed to be added. The y-axis denotes the expected *frequency* with which states in $Repeat_0^{\mathcal{P}\mathcal{M}^\varphi}$ were visited for the product-POMDP resulting for the POMDP-World from Figure 6.5(c) and the DRA of φ_1 given by Equation (6.56).

6.7.2 Case Study II - Repeated Reachability with Safety

This case study illustrates how the Bounded Policy Iteration, especially the addition of I-states to the FSC, improves the steady state behavior of the controlled system.

System Model and LTL specification: The model used for this example is the POMDP-World of Figure 6.5(c) for $N = 3$, and the LTL specification is φ_1 of Equation (4.51), repeated here.

$$\varphi_1 = \square\diamondsuit a \wedge \square\diamondsuit b \wedge \square\neg c. \tag{6.56}$$

Results: For this example, the controller was seeded with a feasible FSC of size $|G| = 3$, with $|G^{ss}| = 2$, using Algorithm 4.2. After the first few policy improvement steps, the initial I-state was found to be in G^{ss}. Since by construction, once the FSC transitions to an I-state in G^{ss} it can no longer visit states in G^{tr}, when local maxima was encountered, subsequently all new I-states were assigned to G^{ss}. The improvement in steady state behavior with the addition of each I-state is shown in Figure 6.7, where it can be seen that the expected frequency of visiting $Repeat_0^{\mathcal{P}\mathcal{M}^\varphi}$ steadily increases with the addition of I-states.

6.7.3 Concluding Remarks

In this chapter, it was shown how the Poisson Equation, which is closely related to the Bellman Equation arising in dynamic programming, can be leveraged to solve the Conservative Optimization Problem. The problem was shown to be bilinear and one convex relaxation method involving McCormick envelopes was introduced as a solution. The stochastic bounded policy iteration algorithm, which is normally applied to constrained discounted reward problem, was adapted to the case in which certain states were required to be never visited. This allowed that a path in the global Markov chain of the controlled POMDP eventually is absorbed in a recurrent set that does not include states from $Avoid_{\mathfrak{r}}^{\mathcal{PM}^{\varphi}}$, while simultaneously incentivizing frequent visits to some state(s) in $Repeat_{\mathfrak{r}}^{\mathcal{PM}^{\varphi}}$. The key benefit of using this variant of dynamic programming is that it allows a controlled growth in the size of the FSC, and can be treated as an anytime algorithm in which the performance of the controller improves with successive iterations, but can be stopped by the user based on time or memory considerations. Case studies highlighting key attributes of the algorithm can be found in Section 6.7, and its application to robotic manipulation tasks can be found in Section 7.5.

Chapter 7

Robot Task Planning for Manipulation

This chapter gives a detailed introduction to task planning in robotics. Task planning has traditionally been studied under the area of domain independent planning and therefore, some historical overview of this field of study is also provided. This chapter differentiates three different types of planning/control problem in manipulation tasks – the path or motion planning, simple sequencing to achieve short horizon tasks, and high level task planning for manipulation robots. This is because manipulation tasks present some unique and specific challenges relating to kinematic constraints of manipulators, some of which can be addressed using specialized re-manipulation or re-grasping techniques. This re-grasping can therefore be separated with relative ease from the path planner as well as the high level task planner. Finally, the chapter shows the details of applying the policy iteration algorithm of Chapter 6 to a few concrete task planning problems inspired by the real challenge tasks presented to the teams participating in the DARPA ARM-S challenge.

7.1 Introduction

Modern manipulation robots [3,6,9] are no longer confined to large industrial settings with controlled environments, or work on highly specialized assembly or machining tasks. Instead, they are expected to function in the same environment as humans and adapt to a variety of goals. These new desired behaviors from robots also come with additional burdens arising out of large, possibly unknown, environmental disturbances. Moreover they must rely heavily on sensors and perception algorithms that may be unreliable in unconstrained environments. These two issues are addressed in planning problems by incorporating probabilistic or even non deterministic changes in the robot's system model and by introducing partial observability.

7.1.1 Robotic Paradigms

All robots work on some paradigm incorporating the three main components that they need to function: Sense, Plan and Act [28, 103, 109]. The classical way to model robotic behavior was to look at its operation in a top down fashion with a heavy focus on planning. The robot first senses the world, plans the next action, and finally acts. Then the process repeats itself. This is sometimes called the Hierarchical or Deliberative Paradigm and is shown in Figure 7.1(a). It assumes *a-priori* knowledge of the world model. Another paradigm is the Reactive Paradigm, in which several instances of very tight Sense-Act couplings run concurrently and sensor data is used to compute which action to take, independent of what the other instances are doing, with some mechanism to arbitrate. There is no explicit abstract model of the world. The Sense-Act couplings are termed "behaviors" and are typically implemented using finite state machines and connected directly to the sensors and actuators. See Figure 7.1(b). These concurrent independent behaviors lead to overall behavior that is *emergent*. While the reactive paradigm has had success with speed and resilience against dynamic environments, they have had limited success with planning for *long range goals* and *optimizing* the robot behavior. For long range goals, it is necessary to have a deliberative planner capable of planning executions that realize the goals. Optimizing against some constraints on time or efficiency is also crucial. This suggests the use of a hybrid structure with a hierarchy of capabilities. The low levels can still be highly reactive, while the higher levels provide increasingly intensive deliberation for long range goal satisfaction. Figure 7.1(c) gives a figurative configuration of the various classical components in a hybrid architecture.

7.1.2 Planning in Robotics

A robot can be expected to carry out several types of planning: e.g., robots may need to plan for information gathering tasks or better perception of objects; mobile robots and autonomous vehicles must plan for navigation, which subsumes path and trajectory planning; networked robots with communication constraints need to plan for this information exchange; manipulation robots need to work with physical objects and must deal with kinematic constraints. These constraints arise out of object and manipulator geometry, dynamic issues during lifting and manipulation due to gravitational and frictional effects, etc.

There are domain specific planners that are crucial for performance of single tasks, e.g., path planning techniques for mobility or manipulator planning in the configuration space; scheduling algorithms for resource planning problems, SLAM [85] for the optimal actions for map building in unknown environments, etc. These types of planners are efficient for their domain, as they leverage the assumptions on the underlying structure of the problem and its eventual application. However, the planners of interest in this work are those that work by abstracting the domain and are therefore

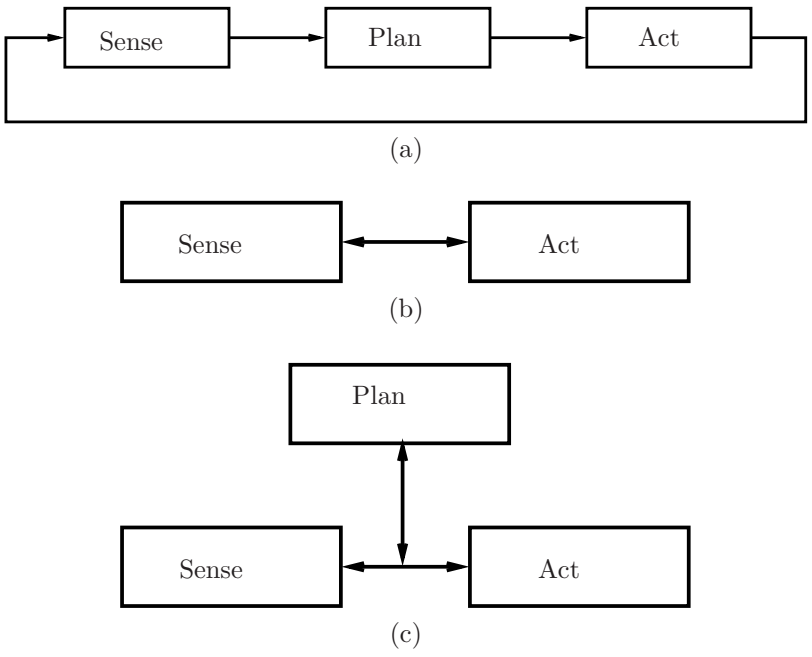


Figure 7.1: Various Robotic Paradigms. (a) The Deliberative/Hierarchical Robotic Paradigm in which the robot senses the state of itself and environment, plans its next action and then acts, in sequence. (b) The Reactive Robotic Paradigm in which multiple concurrent Sense-Act couplings react directly to the environment without an abstract world model or high level deliberation.(c) The Hybrid Robotic Paradigm in which there are low level sense-act loops, with higher level planners/sequencers that may work with explicit worlds models for increasingly involved computations.

applicable to many problems using the same mathematical tools. These rely on the *model* of all the actions available to the robot or controller, which includes preconditions on the state of the world for applicability and each action's effect on the world. These planners start with the initial or measured state of the world, and a goal, and output a rule for choosing the actions needed to reach the goal, while satisfying constraints. This is called the *plan synthesis* problem [47].

For articulated robots performing manipulation tasks, a general high level software architecture is shown in Figure 7.2. This architecture closely approximates the architecture for the JPL/Caltech entry to the DARPA Autonomous Robotic Manipulation challenge (see Section 7.2).

The planner can obtain the initial world state from the robot's own inference engine, or from human input using a model of how its actions effect the world (dashed purple). A plan is generated to achieve high level goals. The plan is either a sequence of actions or a rule to generate these actions and is handed to the sequencer for execution.

The sequencer, sometimes also called the executive, is a lower level module that monitors the execution of the plan, and is possibly equipped with local planners to achieve the high level actions. For mobile robots, the path and/or trajectory planner may be incorporated at this level. For a robot which must manipulate objects, a configuration space path and/or trajectory planner such as an RRT-planner [82] provides the sequencer with specific parameters to complete the current action, bypassing the high level task planner. The sequencer then passes these specific parameters to the controller.

Typically, the controller implements a fast control loop between the actuators and sensors in order to achieve the goals provided by the sequencer. For example, it may implement a linear quadratic Gaussian regulator [5] for navigational path following. For articulated robots it may utilize feedback and feed-forward adaptive methods to control a combination of end effector position and exerted force/torque. The capabalities of the manipulator system are encoded as a set of behaviors. A behavior may consist of a hybrid automaton, each discrete state describing the motion using the dynamics of the system and also encoding end conditions. For example, a "contact-grasp" behaviour may comprises of multiple discrete states. First, a free-space motion is carried out to a defined offset from the object, which may require visual servoing. This may be followed by a task frame velocity input towards the object grasp point until contact is made. Further motions upon contact include finger closure, finger strain control etc.

Both the controller and the inference engine provide feedback to the sequencer to update it regarding the status of individual actions. If the plan received by the sequencer already accounts for every outcome of individual actions, the sequencer has enough information to either proceed with the next action or recover from an unsuccessful action. However, if something unseen happens, or the original plan was partial, the sequencer may optionally request a new plan (black dashed line). This type of planner is an online planner which refines the high level plan as execution proceeds.

The inference engine fuses information from all the different sensors – cameras, laser range finder, tactile, force/torque sensors – to infer the state of the world. It utilizes numerous specialized algorithms such as computer vision, Kalman filtering, etc., to perform map building and self localization [85], pose estimation [93], object segmentation and identification [139]. It can access the state of the controller, e.g., force closure criterion can indicate the grasped status of objects [105]. In order to estimate the status of the world, it may additionally utilize a predictive engine on the model of the world, and use Bayesian algorithms to find good estimates and an idea of its own estimation error [84].

While the world itself can be quite complex, the planner works from a *model* of the world. The model is comprised of a state transition system with a function γ that determines how the state evolves.

It is worth reiterating the difference between the sequencer and the high level planner in Figure 7.2. Planning is significantly harder when it is a hybrid problem, i.e., when the planner must simultaneously search over sequences of discrete actions and also search for continuous paths within actions, especially since the continuous controllers effect the outcome at the discrete level as well. Although there is now a large body of work on controlling hybrid systems, and a recent push towards integrating the discrete planning problem with continuous controllers [78], these systems are usually confined to low state space dimensions. For manipulation robots, the configuration space [39] can unfortunately be very high dimensional [70], as the number of movable objects increase.

This thesis focuses exclusively on planning over finite discrete choices. The problems arising out of continuous low level controllers are assumed to be captured by the introduction of uncertainty in the model. This is in addition to the uncertainty due to actual noise that actuators introduce and also due to unexpected world behavior. Moreover the partially observability at the highest task or action planning level captures the fact that the inference engine is usually imperfect. During dexterous tasks, manipulated objects may be miss-classified, and it may be difficult to detect completion of some tasks, such as successful disassembling of small parts from a fixture. POMDPs offer a powerful way to model robotic tasks. POMDPs are a general and expressive model class and are now being studied extensively in the field of domain independent planning [120]. Therefore, a brief historical review of domain independent planning is provided in the next section.

7.1.3 Background on Domain Independent Planning

The classical planning problem representation, e.g. STRIPS [44], ADL [110] starts with a first order language over a (finite) set of atomic propositions and constant literals which may represent the objects in the world. For example, *hammer1* and *hammer2* may refer to specific hammers. There may be variables that abstract these objects. For example *tool* is a variable that can take be instantiated to *hammer1* or *hammer2*. The atomic propositions are typically truth valued statements describing

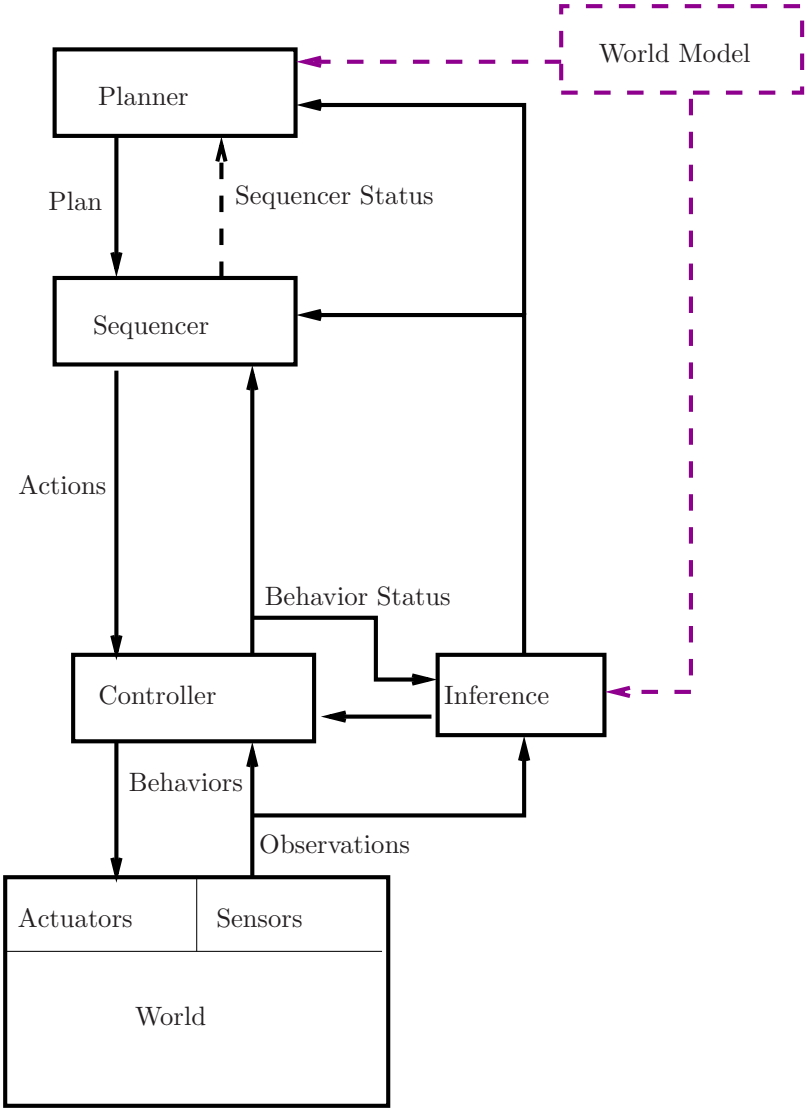


Figure 7.2: A Hybrid Robot Architecture.

properties of the object, e.g., *grasped_left_hand(tool)* says that some object represented by *tool* is grasped in the left hand. When the object represented by all variables appearing in atomic propositions are instantiated by an object or constant literal, then the proposition is called *grounded*. The opposite of grounding is called *lifting*. For example, the proposition *grasped_left_hand(tool)* describes the state of an abstract entity *tool*. Similarly, the operator *GRASP(left_hand, tool)* may similarly define the effect of grasping action on the abstract entity. Lifting allows compact representation of the world and its dynamics. It is independent of the instantiation of abstract entities by actual objects, which may differ situationally.

Let the set of grounded atomic propositions be given by $\mathcal{L}$. Then, the state space of the world model is given by all possible truth assignments, $2^{\mathcal{L}}$. Next, there are a finite set of operators that have two main components:

1. **Preconditions:** These are list of atomic propositions, that must be true or false for the operator to be applied.

2. **Effects:** A list of atomic propositions that become true or false as a result of applying the operator. There has historically been ambiguity as to what happens to propositions that are not listed in the effects, and most implementations assume that the remaining propositions remain unchanged. In newer representations such as PDDL [99] these effects may be conditional.

An example of an operator can be

$$
\begin{aligned}
&place(tool, table): \\
&\quad \text{preconditions: } grasped(tool) \\
&\quad \text{effects: } \neg grasped(tool) = False,\ on(table, tool) = True.
\end{aligned}
$$

Exact format, syntax and semantics vary across several representations [44, 47, 110]. This is another operator that is lifted, i.e., it can be used for any instantiation of *tool*. When all propositions appearing in operators are grounded, the operator is termed grounded and is called an action.

In classical domain independent planning, the goals are reachability goals. The planner simply needs to find a path (a sequence of actions and hence a sequence of intermediate points in the state space) that takes the initial state of the world to a given final state. The goal is also specified as a list of atomic propositions that must be true or false. Any state that agrees with the goal is a valid end state. Classical problems do not deal with finding "optimal" paths.

Since the first formalization of planning problems, several new problem domains have been added. For example, probabilistic effects as in Markov decision processes and partial observability, which is the topic of thesis, are now an important subject in the AI planning and robotics communities [107, 120]. Another addition is that of durative actions, in which time is explicitly represented. Planning under concurrency is another important area. The goals of planning problems have also

become more complex: optimality criteria based on rewards / cost, constraint satisfaction problems and temporal goals are all now studied under the umbrella of planning.

In order to incorporate these new classes of problems, as defined by the mathematical models and goal criteria, the representation of planning problems has also evolved. Over the past decade, the AI planning community has been standardizing the representation of increasingly new classes of planning problems in a consistent fashion using the Propositional Domain Description Language (PDDL) [99] and, recently, Relational Dynamical Influence Diagrams (RDDL) [129] to incorporate lifted representations of probabilistic concurrent systems.

This thesis is concerned with probabilistic domains: specifically Partially Observable Markov Decision Processes (POMDP), which are powerful models that allow uncertain disturbances to be incorporated into the model. They are widely popular, and despite their computational complexity, the optimal POMDP policy problems for these domains have been shown to work over increasingly large state spaces in seconds [79]. In fact, infinite state space models have also been introduced [41]. In large engineering applications, such as a concurrent city wide traffic management system, several thousand actions can be taken simultaneously. It quickly becomes intractable to even represent all possible concurrent actions if they are explicitly enumerated. Similarly, many action effects only depend on a subset of state variables, as do the rewards or cost. For these domains a compact representation is crucial if they are to be applied in reality. This has led to a crucial area in (PO)MDP planning where the domain is factored [20, 52]. In factored representations, the transition probabilities are conditioned over the evaluations of a subset of the state variables. In order to capture these new developments in stochastic planning, recently there has been a push to standardize the representation of problems of this nature using RDDL [129]. RDDL uses Dynamic Bayesian Networks (DBN), Algebraic Decision Diagrams (ADD) and the lifted representations which allow for compact descriptions of large domains. DBNs represent how variables at one time step affect each other at the next time step [102]. ADDs are a generalization of Binary Decision Diagrams (BDD) to allow efficient representations of functions and implementation of algorithms such as matrix multiplication, Gaussian elimination, etc. [7].

7.2 The DARPA Autonomous Robotic Manipulation Software Challenge

The DARPA Autonomous Robotic Manipulation Software (ARM-S) program is a dexterous manipulation challenge in which participating teams are asked to complete complicated object manipulation tasks with minimal high level supervision. These tasks need to be carried out in partially unknown, changing environments. The goal has been to push the technology for robots to become useful in adaptive manufacturing, in-home care, and deployment in emergencies under hazardous conditions.

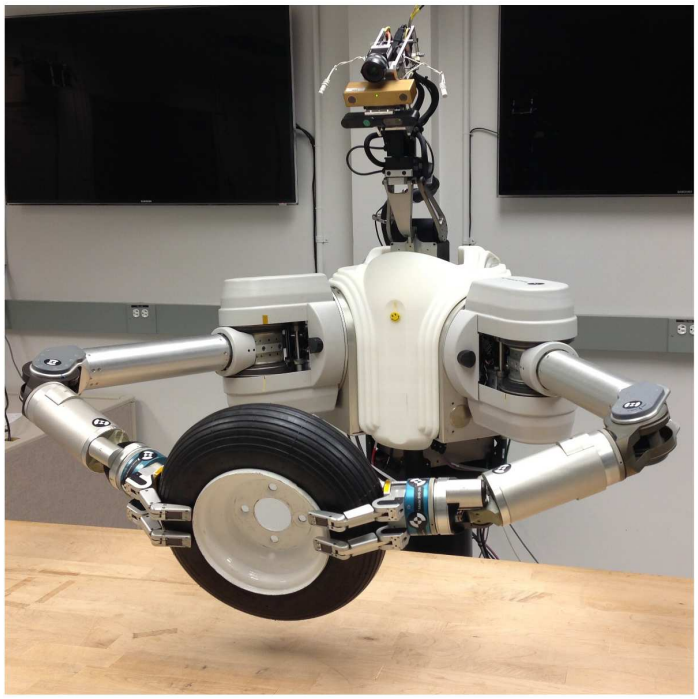


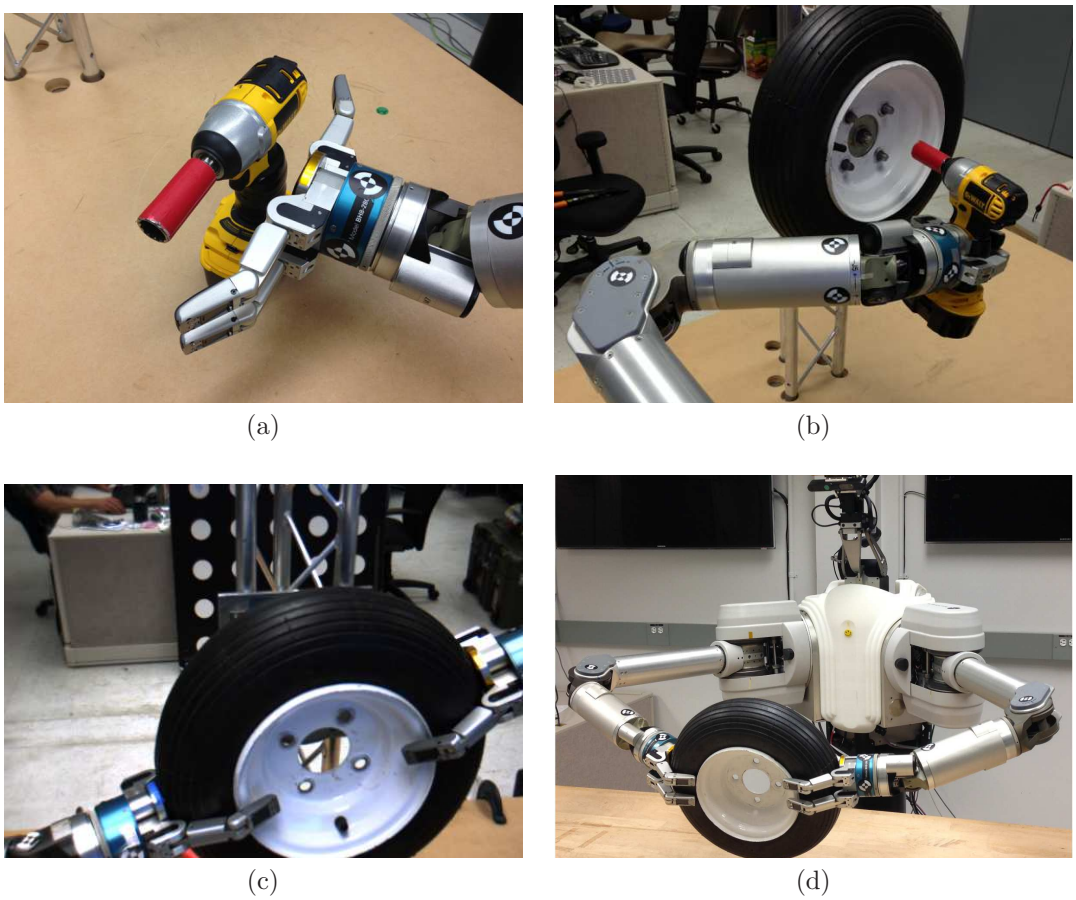
Figure 7.3: The DARPA ARM-S Robot.

The first phase of the challenge included six teams, while the second and final phase was comprised of three teams. This thesis work has been motivated by the task planning challenges faced by the JPL/Caltech team during the second phase, in which long sequences of dual armed tasks needed to be carried out in the correct sequence in order to complete the challenge tasks. The end-to-end system description of the technologies used in this competition and the system integration details can be found in [63, 64].

The ARM robot [3], see Figure 7.3, consists of two arms that are Barrett Technology 7-DOF WAM arms with a 6-DOF force sensor at each wrist. The Barrett BH8-280 hands have a strain gauge in each of the three fingers and tactile sensing pads on the palm and distal finger surfaces. The sensor head consists of a Point Grey Research Bumblebee2 color stereo camera, PrimeSense ASUS Xtion-Pro depth camera, Prosilica Gig-E color camera, and two microphones, mounted on a 4-DOF neck.

7.2.1 An Example DARPA Challenge Task for ARM-S

One of the tasks that the ARM-S teams were required to carry out was a wheel change scenario. The process of changing a wheel includes the following main tasks: finding a battery operated impact drive on the table (work-space), removing lug nuts from the wheel using the impact driver, removing the wheel from the axle and placing it on the table. This sequence of desired behavior is seen in



(a) (b)

(c) (d)

Figure 7.4: Wheel Removal Task for the ARM-S Robot. (a) Pick impact driver. (b) Remove the four lug nuts that attach the wheel to the hub. (c) Remove the wheel from the hub. (d) Place wheel on table.

Figure 7.4.

7.3 Overview of Planning Challenges in ARM-S

The JPL/Caltech entry to the software challenge closely follows the architecture shown in Figure 7.2 and the details of each component can be found in [64]. This section focuses on the planning challenges faced in carrying out the challenge tasks in the competition.

7.3.1 Motion Planning

For single primitive tasks, such as grasping an object with a free hand, the JPL/Caltech ARM-S software calls upon the motion planner. There are three main components in the motion planner: Manipulation Sets, Manipulation Planner and the Arm Planner. The manipulation sets are sets of relative poses which are needed for controller behaviors, a primary object (e.g., a tool to be used),

and an optional secondary object (e.g., the object on which the tool operates). For example, the manipulation set for the behavior *power grasp* and the primary object *impact driver* is a set of poses which describe the location of the hand relative to the impact driver as well as a corresponding set of finger parameters. If the robot hand can be brought into this relative pose w.r.t. the impact driver, then the control behavior has high chance of succeeding with local feedback / feed forward controller. These sets were generated offline because the CAD models of the objects were available. The Manipulation Planner, in combination with the Arm Planner, chooses amongst reachable relative poses for a given primitive task. The primary goal of the Arm Planner is to work in the 7-DOF arm joint space to find collision free paths to the poses suggested by the Manipulation Planner. The Arm Planner uses RRT to plan for both arms in master slave configuration [64].

7.3.2 Limited Task Planner / Sequencer for ARM-S

The ARM-S challenge did not require the full capabilities of synthesizing long task sequences autonomously. The high level sequence of actions to accomplish each task was permitted to be encoded explicitly by the participating teams. However, the need for a basic task sequencer / executive was necessary. Since this problem is the main motivation for this work, the next few sections expand on this topic.

The dexterous manipulation tasks in DARPA ARM-S program require tasks to be executed in a particular order for successful completion. The task level planner maintains communication with other components of the system such as the estimator, planner, and control. The material presented in this section was carried out by the author as part of the work for the JPL/Caltech entry into DARPA ARM-S challenge and follows closely the work published in [64].

The overall JPL/Caltech software architecture relies on a model of the world, described by world state W^{model} at any time. This includes the state of all objects $\{O\}$ in the work-space, and the state of the robot, R. The state of the objects includes a linkage list L that describes the possible links or attachment between objects. Furthermore, the state of the robot includes a grasp state Gr that describes which object each manipulator is grasping. However, since the world state is inferred using sensors and possibly inference based algorithms, the task level planner maintains the *estimated* state of the physical world W^{est}, which is updated using the estimator.

The task planner additionally maintains a library of atomic tasks $T^{atom} = \{T_1^{atom}, T_2^{atom}, \dots\}$ which correspond to single behaviors that some component (including itself) of the system can carry out. Each component when asked to execute an atomic task (behavior), may in turn carry out a sequence of actions, which the task level planner does not track. The task level planner concerns itself with the overall success or failure of such tasks. Some examples of tasks are ESTIMATE_ WORLD_STATE (estimation), GENERATE_ MANIPULATION_SET (planner), PLAN_ARM_TO_ POSE (planner) and MOVE_USING_PLAN (control). These atomic tasks can take arguments α_i^{atom}

Task Sequence 7.1 A common compound task

EXECUTE_BEH_SEQ(grasp_type, $O_{primary}$, $O_{secondary}$)
1: (Estimator) w = ESTIMATE_WORLD_STATE
2: (Planner) M = GENERATE_MANIPULATION_SET(w, grasp_type, $O_{primary}$, $O_{secondary}$)
3: (Task Planner) **if** $M = \varnothing$, **return** Failure, **exit**
4: **repeat**
5: (Task Planner) pick new $m_i \in M$
6: (Planner) p = PLAN_ARM_TO_POSE(w, m_i)
7: (Task Planner) if p is not valid, **goto step 11**
8: (Control) MOVE_ARM_USING_PLAN(p)
9: (Control) MOVE_FOR_BEHAVIOR(grasp_type)
10: **return** Success, **exit**
11: **until** M is exhaustively sampled.
12: **return** Failure, **exit**

such as the object on which the action must be performed, or the current world state. An atomic task with an instantiated argument can be thought of as a map

$$T_i^{atom}(.|\alpha_i^{atom}) : W \rightarrow W\,.$$

In the above equation, $W \in \{W^{model}, W^{est}\}$, implying that the application of these atomic tasks can be on W^{est} or W^{model}. In the former case, the tasks are carried out by the actuators and sensors, in the presence of noise and imperfections. However, in the latter case, application of a task implies simulating the expected change in the world state assuming perfect observation and control.

Atomic tasks that involve kinematic changes correspond to executing control behaviors, which are tight sense-act loops that carry out a (small number of) single atomic task(s), such as moving the robot hand to an object until contact and securely grasping the object. When such tasks are simulated, W^{model} is propagated using the same *expected motion* that the controller uses to complete the corresponding behavior. In fact, the inverse of this *expected motion* is used by the manipulation planner to generate the starting pose in the first place. This approach makes the simulation consistent with idealized execution by the controller. The resulting state, W^{model}, kinematically satisfies the *end-conditions* for the control behavior. Changes in grasp state Gr and linkage list L are also updated by appropriately translating the *end-conditions* of the control behavior. Thus, simulated grasping of an object correctly updates Gr with the object grasped, while a parts assembly behavior correctly updates L in W^{model}.

The library also contains some basic sequences (seq.) of atomic tasks, called compound tasks, $T^{comp} = \{T_1^{comp}\ ,\ T_2^{comp}\ ,\ \ldots\}$, where T_k^{comp} are finite sequences of elements of T^{atom}. The parameters for each atomic task are stacked or sequentially generated to make the argument for the sequence. A common compound task is given by Task Sequence 7.1.

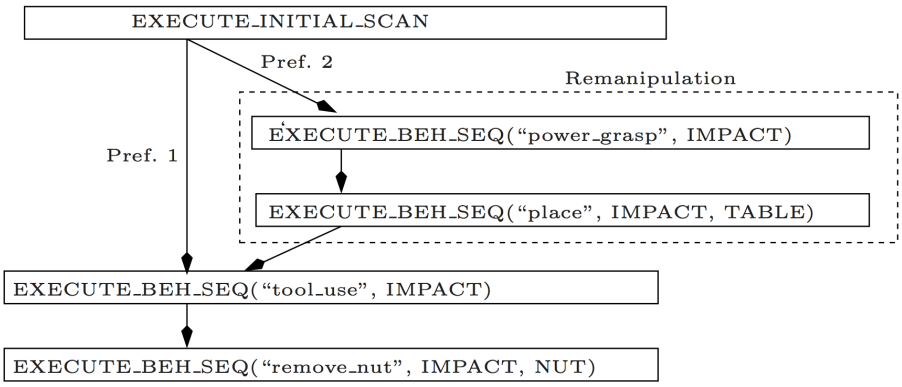


Figure 7.5: Abstracted digraph for removing nuts with impact driver, Example 2. Two alternate routes exist with lower preference for re-manipulation.

In the ARM-S program, manipulation tasks are specified as an ordered list of T_j^{comp}. This list is internally translated to a directional graph where each node represents an atomic task. Note that there is no need to restrict ourselves to two levels of encoding task hierarchy. One can make compound tasks from other completely defined compound tasks as well.

7.3.2.1 Re-manipulation or Re-grasping

On many occasions, due to kinematic constraints on the robot's motion, there may be no feasible plans to accomplish a behavior. For example, for instruction EXECUTE_BEH_SEQ("tool_use_grasp", IMPACT, Ø) the grasp planner may find no feasible solutions in the current pose of the impact on the table. To handle this situation, the task sequencer can optionally execute a sub-sequence for re-manipulating or re-grasping the impact driver on the table so that subsequently a tool use grasp is feasible.

Adding alternative optional sub-sequences involves allowing the nodes of the task sequence digraph to have multiple children and parent nodes, and this approach can be represented as in Figure 7.5. Whenever a node has multiple children, the child nodes are ordered by preference.

7.3.2.2 Kinematically Dependent Tasks

In many situations, some (compound) tasks in the sequence are kinematically dependent on each other. For the tool use example, this can arise in multiple ways.

Example 1: Assume that the robot can pick up the impact driver with a hand pose that is amenable to tool use (press trigger). However, after the impact driver is lifted off the table with a certain pose, the RRT planner may fail to find a path to orient the impact driver over the nut that must be removed from the fixture. This situation happens usually when the initial plan for picking up the impact is near the physical joint limit of the wrist, even though other wrist solutions may

exist.

Example 2: This example considers the case when re-manipulation of the impact driver is needed in order that the next task (tool use grasp) is feasible.

Kinematically dependent tasks are usually linked by manipulation sets. They are therefore solved by linking them together inside nested iterative tasks and exhaustively searching until a feasible solution is found for all tasks in the linked subset. The task sequence for solving Example 1 is shown in Task Sequence 7.2.

Task Sequence 7.2 Task sequence for Example 1

EXECUTE_LINKED_SEQ("tool_use_grasp", IMPACT, ∅)
1: (Estimator) w_1 = ESTIMATE_WORLD_STATE
2: (Planner) M_1 = GENERATE_MANIPULATION_SET(w_1, "tool_use_grasp", IMPACT, ∅)
3: (Task Planner) **if** $M_1 = \varnothing$, **return** Failure, **exit**
4: **repeat**
5: (Task Planner) pick new $m_{1,i} \in M_1$
6: (Planner) p_1 = PLAN_ARM_TO_POSE(w_1, $m_{1,i}$)
7: (Task Planner) if p_1 is not valid, **goto step 21**
8: (Control) MOVE_ARM_USING_PLAN(p_1)
9: (Control) MOVE_FOR_BEHAVIOR(tool_use_grasp)
10: (Estimator) w_2 = ESTIMATE_WORLD_STATE
11: (Planner) M_2 = GENERATE_MANIPULATION_SET(w_2, "remove_nut", IMPACT, NUT)
12: (Task Planner) **if** $M_2 = \varnothing$, **goto step 21**
13: **repeat**
14: (Task Planner) pick new $m_{2,j} \in M_2$
15: (Planner) p_2 = PLAN_ARM_TO_POSE(w_2, $m_{2,j}$)
16: (Task Planner) if p_2 is not valid, **goto step 20**
17: (Control) MOVE_ARM_USING_PLAN(p_2)
18: (Control) MOVE_FOR_BEHAVIOR("remove_nut")
19: **return** Success, **exit**
20: **until** M_2 is exhaustively sampled.
21: **until** M_1 is exhaustively sampled.
22: **return** Failure, **exit**

Similarly, the sequence for Example 2, which require a re-manipulation event before executing the sequence from Example 1 consists of searching over four manipulation sets, corresponding respectively to:

1. EXECUTE_BEH_SEQ("power_grasp", IMPACT, ∅)
2. EXECUTE_BEH_SEQ("place", IMPACT, TABLE)
3. EXECUTE_BEH_SEQ("tool_use_grasp", IMPACT, ∅)
4. EXECUTE_BEH_SEQ("remove_nut", IMPACT, NUT)

Figure 7.5 showed an abstracted digraph of the task level plan for this example. Images of the robot resorting to re-manipulation are shown in Figure 7.6. All four compound tasks are kinematically linked.

7.3.2.3 Kinematic Verification Based Execution

Once the task sequence directed graph is populated from the task specification, a kinematics-only verification of the entire sequence is carried out. First a sensor based estimate of the world is made and is used to initialize W^{model}. A depth-first traversal of the sequence is carried out which preferentially chooses highly ranked child nodes first. As each node, n, is encountered its effect is *simulated* on the output of the parent node corresponding to its initial condition, $W^{model}(parent(n))$. Planning tasks can produce additional quantities such as the manipulation set M or a plan p, which can be considered to augment the world state. Each node can flag *success* in which case a new world state, $W^{model}(n)$ is generated and search proceeds or *failure* in which case we backtrack until a parent with unexplored child or an unfinished iteration task is encountered.The verification ends either in success (a node with no children is reached with no failure) or fails if exhaustive search produces no successful path.

Once the task sequence execution starts, it is natural to observe discrepancy between W^{est} and W^{model} from the kinematic verification at any node. If this discrepancy is large and tasks are kinematically linked, the manipulation sets may need to be recomputed along with the associated RRT plans, since they are sensitive to the actual world state.

7.4 Task Planning for ARM-S

The preceding sections outlined the limited functionality of the ARM-S sequencer. In order to apply the full power and algorithms of domain free task planning for manipulation robotics, it is necessary to represent the capabilities of the robot in a language that admits a compact lifted representation. Another key realization during the ARM-S challenge was that partial observability was not only an issue at the local / continuous control level, but also filtered upto the task level. That is, partial observability makes it difficult to determine if one segment of a sequence has been successfully completed.

7.4.1 Probabilistic Outcomes and Partial Observability at the Task Level

Consider the task of wheel removal by the robot. Once the robot has grasped the impact driver, it must now remove the four lug nuts as shown in Figure 7.7. Panel (a) shows the perception algorithm in the top left box. It segments the image from the visual sensors to locate the nuts. The controller and the perception algorithm must have high accuracy to ensure that the tool tip is placed properly over the lug-nuts for removal. The local continuous errors arise out of a combination of noise and inaccuracy in sensors, and disturbance and inaccuracies in the robot actuators. At the local control level, visual servoing tries to make the action robust to these sources of uncertainty.

However, successful completion at the task level cannot be guaranteed. A typical recourse is to model the possible outcomes probabilistically. Panel (b) shows the events once lug nuts removal actions have been completed. The visual sensors and the force torque sensors in the wrist may not have enough accuracy to measure whether the lug nuts were successfully removed with perfect confidence. However, if the robot proceeds forward and tries to remove the wheel from its attachment to the hub, inferring success and failure for this later task has much more accuracy. Moreover, success and failure can provide information about the success or failure at the lug nut removal case. This could be modeled as conditional distributions as follows.

Partial Observability

$$\begin{aligned} O\left[ObsAttachedNutToWheel(nut_i) \mid AttachedNutToWheel(nut_i)\right] &= p \\ O\left[\neg ObsAttachedNutToWheel(nut_i) \mid AttachedNutToWheel(nut_i)\right] &= 1-p \\ O\left[ObsAttachedNutToWheel(nut_i) \mid \neg AttachedNutToWheel(nut_i)\right] &= 1-q \\ O\left[\neg ObsAttachedNutToWheel(nut_i) \mid \neg AttachedNutToWheel(nut_i)\right] &= q \end{aligned} \tag{7.1}$$

Probabilistic Outcomes

$$\begin{aligned} T\left[\neg AttachedNutToWheel(nut_i) \mid AttachedNutToWheel(nut_i), ActRemoveNut(nut_i)\right] &= r \\ T\left[AttachedNutToWheel(nut_i) \mid AttachedNutToWheel(nut_i), ActRemoveNut(nut_i)\right] &= 1-r \end{aligned} \tag{7.2}$$

7.4.2 LTL Goals for ARM-S Task Planner - Case Studies

7.4.2.1 Simple Reachability Goal

The first case study will look at a simple reachability task for the ARM-S robot in which it must remove the wheel and place all tools and objects on the table. Formally, this can be written as an LTL formula over the state space

ARM-S Task 1

$$\varphi_1^{\text{ARM-S}} = \diamondsuit\left(\neg AttachedWheelToHub \wedge \neg GraspedWheel\right). \tag{7.3}$$

Note that this problem is no different from the classical planning problem, except for the non-classical domain of Partially Observable Markov Decision Process. In fact it can be solved by already known methods of reward maximization, by simply assigning a reward of +1 to the states in which the formula $\neg AttachedWheelToHub \wedge \neg GraspedWheel$ holds. The DRA for this specification is given in Figure 7.8.

7.4.2.2 Temporally Extended Goal

In order to demonstrate the power and flexibility of using LTL for goal specifications, next consider a particular ARM-S robot deployed on an assembly line. The robot should perform the wheel removal task as the assembly fixtures appear in front of it. The fixtures are assumed to be sent by a scheduler which routes them to available robots. The wheel removal task may take an unknown amount of time due to the probabilistic nature of the task. However once the task is finished, it would be useful if robot itself could signal that it can receive a new task. We can formulate this requirement as an LTL formula as follows:

ARM-S Task 2

$$\begin{aligned}\varphi_2^{\text{ARM-S}} \;=\; & \Box\Diamond\left(\neg \mathit{AttachedWheelToHub} \wedge \neg \mathit{GraspedWheel}\right) \wedge \\ & \Box\left(\left(\neg \mathit{AttachedWheelToHub} \wedge \neg \mathit{GraspedWheel}\right) \implies \bigcirc \mathit{AvailableSignal}\right).\end{aligned} \tag{7.4}$$

7.4.3 Description of the System in RDDL

I choose to compactly represent the ARM-S system for the two case studies using RDDL [129]. As mentioned before, RDDL is based on the idea of a Dynamic Bayes Net (DBN), and supports factored representations of planning domains. The RDDL description for ARM-S Task 2 described earlier is shown in Listing 7.1. The graphical representation of the DBN is shown in Figure 7.9. Note that the full power of RDDL is not needed for this problem. For example, to satisfy the above goals, an explicit reward function is not required, since the reward assignment will be designed as part of the algorithm presented in this thesis. Moreover RDDL allows for continuous conditional distributions, even though algorithms for these types of POMDP problems are still in their nascency. However, the above tasks can be formulated by restricting the conditional distributions to the Bernoulli (coin toss) likelihoods or deterministic outcomes which are denoted using the Kronecker delta function.

Listing 7.1: Description of the ARM-S Task Planning Domain

```
domain arms {

requirements = { partially-observed };

types { nut : object;};

pvariables {
// Non-fluents
// These are model parameter values that don't change during execution,
// but are initialized before computing the policy.
p                       : {non-fluent, real, default = 1};
q                       : {non-fluent, real, default = 1};
r                       : {non-fluent, real, default = 1};
```

```
//State variables
GraspedImpact            : {state-fluent, bool, default = false};
GraspedWheel             : {state-fluent, bool, default = false};
AttachedWheelToHub       : {state-fluent, bool, default = true};
AttachedNutToWheel(nut) : {state-fluent, bool, default = true};
AvailableSignal          : {state-fluent, bool, default = false};

//Actions
ActGraspImpact         : {action-fluent, bool, default = false};
ActGraspWheel          : {action-fluent, bool, default = false};
ActRemoveNut(nut)      : {action-fluent, bool, default = false};
ActRemoveWheel         : {action-fluent, bool, default = false};
ActPlaceImpact         : {action-fluent, bool, default = false};
ActPlaceWheel          : {action-fluent, bool, default = false};
ActSignal              : {action-fluent, bool, default = false};
ActIdle                : {action-fluent, bool, default = false};

//Observations
ObsGraspedImpact            : {observ-fluent, bool};
ObsGraspedWheel             : {observ-fluent, bool};
ObsAttachedWheelToHub       : {observ-fluent, bool};
ObsAttachedNutToWheel(nut) : {observ-fluent, bool};
ObsAvailableSignal          : {observ-fluent, bool};
};

cpfs {

GraspedImpact' = if (~GraspedImpact ^ ~GraspedWheel ^ ActGraspImpact)
                    then Bernoulli(1)
                 else if (GraspedImpact ^ ActPlaceImpact)
                    then Bernoulli(0)
                 else KronDelta(GraspedImpact);

GraspedWheel' = if (~GraspedImpact ^ ~GraspedWheel ^ ActGraspWheel)
                   then Bernoulli(1)
                else if (GraspedWheel ^ ActPlaceWheel)
                   then Bernoulli(0)
                else KronDelta(GraspedWheel);

AttachedWheelToHub' = if (~AttachedWheelToHub ^ ~GraspedWheel ^ AvailableSignal)
                         then Bernoulli(1)
                      else if (AttachedWheelToHub ^ GraspedWheel ^ ActRemoveWheel ^
                         (forall_{?n : nut} ~AttachedNutToWheel(?n)))
                         then Bernoulli(0)
```

```
                    else KronDelta(AttachedWheelToHub);

AvailableSignal' = if (AttachedWheelToHub) then Bernoulli(0)
                   else if (ActSignal) then Bernoulli(1)
                   else KronDelta(AvailableSignal);

AttachedNutToWheel'(?n) = if (AttachedNutToWheel(?n) ^ GraspedImpact ^
                             ActRemoveNut(?n)) then Bernoulli(r)
                          else if (~AttachedWheelToHub ^ ~GraspedWheel ^
                             AvailableSignal) then Bernoulli(1)
                          else KronDelta(AttachedNutToWheel(?n));

ObsGraspedImpact = KronDelta(GraspedImpact');
ObsGraspedWheel = KronDelta(GraspedWheel');
ObsAttachedWheelToHub = KronDelta(AttachedWheelToHub');
ObsAvailableSignal = KronDelta(AvailableSignal');
ObsAttachedNutToWheel(?n) = if (AttachedNutToWheel'(?n))
                               then Bernoulli(p)
                            else Bernoulli(1-q);
}; }

instance wheelchange {
  domain = arms;
  non-fluents { p = 0.5; q = 0.5; r = 0.8; };
  objects { nut : {nut1, nut2, nut3, nut4}; };
  init-state { GraspedImpact = false;};
  max-nondef-actions = 1; //Allow only 1 action at each time step
  discount = 0.9;
}
```

7.5 Application of Bounded Policy Iteration to ARM-S Tasks

This section presents the results and insights from applying the bounded policy iteration algorithm to the ARM-S Tasks 1 and 2.

7.5.1 Preprocessing

Before the numerical results for the ARM-S task planning problems are shown, some discussion about the size of the problem is presented. Given the planning domain in Listing 7.1, the size of a naive implementation, in which all states, actions and observations are incorporated into the optimization is given in Table 7.1.

The problem size can be reduced by a few simple preprocessing steps. The first approach is to

	Formula	φ_1 ARM-S	φ_2 ARM-S
Model state space size	$\lvert\mathcal{S}^{model}\rvert$	256	256
Number of observations	$\lvert\mathcal{O}\rvert$	256	256
Number of actions	$\lvert Act\rvert$	11	11
Size of the DRA state space	$\lvert Q\rvert$	2	5
Product state space size	$\lvert\mathcal{S}\rvert = \lvert\mathcal{S}^{model}\rvert\lvert Q\rvert$	512	1280
Total number of unknowns for ω	$\lvert\mathcal{O}\rvert\lvert Act\rvert\lvert G\rvert^2$	$2816\times\lvert G\rvert^2$	$2816\times\lvert G\rvert^2$
Total number of unknowns for $\vec{V}^{av}$, $\vec{\mathfrak{g}}$, $\vec{V}^{\beta}$ (each)	$\lvert\mathcal{S}\rvert\lvert G\rvert$	$512\times\lvert G\rvert$	$1280\times\lvert G\rvert$

Table 7.1: Problem size of policy iteration for naive implementation.

restrict the *model state space* to the reachable states from the initial condition. Note that the initial condition is given by

$$\begin{aligned}\neg GraspedImpact \wedge \neg GraspedWheel \wedge \neg AvailableSignal \wedge AttachedWheelToHub \\ \wedge \left(\textstyle\bigwedge_{i=1}^{4} AttachedNutToWheel(nut_i)\right).\end{aligned} \tag{7.5}$$

Similarly, when taking the product with the DRA, further reduction in the number of states can be made by keeping only the reachable states in consideration. In order to carry out the actual state space reduction, graph algorithms such as connected component computation were used. When applied to the partially observable ARM-S planning domain, it was found that the first reduction resulted in a model state space size of $\lvert\mathcal{S}^{model}\rvert = 88$. The number of observations could be reduced to $\lvert\mathcal{O}\rvert = 128$. Next, the product POMDP was constructed using Definition 3.2.1 and only those product states that were reachable from the initial state were kept. For the product POMDP resulting from $\varphi_1^{\text{ARM-S}}$, the size of the product state space was found to be $\lvert\mathcal{S}\rvert = 144$. For $\varphi_2^{\text{ARM-S}}$, the product state space size was only $\lvert\mathcal{S}\rvert = 160$.

Usually in task planning domains, each action can be applied successfully only in certain states. In completely observable cases, this fact reduces the planning problem size considerably since the agent can focus exclusively on those actions that are capable of changing the state. In partially observable domains, this reduction process can still be carried out, but to a smaller extent. In order to understand this issue completely, consider a particular observation $o_k \in \mathcal{O}$. Then, let $\mathcal{S}^{model}_{o_k} \subseteq \mathcal{S}^{model}$ be defined as follows. $\forall\ s \in \mathcal{S}^{model}$,

$$s \in \mathcal{S}^{model}_{o_k} \quad \text{if and only if} \quad O(o_k|s) > 0. \tag{7.6}$$

Equation (7.6) implies that $\mathcal{S}^{model}_{o_k}$ is the set of states which can generate observation o_k with non-zero probability.

Next, for any action $\alpha \in Act\backslash\{ActIdle\}$, if $T(s|s,\alpha) = 1, \quad \forall\ s \in \mathcal{S}_{o_k}$, then the parameter $\omega(g, o_k|g', \alpha)$ can be set to zero for all $g, g' \in G$, and be removed from the optimization over the

	φ_1ARM-S	φ_2ARM-S
Model state space size	88	88
Number of observations	128	128
Number of actions	11	11
Size of the DRA state space	2	5
Product state space size	144	160
Total number of unknowns for ω	$288\times\|G\|^2$	$288\times\|G\|^2$
Total number of unknowns for $\vec{V}^{av}$, $\vec{\mathbf{g}}$, $\vec{V}^{\beta}$ (each)	$144\times\|G\|$	$160\times\|G\|$

Table 7.2: Reduced problem size for policy iteration after basic preprocessing.

FSCs. In other words, if making a single observation o_k localizes the state of the product-POMDP to a few states, then only those actions need to be considered for o_k, which move these states forward. The exception in case of robotics is an action that explicitly says that the robot should do nothing, which is never removed from consideration. This is useful in two ways:

- When tasks are mainly about reaching a goal state, the idling of a robot after task completion is likely to be the most energy and cost conserving option.
- If the best policy is, in fact, to do nothing, the robot uses the action that is specifically meant for this, and does not rely on the fact that taking an inadmissible action will leave the state unchanged. The resulting policy is easily understood by humans, and all actions have intuitive meaning. Additionally, it allows the initial model to be safely written with the assumption that the state remains unchanged when inadmissible actions are taken. This simplifies the encoding or description of the problem. However, this aspect may have to be evaluated on the basis of the actual application.

When the above procedure is carried out for the arm task, it was found that the total number of unknown ω's reduced to 288 for both ARM-S tasks. The final size of the problem using all the above preprocessing steps can by found in Table 7.2.

7.5.2 ARM-S Task 1

This task was tested under varying ARM-S parameters, p, q and r. The parameters p and q change the partial observability of the RDDL model as they appear in computing the observation distribution (See (Equation 7.4.1)). The parameter r controls the probability of successively removing a lug nut from the wheel fixture when the action $RemoveNut(nut_i)$ is issued. Four cases are studied:

1. **Deterministic Model:** In this case, all parameters $p = q = r = 1$, denoting that the lug nut removal task is always successful, and that the states $AttachedNutToWheel(nut_i)$ are perfectly observable.

2. **MDP:** In this case, $p = q = 1$, denoting that the lug-nut states are fully observable, but the actions $RemoveNut(nut_i)$ are only successful with probability $r = 0.8$ (see Equation (7.4.1)).

3. **POMDP ($p = q = 0.7$):** In this case, the sensing algorithm for lug-nut states are imperfect. The observation probabilities are given by

$$\begin{aligned}
\Pr(ObsAttachedNutToWheel(nut_i)|AttachedNutToWheel(nut_i)) &= p &= 0.7 \\
\Pr(\neg ObsAttachedNutToWheel(nut_i)|AttachedNutToWheel(nut_i)) &= 1-p &= 0.3 \\
\Pr(\neg ObsAttachedNutToWheel(nut_i)|\neg AttachedNutToWheel(nut_i)) &= q &= 0.7 \\
\Pr(ObsAttachedNutToWheel(nut_i)|\neg AttachedNutToWheel(nut_i)) &= 1-q &= 0.3.
\end{aligned} \tag{7.7}$$

4. **POMDP ($p = q = 0.5$):** In this case, the sensing algorithm for lug-nut states provide no information. The observation probabilities are given by

$$\begin{aligned}
\Pr(ObsAttachedNutToWheel(nut_i)|AttachedNutToWheel(nut_i)) &= p &= 0.5 \\
\Pr(\neg ObsAttachedNutToWheel(nut_i)|AttachedNutToWheel(nut_i)) &= 1-p &= 0.5 \\
\Pr(\neg ObsAttachedNutToWheel(nut_i)|\neg AttachedNutToWheel(nut_i)) &= q &= 0.5 \\
\Pr(ObsAttachedNutToWheel(nut_i)|\neg AttachedNutToWheel(nut_i)) &= 1-q &= 0.5.
\end{aligned} \tag{7.8}$$

The results for the BPI algorithm are shown Figure 7.10. For the deterministic and MDP case, the optimization indeed yielded a deterministic FSC, which coincides with a plan from a typical deterministic MDP planner. The POMDP cases yielded probabilistic actions and I-state transitions, but were found to be very sparse. In fact, the fraction of non-zero ω's with respect to the total unknown ω's in Table 7.1 was found to be 0.0095 when $|G| = 6$.

7.5.3 ARM-S Task 2

The main goal of this task is to demonstrate that robot behavior can be optimized for repeated behaviors such as performing tasks on an assembly line while maintaining safety conditions. The results are are shown in Figure 7.11. Again, it can be seen that for the deterministic and MDP cases, the controllers could be optimized with a single I-state in G^{ss}. For both these cases, the resulting controller was completely deterministic. The bars in the figure show the expected frequency of finishing one sequence of removing the wheel and turning on the AvailableSignal. For the POMDP cases, the frequency was much lower. In addition, the smallest controllers for the two POMDP cases were found to have size 6 for $p = q = 0.7$, and 8 for $p = q = 0.5$. As before, ω was found to be

sparse, the number of non-zero parameters were only 0.0091 times the total number of unknown ω's as shown in Table 7.1 when $|G| = 8$.

It was observed during numerical simulation that the POMDP case with $p = q = 0.5$ was prone to be stuck in a local maxima when started from an initial controller that was constructed using Algorithm 4.2. The BPI algorithm had to be started from a controller of size 6 obtained from the POMDP case study with $p = q = 0.5$, from which the policy could be subsequently improved.

7.5.4 Resulting Control Policy

Since the controller has internal states, and observation dependent transitions, it is difficult to completely describe the FSC resulting from the application of the Bounded Policy Iteration here. However, the qualitative behavior of the FSC is shown in Figure 7.12. It can be seen that the FSC results in predominantly deterministic behavior in regions of full observability. However, the subsequence of removing the nuts and making the decision to proceed forward by placing the impact, are probabilistic.

7.6 Concluding Remarks

This chapter gave a detailed outlook of the crucial position and difficulties of task planning in a hybrid robotic paradigm. It also touched upon domain independent planning which offers a powerful method to represent the abstraction of a robot's functionality. Domain independent representation is naturally amenable to the hierarchical architecture of most modern autonomous platforms and are crucial to deploy robots into unstructured environments. Next, specific challenges of dexterous manipulation tasks were highlighted using the Caltech/JPL entry into the DARPA ARM-S challenge and its use of a limited domain specific sequencer. The need for a high level planner that ensures the realizability of long term, complex goals is crucial. Finally, the novel policy iteration algorithm developed in this thesis was applied to a few case studies that were inspired by the real task challenges presented to the participants during the DARPA challenge. These examples highlighted how a small lifted model of the robot and its environment, and simple high level goals can lead to a problem of significant size. The ability to carry out state space reduction was found to be important in finding a control policy using a computer. However, the policy iteration algorithm was shown to be effective in finding simple intuitive discrete control policies for these real scenarios.

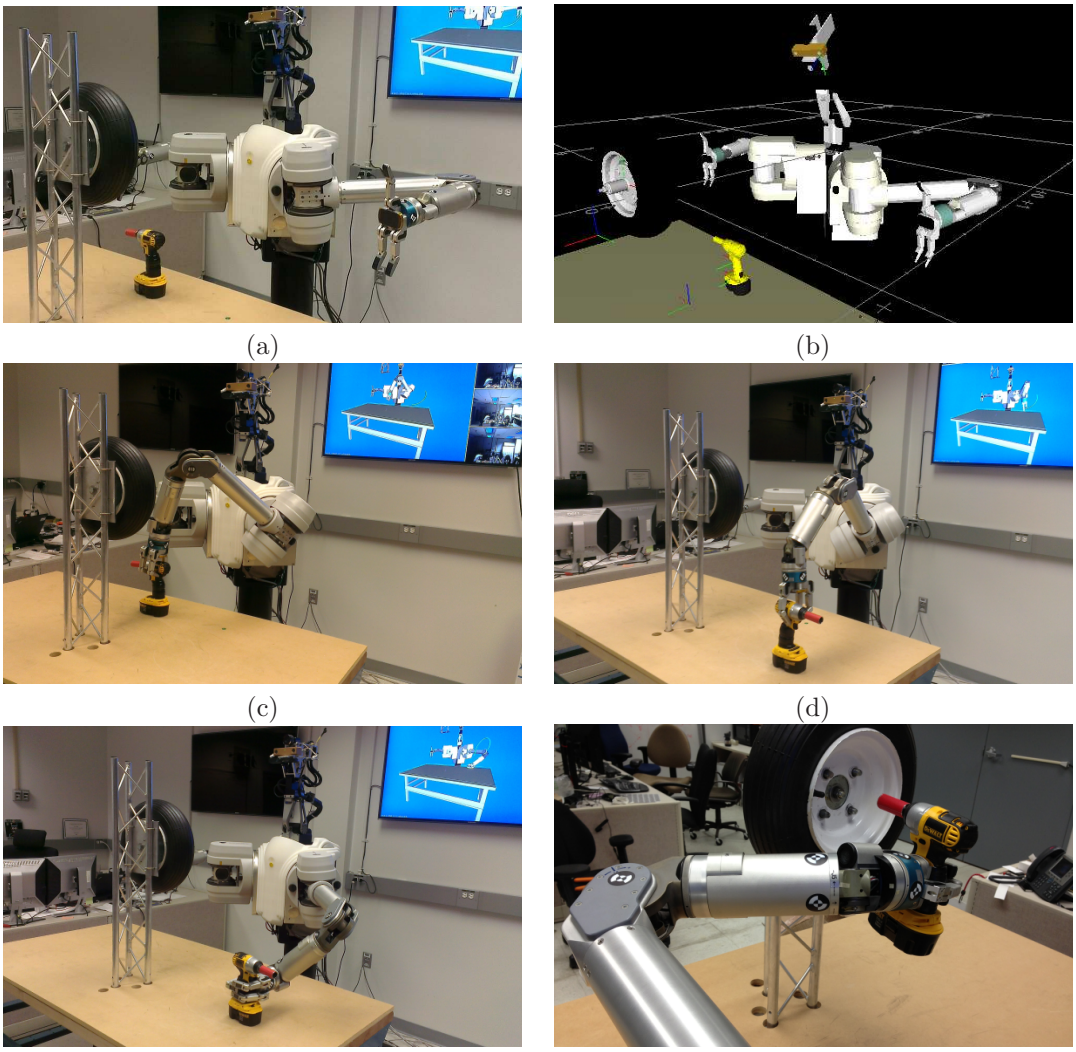
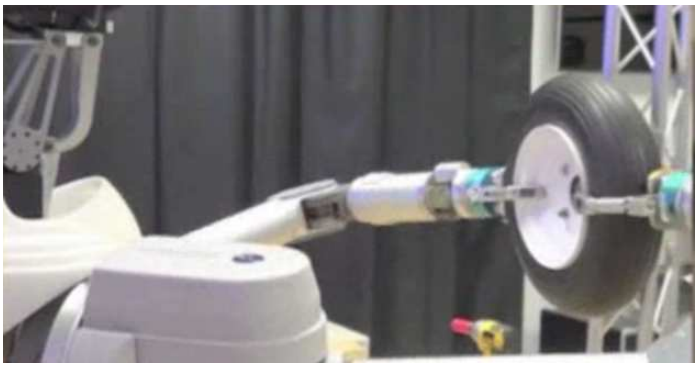


Figure 7.6: Execution for Example 2, Figure 7.5. (a) Initial world state. (b) Initial scan used to populate W^{model}. Kinematic verification reveals that the robot cannot grasp the impact for tool use in this configuration (the Pref. 1 path). (c) Robot picks up impact driver using power grasp (Pref. 2 path). (d) Impact is placed in a location which will allow feasible behavior for next step. (e) Robot is able to pick up impact with a tool use grasp. (f) Robot is able to plan to correct position for nut removal.

(a)



(b)

Figure 7.7: Probabilistic Outcomes and Partial Observability in ARM-S. (a) When the robot tries to remove the lug nut from the wheel-hub fixture, the action may not complete successfully, due to imperfect sense-act loop for nut removal. (b) Moreover, the on board sensors are unable to detect the successful removal of the lug-nuts. However if the robot attempts to remove the wheel off the hub, the success of wheel removal may be used to infer the state of the lug-nuts. This leads to partial observability at the task level.

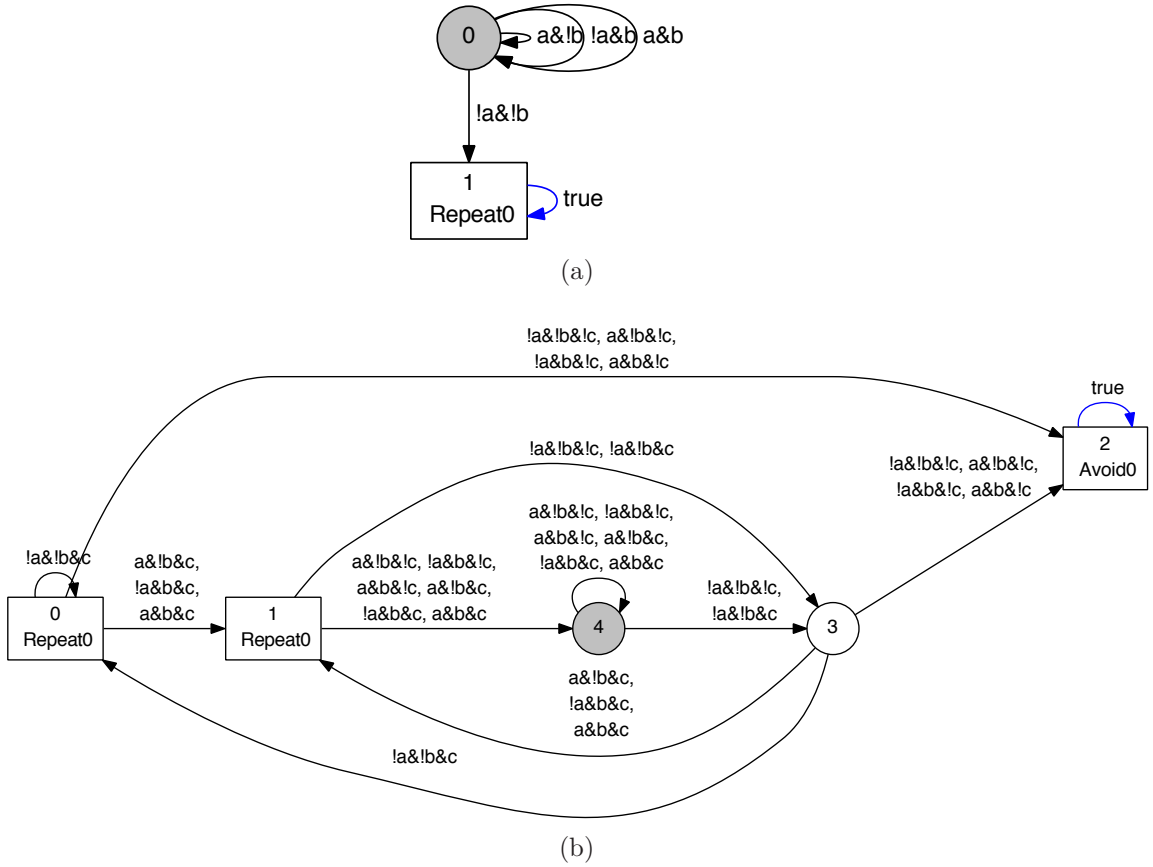


Figure 7.8: (a) DRA for ARM-S Task 1. The proposition a denotes $AttachedWheelToHub$, and b denotes $GraspedWheel$. The DRA has no *Avoid* states. As soon as the wheel has been removed from the hub and is no longer grasped, the specification is met, because DRA state 1 (which belongs to $Repeat_0$) is reached and the DRA state remains here for all future time steps. (b) DRA for ARM-S Task 2. The propositions a and b are the same as for Task 1. Proposition c denotes $AvailableSignal$. The start state is given by the grey state 0. One part of the LTL formula is explained using the DRA here. Consider that at the task beginning, the wheel is attached, but it is not grasped. The DRA state remains in 4 until both the wheel is no longer detached and is no longer grasped by the robot, at which point it transitions to state 3. Now, the specification requires that when this happens, in the very next step the $AvailableSignal$ must be true. If this fails to happen, then the DRA transitions to state 2, which is an *Avoid* state and is trapped there forever, thus indicating that the specification has been violated. In fact, the system cannot recover from this violation. If the robot does manage to turn the AvailableSignal on in the very next step, then the transition to a *Repeat* state numbered 0 occurs. This explains the part of φ_2^{ARM-S} given in line 2 of Equation 7.4. Other parts of the formula can be understood by following the DRA graph similarly.

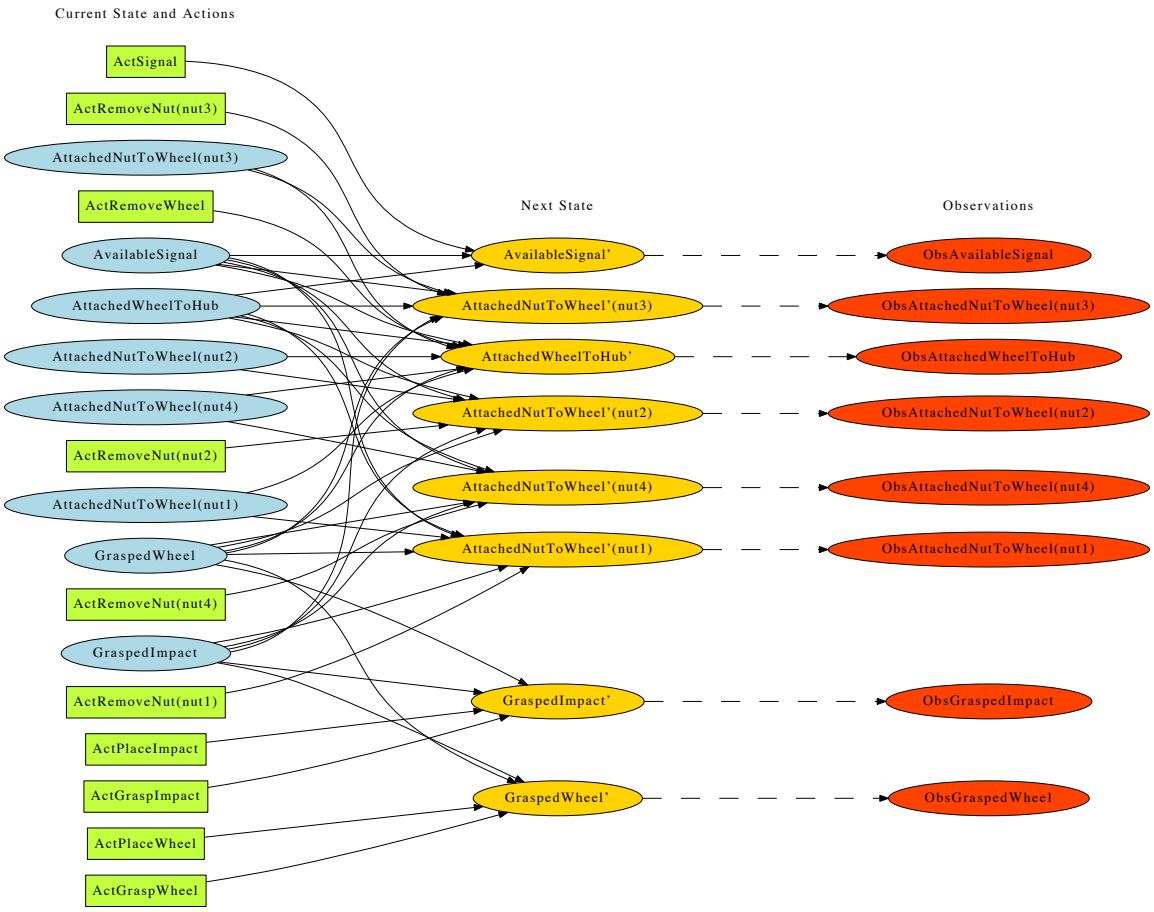


Figure 7.9: DBN of the ARM-S task planning domain and instance of Listing 7.1. The left most row of entities comprise of all the state variables and actions. The middle and right most row of entities show the state variables and observations at the next time step. The arrows indicate which entitites (current state variables / action) influence the variables at the next state (next state variables / observations). Note that DBN's only indicate the influence relationship between time steps and not the actual transition rules based on evaluations of the variables.

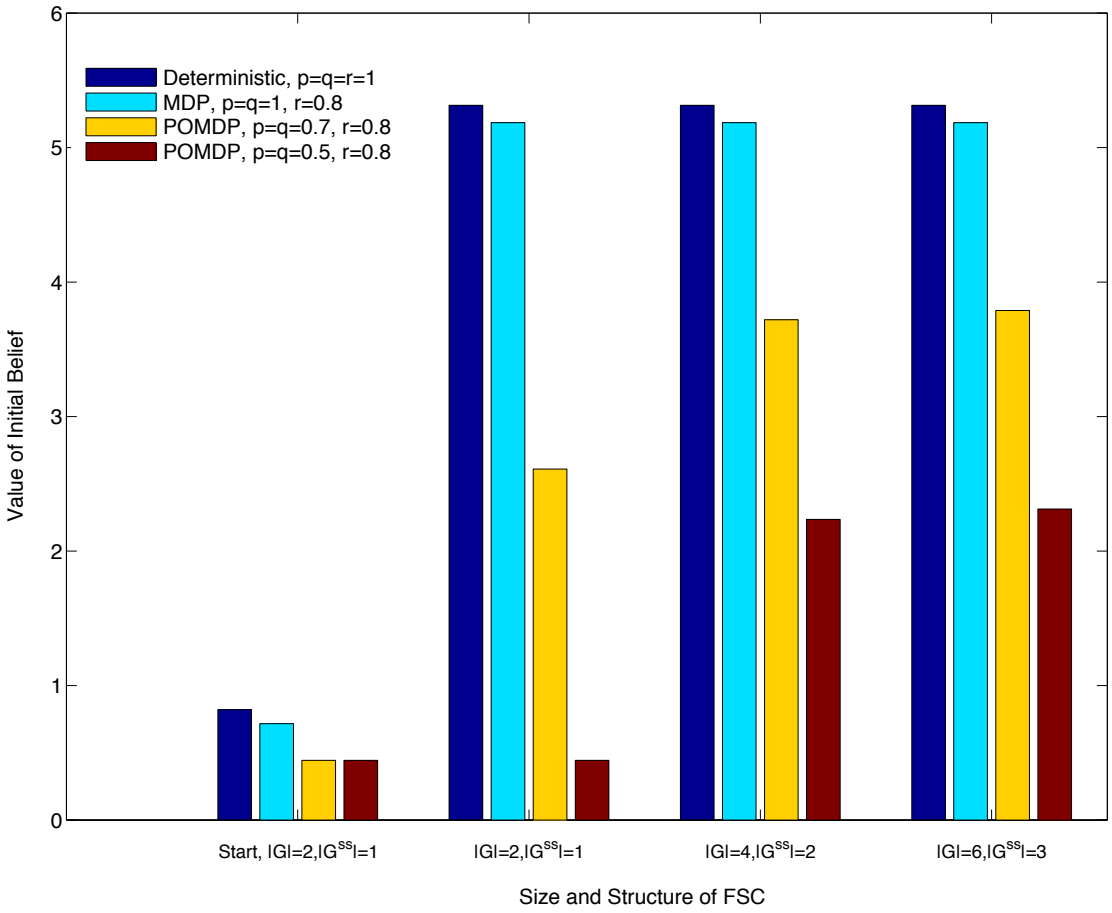


Figure 7.10: Bounded Policy Iteration for ARM Task 1. The results are shown for the case when the ARM-S planning domain is deterministic, probabilistic but fully observable (MDP), and partially observable (POMDP). As would be reasonable to expect, the deterministic case yields the best value for the initial belief, indicating that the reachability goal can be achieved quickly. The value of the initial belief denotes the expected long term discounted reward for the given initial belief. the MDP case is slightly worse, because of probabilistic effects. The adverse affect in performance for the POMDP case can also be seen, with the lowest values found for the case where the lug nut state measurement provides no information, i.e., $p = q = 0.5$.

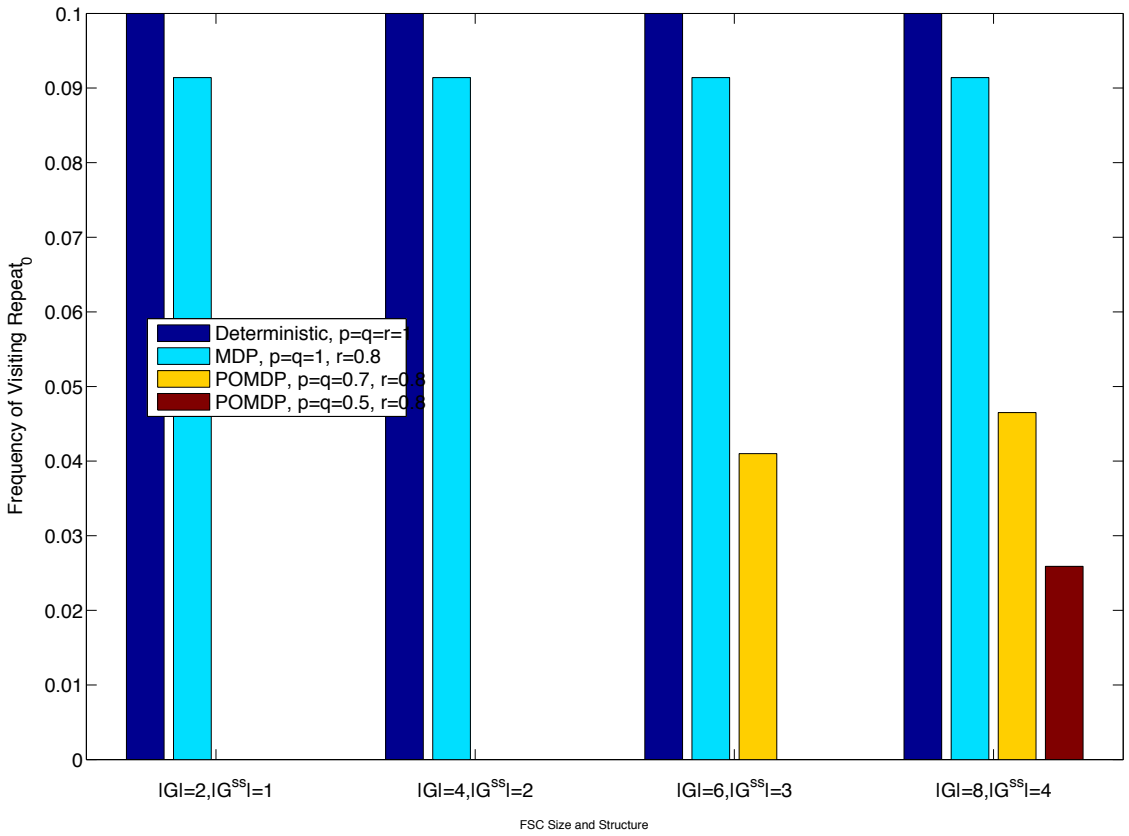


Figure 7.11: Bounded Policy Iteration for ARM Task 2. The expected frequency of completing wheel removal and turning on the AvailableSignal is shown.

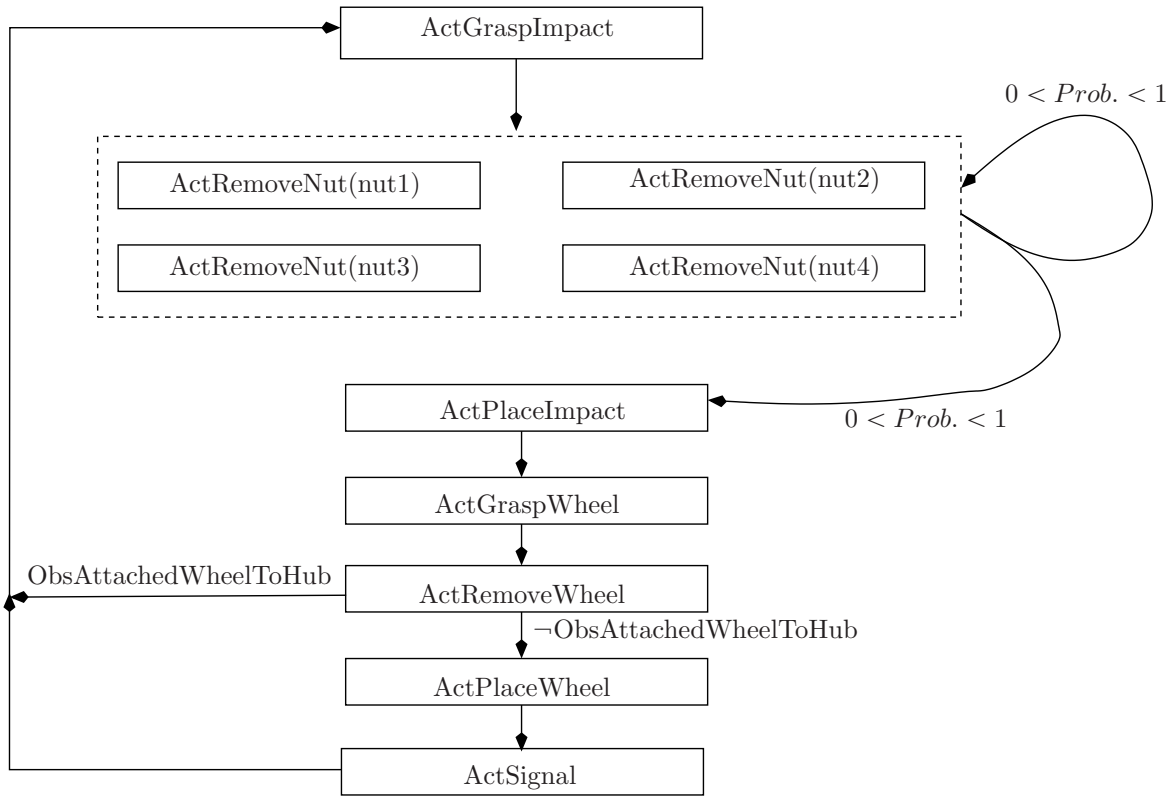


Figure 7.12: ARM-S Task 2 Policy. Qualitative behavior of the policy was inspected when the algorithm was stopped at $|G| = 8$, with $|G^{ss}| = 4.$, in order to determine the sequence of actions resulting from consolidating the FSC I-state transitions and system observations. The sequence of actions that remove the four lug-nuts (dashed box) are probabilistically determined and included repeated trials to remove the same lug-nut. The choice to proceed forward by placing the impact driver was also probabilisitic. Moreover, it was found that a $True$ value of the observation $ObsAttachedWheelToHub$ deterministically causes the FSC to restart the nut removal sequences as would be expected, whereas a $False$ value of the same observation allows the task sequence to proceed forward. Moreover, the model is such that whenever the action $ActSignal$ is carried out, the state variable $AvailableSignal$ first changes to $True$. At this step the FSC issues $ActIdle$. In the next step, the model is then guaranteed to produce a new hub-wheel attachment at which point the action sequence repeats.

Chapter 8

Conclusion and Future Directions

8.1 Summary

The primary contribution of this thesis is the development of approximate algorithms for optimizing system performance with respect to linear temporal logic specifications in partially observed settings. As was demonstrated in the case study of the DARPA ARM-S challenge, partial observability is a grave challenge faced by autonomous robotic systems. Partial observability also comes into play in complex and high security systems such as aircraft, in which silent sensor failure or tampering can compromise safe operation.

The thesis relies on the assumption that the environment and the controlled system can be abstracted into a finite Partially Observable Markov Decision Process. This model class has is known to be very general. In addition, most of the software written for autonomous systems typically results in a two layer abstraction naturally. First, a set of low level control modules that carry out a single objective, such as motion in free configuration space. Each of these low level modules have either clear pre- and post-conditions that involve higher discrete concepts involving the environment, the autonomous system, or is based on time-intervals. Second, the software encodes complex logic to invoke one or more of these controllers to accomplish its goals. Typically, partial observation is incorporated well into the set of low level controller modules via environmental disturbance and sensor noise models. Using the DARPA ARM-S challenge, the thesis emphasized the need to explicitly encode and incorporate partial observability at a higher discrete level as well, taking a model based approach.

The thesis presented approximate methods to address quantitative aspects of LTL satisfaction over POMDPs. It restricts the search for POMDP controllers to Finite State Controllers, which allows the analysis to be carried out over a finite state space Markov chain. This circumvented the difficulty arising from infinite state space when belief state methods are used to find POMDP controllers.

Two different algorithms to search for FSCs were shown. The first algorithm, in Chapter 4,

utilized a structure preserving parametrization for the FSC. Therein, it was shown how gradient based optimization could be utilized for maximizing the probability of LTL satisfaction. While the size and structure of the FSC itself did not lend itself to systematic exploration, and the nonlinear optimization suffered from local maxima, the algorithm demonstrates a novel gradient based method for LTL satisfaction over POMDPs.

The second algorithm was covered in Chapters 5 and 6. Designing a reward scheme which can be used to compute the probability of LTL satisfaction is still an open problem. However, in Chapter 5, it was shown how two different global state space reward schemes could be utilized to compute the two conditions necessary for LTL satisfaction – visiting particular states quickly and frequently, especially in steady state, and guaranteed transience of certain other states. A novel partitioning of the FSC internal states was then introduced, which allowed these two conditions to be expressed as an optimization objective and a constraint on the optimization problem respectively.

In Chapter 6, the multi-chain Poisson Equation was used in a novel way to allow for explicit expression of the constrained optimization of the preceding chapter. While the Poisson Equation has been used in dynamic programming for optimizing the average reward criterion, this thesis uses this fundamental equation in a novel way to ensure LTL satisfaction. In addition, this chapter also showed how a recent cutting edge algorithm for POMDP reward maximization, namely the Stochastic Bounded Policy Iteration, can be adapted in a novel way to search over stochastic FSCs to optimize long term discounted reward under additional constraints. While still suffering from local maxima, this algorithm allows for escaping them by providing a method to add more internal states to the FSC in a controlled, tunable manner.

Two main challenges were faced when these algorithms were applied to the practical case studies from the DARPA ARM-S challenge. The first is that both algorithms require an initial feasible controller. In the gradient based algorithm, this can be especially difficult for large systems and those that require long sequences of actions and observations to distinguish the unsafe states from the rest of the state space. In Chapter 6 the problem was partly mitigated by formulating an optimization problem that can be solved using dynamic programming.

The second challenge was the presence of bilinear constraints in the policy iteration methodology of Chapter 6. This required a relaxation based optimization algorithm that make the improvement step sub-optimal. There are two popular ways to solve general bilinear problems in the literature. First, is using an SDP relaxation which admits global nonlinear optimization methods such as branch and bound. However during practical implementation the off the shelf MATLAB tool in [89] was found to run out of memory for the ARM-S case studies. This motivated the use of McCormick envelopes directly to solve a linear program relaxation of the optimization problem.

The two algorithms presented in this work can be viewed as 'anytime' algorithms – computation time and memory constraints can be used to halt the optimization procedure for the best controller

at any iteration. This can be useful in practical applications in which time and memory constraints are the bottle neck, rather than the need for the optimal solution.

8.2 Open Issues and Future Work

While preceding chapters have provided novel contributions to verification based control synthesis in partially known environments, there are several open issues and opportunities for future work.

One algorithmic challenge is to circumvent the extra computational complexity due to the bi-linear constraints. We note that the objective is a linear function of FSC model parameters and is independent of solution $(\vec{\mathfrak{g}}, \vec{V}^{av})$ of the Poisson equation. It may be worthwhile to compare the sub-optimality using different algorithms for the convex relaxation such as barrier and interior point methods. Moreover, since bilinearity implies that the feasible set is convex in each variable when other variables are held constant, it may be possible to devise an alternating optimization scheme for the Poisson Equation similar to the policy iteration technique itself [17–19].

First, there are several challenges that must be addressed to enable the algorithm to be applied to large state space systems. It is worth noting that for the algorithms proposed in this thesis, a large state space can result from the Deterministic Rabin Automata (DRA) generated from the LTL formula, or from the underlying POMDP model of the controlled system. For the first source of complexity, it is well known that, in the worst case, the number of states of the DRA generated from an LTL formula φ can be doubly exponential in the number of variables that appear in φ. However empirical studies [74] have shown that many practical applications admit exponential complexity. In addition, polynomial complexity has been shown for an expressive fragment of LTL synthesis [114].

However, the key area that is open to further improvement is relaxing the assumptions on the model in Section 2.2, which can allow wider application of the algorithm. There has been a recent surge in efficient algorithms for factored (PO)MDPs [20,52]. In factored models, the state transition probabilities and rewards are defined using a Dynamic Bayesian Network that was used to encode the ARM-S case studies in Chapter 7. Factored models exploit the fact that the transition of a particular state variable only depends on a few other variables. In addition, the rewards may also depend on a subset of variables. The algorithms for these factored representation compute the control policy via Value or Policy Iteration techniques directly in the factored space instead of enumerating the entire state space for the computation of the value function. This is carried out by approximation techniques - the value function is approximated as a linear combination of possibly nonlinear basis functions [51]. These methods have also been applied to models with mixed observability in which only a subset of the state variable are partially observed, while the rest are known accurately [50].

Another key advancement can be in the context of Reinforcement Learning in which the agent or robot can only gather information regarding the system by exploring it. This situation can be

encountered due to imperfect abstraction. It is possible that while the discrete state space dynamical model is known, the proposition labeling of the state space is unknown until the agent reaches a particular state and can query the environment itself, or a black box model of the environment. This can be seen as a multi-valued extension of the bandit problem popular in Reinforcement Learning literature [69].

Both in the context of Reinforcement Learning, and for large state space models, sampling and point based approaches are a crucial methods for (PO)MDP policy optimization. These algorithms are especially challenging for formal verification of LTL specification, because properties such as safety and repeated reachability must be guaranteed over a given path with probability 1. It is noteworthy that the latest point based techniques in POMDP policy search algorithms for reward maximization can be applied to hundreds of thousands of states [138]. Point based methods accomplish this by restricting the search to only those belief states that are reachable from the initial distribution of states. However sampling based approaches need to be extended to ensure pointwise satisfaction of the Poisson equation in the Conservative Optimization Criterion and not just at the sampled beliefs.

A crucial aspect of abstraction of physical or cyber-physical systems is modeling error. Some work has been done in the control community to overcome the challenges of imperfect model abstraction by using robust verification based control synthesis in fully observable scenarios [122, 144]. These aspects are also crucial in partially observable models. Sensors may lose calibration during operation, thus requiring robustness in the observation model as well.

Two more areas of research that can also yield fruitful results are partially observable concurrent systems and distributed systems. Concurrent systems were the key motivation behind the development of temporal logic based verification methods. Distributed systems with concurrency appear ubiquitously in wide array of applications such as performance and safety guarantees in sensor networks, fault detection and repair in a distributed system and swarm robotics [23, 31, 90].

Appendices

Appendix A

Basic Measure Theory

In order to reason about probability of events, a brief overview of measurable spaces and probability measure is provided below. Special focus is on countable sets as they form the basis for this thesis.

A.1 σ-Algebra

Let X be an arbitrary non-empty set, and 2^X represent its power set. Next, let $\mathcal{F} = \{A_1, A_2 \dots\}$ be a family of subsets of X, i.e., $\mathcal{F} \subseteq 2^X$. $\mathcal{F}$ is called a $\sigma-$algebra if it satisfies the following properties

- $\varnothing \subseteq \mathcal{F}$.
- If a set A is in $\mathcal{F}$, then its complement, denoted $\tilde{A}$ is in $\mathcal{F}$. That is,

$$A \in \mathcal{F} \implies \tilde{A} \in \mathcal{F}. \tag{A.1}$$

- $\mathcal{F}$ is closed under countable unions. That is, for A_1,A_2,$\dots \in \mathcal{F}$, then

$$A_1 \cup A_2 \cup \cdots \in \mathcal{F}. \tag{A.2}$$

As a corollary of the above definition, the underlying set X is in $\mathcal{F}$.

A.2 Measurable Space and Measure

Elements $A_1, A_2, \dots$ of a $\sigma-$algebra $\mathcal{F}$ are called measurable sets. The ordered pair $(X, \mathcal{F})$ with $\mathcal{F}$ is $\sigma-$algebra over X is called a measurable space.

Let $\mu : \mathcal{F} \to [0, \infty)$ be a non negative bounded function with the following properties

- $\mu(\varnothing) = 0$.
- $\mu(A_i \cup A_j) + \mu(A_i \cap A_j) = \mu(A_i) + \mu(A_j)$, for all $A_i, A_j \in \mathcal{F}$.

- $\mu(\bigcup_{n=1} NA_n) = \sum_{n=1} N\mu(A_n)$ for pairwise disjoint sets $A_n \in \mathcal{F}$.

Such a function is called a non-negative additive set function. If the last property is extended to infinite disjoint unions, i.e.,

$$\mu(\bigcup_{n=1} \infty A_n) = \sum_{n=1} \infty\mu(A_n) \text{ for pairwise disjoint sets } A_n \in \mathcal{F} \tag{A.3}$$

then, μ is called a non-negative completely additive set function and is called a *measure*. The ordered tuple $(X, \mathcal{F}, \mu)$ is called a *measure space*.

A.3 Probability Space and Measure

Let $(X, \mathcal{F}, \mu)$ be a measure space as defined in the previous section. If the total measure of the underlying set given by $\mu(X)$ equals 1, then μ is called a *probability measure*. and $(X, \mathcal{F}, \mu)$ is a probability space.

A.4 Natural σ-Algebra and Distributions over Countable Sets

For a countable set X, if $\mathcal{F} = 2^X$, then we say that $\mathcal{F}$ is the *natural* σ-algebra. Next, let $\Pr : X \to [0, 1]$ be a function such that

$$\sum_{x \in X} \Pr(x) = 1 \tag{A.4}$$

Then Pr induces a unique probability measure $\mu_{\Pr}$ over the natural $\sigma-$algebra. Let $E \in 2^X$. Recall that $E \subseteq X$. Define the measure on E as follows

$$\mu_{\Pr}(E) = \sum_{x \in E} \Pr(x) \tag{A.5}$$

It is easy to see that $\mu_{\Pr}$ satisfies all properties of a (probability) measure. In addition by $\Pr(\varnothing) = 0$. Any function $\Pr : X \to [0, 1]$ over countable sets X that satisfy equation A.4 will be called *distributions* over X. The set of all distributions over the set X are denoted M_X throughout this thesis.

In the above context, the set X is often called a set of outcomes of an experiment. The sets in $\mathcal{F} = 2^X$ are called *events*. This is made concrete in the following coin toss example.

Consider the experiment of the side facing up after tossing a coin. The underlying set of outcomes is given by the set $X = heads, tails$. The natural $\sigma-$algebra is the collection

$$A_1 = \varnothing, \quad A_2 = \{heads\}, \quad A_3 = \{tails\}, \quad A_4 = \{heads, tails\}. \tag{A.6}$$

A_1 represents that no side of the coin faces up. A_2 denotes *heads* faces up, A_3 denotes *tails* faces up, while A_4 denotes *heads or tails* face up. For a fair coin the distribution or probability measure is given by

$$\Pr(A_1) = 0, \Pr(A_2) = \frac{1}{2}, \Pr(A_3) = \frac{1}{2}, \Pr(A_4) = 1. \tag{A.7}$$

A.5 Smallest $\sigma-$algebra and Basis Events

Let the underlying set be denoted X. Let $\mathcal{F}_1, \mathcal{F}_2 \subseteq 2^X$ denote two $\sigma-$algebras over X. The following is a well known fact [130].

Lemma A.5.1 *$\mathcal{F}_1 \cap \mathcal{F}_2$ is also a $\sigma-$algebra.*

This property holds for countably infinite intersections as well.

Let $\mathcal{T} \subset 2^X$ not necessarily a $\sigma-$algebra. Then there exists a *smallest* $\sigma-$algebra that contains $\mathcal{T}$. It is given by

$$\sigma(\mathcal{T}) = \bigcap \{\mathcal{F} \in 2^X : \mathcal{F} \text{ is a } \sigma - \text{algebra over } X \text{ and } \mathcal{T} \subseteq \mathcal{F}\}, \tag{A.8}$$

where the intersection ranges over all (possibly infinite number of) $\sigma-$ algebras over the countable set X. Additionally, $\sigma(\mathcal{T})$ is said to be *generated* by $\mathcal{T}$. Conversely, $\mathcal{T}$ is called the *basis* for $\sigma(\mathcal{T})$.

Appendix B

Proofs

B.1 Proof of Lemma 5.1.1

Consider a finite path fragment $\pi = s_0 s_1 \dots$ in each of the two Markov chains given by T^{φ} and T^{φ}_{mod} respectively. Consider the event of visiting a state in $Avoid^{\varphi}_{\mathfrak{r}}$ for the first time at the k-th time step. A path that satisfies this can be written as

$$\pi_k = s_0 s_1 \dots s_k \dots \text{ s.t. } s_0 \dots s_{k-1} \notin Avoid^{\varphi}_{\mathfrak{r}} \text{ and } s_k \in Avoid^{\varphi}_{\mathfrak{r}} \tag{B.1}$$

Then, from the definition of the probability measure of cylinder sets in Equation (2.11), the probability measures of the cylinder sets under the two Markov chains are identical:

$$\begin{aligned} \Pr{}_{\mathcal{M}}\left[Cyl_{\mathcal{M}}(\pi_k)\,\middle|\,\iota^{\varphi,\mathcal{G}}_{init}\right] &= \iota^{ss}_{init}(s_0)\Pi_{t=1}^{k} T^{\varphi}(s_t|s_{t-1}) \\ &= \iota^{ss}_{init}(s_0)\Pi_{t=1}^{k} T^{\varphi}_{mod}(s_t|s_{t-1}) \\ &= \Pr{}_{\mathcal{M}_{mod}}\left[Cyl_{\mathcal{M}_{mod}}(\pi_k)\,\middle|\,\iota^{\varphi,\mathcal{G}}_{init}\right] \end{aligned} \tag{B.2}$$

where $Cyl_{\mathcal{M}} \in Paths(\mathcal{M})$ and $Cyl_{\mathcal{M}_{mod}} \in Paths(\mathcal{M}_{mod})$. The equality of lines 1 and 2 follows from the fact that $T^{\varphi}_{mod}(s_j|s_i) = T^{\varphi}(s_j|s_i)$, $\forall s_i \notin Avoid^{\varphi}_{\mathfrak{r}}$ from Equation (5.3).

Next, note that the probability of paths visiting $Avoid^{\varphi}_{\mathfrak{r}}$ in the l.h.s. of the lemma is given by

$$\begin{aligned} \Pr\left[\pi \to (Avoid^{\mathcal{P}\mathcal{M}^{\varphi}}_{\mathfrak{r}} \times G)\,\middle|\,\iota^{\varphi,\mathcal{G}}_{init}\right] &= \sum_{k=0}^{\infty} \Pr{}_{\mathcal{M}}\left[Cyl_{\mathcal{M}}(\pi_k)\,\middle|\,\iota^{\varphi,\mathcal{G}}_{init}\right] \\ &= \sum_{k=0}^{\infty} \Pr{}_{\mathcal{M}_{mod}}\left[Cyl_{\mathcal{M}_{mod}}(\pi_k)\,\middle|\,\iota^{\varphi,\mathcal{G}}_{init}\right] \end{aligned} \tag{B.3}$$

In addition, since each state in $Avoid^{\varphi}_{\mathfrak{r}}$ is absorbing under T^{φ}_{mod} and has a reward 1 under the scheme of Equation (5.4), for a *given* infinite path π of $\mathcal{M}_{mod}$, the long term average sum of rewards can be seen to be

$$Rew(\pi) = \lim_{t\to\infty} \frac{1}{T}\left[\sum_{t=0}^{T} r^{av}_{\mathfrak{k}}(s_t)\,\middle|\,\iota^{\mathcal{P}\mathcal{M}^{\varphi}}_{init}\right] = \begin{cases} 1 & \text{if } \pi \to Avoid^{\varphi}_{\mathfrak{r}} \\ 0 & \text{otherwise .} \end{cases} \tag{B.4}$$

This happens because if a path visits any state $Avoid^{\varphi}_{\mathfrak{r}}$ it forever remains in that state accumulating a reward of 1 at each time step. In the limit as time steps grow to infinity, the average reward per step converges to 1.

Finally, taking the expectation of the function $Rew(\pi)$ gives

$$\begin{aligned}\eta_{av}(\mathfrak{r}) &= \mathbb{E}_{mod}\left[Rew(pi)\right] \\ &= 1.\Pr_{\mathcal{M}_{mod}}\left[\pi \rightarrow Avoid^{\varphi}_{\mathfrak{r}} \,\middle|\, \iota^{\varphi,\mathcal{G}}_{init}\right] + 0.\Pr_{\mathcal{M}_{mod}}\left[\pi \not\rightarrow Avoid^{\varphi}_{\mathfrak{r}} \,\middle|\, \iota^{\varphi,\mathcal{G}}_{init}\right] \\ &= \sum_{k=0}^{\infty} \Pr_{\mathcal{M}_{mod}}\left[Cyl_{\mathcal{M}_{mod}}(\pi_k) \,\middle|\, \iota^{\varphi,\mathcal{G}}_{init}\right],\end{aligned} \tag{B.5}$$

which proves the lemma.

B.2 Proof Sketch of Proposition 4.2.2

The reader is reminded of the definition of the relation given by $\preceq$.

$$[s,g] \preceq \left[s',g'\right] \text{ if } \begin{cases} [s,g],[s',g'] \in R_k \text{ and } [s,g] \preceq_k [s',g'], & \text{or,} \\ [s,g] \in R_k, [s',g'] \in R_l,\ k \neq l, \text{ and } R_k \preceq' R_l, & \text{or,} \\ [s,g] \in R_k \subseteq RecSets^{\mathcal{G}} \text{ and } [s',g'] \in T, & \text{or,} \\ [s,g],[s',g'] \in \mathcal{T} \text{ and } [s,g] \preceq_T [s',g']. \end{cases}, \tag{B.6}$$

where, $\preceq_k$ is a total order over the set of global states in R_k, $\preceq'$ is a total order over the recurrent subsets $R_1, R_2, \cdots \subseteq \mathcal{S} \times G$, and $\preceq_T$ is a total order over the set of global states in transient set $\mathcal{T} \subseteq \mathcal{S} \times G$.

In order for the relation $\preceq$ to be a total order, it must be shown that $\preceq$ satisfies the properties of

1. **Anti-symmetry:** This means that if $[s,g] \preceq [s',g']$ and $[s',g'] \preceq [s,g]$ then $[s,g] = [s',g']$.
2. **Transitivity:** This means that if $[s,g] \preceq [s',g']$ and $[s',g'] \preceq [s'',g'']$, then $[s,g] \preceq [s'',g'']$.
3. **Totatility:** This means that either $[s,g] \preceq [s',g']$ or $[s',g'] \preceq [s,g]$.

First, if both $[s,g],[s',g'] \in R_k$ for some k, or both $[s,g],[s',g'] \in \mathcal{T}$, then the relation $\preceq$ is identical to $\preceq_k$ or $\preceq_{\mathcal{T}}$ respectively. Since $\preceq_k$ and $\preceq_{\mathcal{T}}$ are total orders, $\preceq$ defines a total order *within* each $R_1, R_2, \ldots$ and $\mathcal{T}$.

Next, note that $R_1, R_2, \ldots$ and $\mathcal{T}$ are all disjoint and their union gives the entire state space $\mathcal{S} \times G$. Thus $R_1, R_2, \ldots$ and $\mathcal{T}$ together partition the state space, and each will be called a *disjoint component* in this section. From the assumptions of the proposition, $\preceq'$ defines a total order over the subsets $R_1, R_2, \ldots$. Line 3 in Equation (B.6) makes sure that $\mathcal{T}$ comes *after* or is *greater* than all recurrent sets R_k, thus giving a total order over the disjoint components. For the case when

$[s, g]$ and $[s', g']$ belong to different disjoint components, anti-symmetry is inapplicable, and totality follows from the total order over the disjoint components. For transitivity, it can be seen that only three cases are possible:

(a) $[s, g]$, $[s', g']$, $[s'', g'']$ all belong to the same disjoint component R_k for some k, or $\mathcal{T}$, in which case $\preceq_k$ or $\preceq_{\mathcal{T}}$ guarantee transitivity.

(b) $[s, g]$ and $[s', g']$ belong to the same disjoint component, whereas $[s'', g'']$ belongs to a different disjoint component, in which case the disjoint component total order described earlier and totality property together guarantee transitivity.

(c) $[s, g]$ belongs to one disjoint component, $[s', g']$ and $[s'', g'']$ belong to a different disjoint component, in which case the transitivity arises to due the total order over the disjoint components and the totality property.

Bibliography

[1] Douglas Aberdeen. *Policy-Gradient Algorithms for Partially Observable Markov Decision Processes*. PhD thesis, 2003.

[2] Rajeev Alur. Formal verification of hybrid systems. In *Embedded Software (EMSOFT), 2011 Proceedings of the International Conference on*, pages 273–278. IEEE, 2011.

[3] DARPA ARM. http://thearmrobot.com/aboutrobot.html.

[4] Mikael Asplund, Atif Manzoor, Mlanie Bouroche, Siobhn Clarke, and Vinny Cahill. A formal approach to autonomous vehicle coordination. In *FM 2012: Formal Methods*, volume 7436 of *Lecture Notes in Computer Science*, pages 52–67. Springer Berlin Heidelberg, 2012.

[5] M. Athans. The role and use of the stochastic linear-quadratic-gaussian problem in control system design. *Automatic Control, IEEE Transactions on*, 16(6):529–552, 1971.

[6] Atlas. The agile anthropomorphic robot, http://www.bostondynamics.com/robot_atlas.html. *Boston Dynamics*.

[7] R Iris Bahar, Erica A Frohm, Charles M Gaona, Gary D Hachtel, Enrico Macii, Abelardo Pardo, and Fabio Somenzi. Algebric decision diagrams and their applications. *Formal methods in system design*, 10(2-3):171–206, 1997.

[8] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking (Representation and Mind Series)*. The MIT Press, 2008.

[9] Baxter. http://www.rethinkrobotics.com/products/baxter/. *Rethink Robotics*.

[10] Jonathan Baxter and Peter L. Bartlett. Infinite-horizon policy-gradient estimation. *Journal of Artificial Intelligence Research*, 2001.

[11] Richard Bellman. On the theory of dynamic programming. *Proceedings of the National Academy of Sciences of the United States of America*, 38(8):716, 1952.

[12] Richard Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA, 1st edition, 1957.

[13] Richard Bellman. A markovian decision process. *Indiana Univ. Math. J.*, 6:679–684, 1957.

[14] Calin Belta, Volkan Isler, and George J Pappas. Discrete abstractions for robot motion planning and control in polygonal environments. *Robotics, IEEE Transactions on*, 21(5):864–874, 2005.

[15] Dimitri P Bertsekas. *Dynamic programming and stochastic control*. Number 10. Academic Press, 1976.

[16] Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control, Two Volume Set*. Athena Scientific, 2nd edition, 2001.

[17] James C. Bezdek and Richard J. Hathaway. Partitioning the variables for alternating optimization of real-valued scalar fields. In *PRIS*, pages 1–19, 2002.

[18] James C. Bezdek and Richard J. Hathaway. Some notes on alternating optimization. In *Advances in Soft Computing AFSS 2002*, volume 2275 of *Lecture Notes in Computer Science*, pages 288–300. Springer Berlin Heidelberg, 2002.

[19] James C. Bezdek and Richard J. Hathaway. Convergence of alternating optimization. *Neural Parallel & Scientific Comp.*, 11(4):351–368, 2003.

[20] Craig Boutilier, Richard Dearden, and Moisés Goldszmidt. Stochastic dynamic programming with factored representations. *Artificial Intelligence*, 121(1):49–107, 2000.

[21] Patricia Bouyer. Model-checking timed temporal logics. *Electronic Notes in Theoretical Computer Science*, 231:323–341, 2009.

[22] Ronen I. Brafman. A heuristic variable grid solution method for pomdps. In *AAAI/IAAI*, pages 727–733, 1997.

[23] Manuele Brambilla, Eliseo Ferrante, Mauro Birattari, and Marco Dorigo. Swarm robotics: a review from the swarm engineering perspective. *Swarm Intelligence*, 7(1):1–41, 2013.

[24] Darius Braziunas. Pomdp solution methods. Technical report, 2003.

[25] John S. Bridle. Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In *Neurocomputing*, volume 68 of *NATO ASI Series*, pages 227–236. Springer Berlin Heidelberg, 1990.

[26] Jerry R. Burch, Edmund M. Clarke, David E. Long, Kenneth L. McMillan, and David L. Dill. Symbolic model checking for sequential circuit verification. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 13(4):401–424, 1994.

[27] Samuel Burer and Adam N. Letchford. On nonconvex quadratic programming with box constraints. *SIAM Journal on Optimization*, 20(2):1073–1089, 2009.

[28] Wolfram Burgard, Cyrill Stachniss, Maren Bennewitz, Giorgio Grisetti, and Kai Arras. Introduction to mobile robotics, http://ais.informatik.uni-freiburg.de/teaching/ss10/robotics/slides/02-paradigms.pdf.

[29] Anthony R. Cassandra, Leslie Pack Kaelbling, and Michael L. Littman. Acting optimally in partially observable stochastic domains. In *AAAI*, pages 1023–1028, 1994.

[30] Serenella Cerrito and MartaCialdea Mayer. Using linear temporal logic to model and solve planning problems. In *Artificial Intelligence: Methodology, Systems, and Applications*, volume 1480 of *Lecture Notes in Computer Science*, pages 141–152. Springer Berlin Heidelberg, 1998.

[31] Thomas Chatain and Claude Jard. Symbolic diagnosis of partially observable concurrent systems. In *Formal Techniques for Networked and Distributed Systems–FORTE 2004*, pages 326–342. Springer, 2004.

[32] Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. Qualitative analysis of partially-observable markov decision processes. In *Mathematical Foundations of Computer Science 2010*, volume 6281 of *Lecture Notes in Computer Science*, pages 258–269. 2010.

[33] Krishnendu Chatterjee, Laurent Doyen, Sumit Nain, and Moshe Y. Vardi. The complexity of partial-observation stochastic parity games with finite memory strategies. Technical report, 2013.

[34] Sandeep Chinchali, Scott C. Livingston, Ufuk Topcu, Joel W. Burdick, and Richard M. Murray. Towards formal synthesis of reactive controllers for dexterous robotic manipulation. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 5183–5189. IEEE, 2012.

[35] Igor Cizelj and Calin Belta. Probabilistically safe control of noisy dubins vehicles. In *IROS*, pages 2857–2862, 2012.

[36] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Program. Lang. Syst.*, 8(2):244–263, April 1986.

[37] Edmund Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement for symbolic model checking. *Journal of the ACM (JACM)*, 50(5):752–794, 2003.

[38] Edmund M. Clarke, E. Allen Emerson, and A. Prasad Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 8(2):244–263, 1986.

[39] John J. Craig. *Introduction to Robotics: Mechanics and Control (3rd Edition)*. Prentice Hall, 3rd edition, 2004.

[40] Xu Chu Ding, Stephen L. Smith, Calin Belta, and Daniela Rus. Ltl control in uncertain environments with probabilistic satisfaction guarantees. *CoRR*, abs/1104.1159, 2011.

[41] Finale Doshi. The infinite partially observable markov decision process. In *Neural Information Processing Systems*, volume 22, pages 477–485, 2009.

[42] E. Allen Emerson. Temporal and modal logic. In *HANDBOOK OF THEORETICAL COMPUTER SCIENCE*, pages 995–1072. Elsevier, 1995.

[43] E. Allen Emerson and Edmund M. Clarke. Characterizing correctness properties of parallel programs using fixpoints. In *Automata, Languages and Programming*, volume 85 of *Lecture Notes in Computer Science*, pages 169–181. Springer Berlin Heidelberg, 1980.

[44] Richard E. Fikes and Nils J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3):189–208, 1972.

[45] Robert W. Floyd. Assigning meanings to programs. *Mathematical aspects of computer science*, 19(19-32):1, 1967.

[46] Dov M. Gabbay, C. J. Hogger, and J. A. Robinson, editors. *Handbook of Logic in Artificial Intelligence and Logic Programming (Vol. 4): Epistemic and Temporal Reasoning*. Oxford University Press, Oxford, UK, 1995.

[47] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated planning: theory & practice*. Access Online via Elsevier, 2004.

[48] Patrice Godefroid. *Partial-order methods for the verification of concurrent systems: an approach to the state-explosion problem*, volume 1032.

[49] Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research*, volume 2500 of *Lecture Notes in Computer Science*. Springer, 2002.

[50] Carlos Guestrin, Milos Hauskrecht, and Branislav Kveton. Solving factored mdps with continuous and discrete variables. In *Proceedings of the 20th conference on Uncertainty in artificial intelligence*, pages 235–242. AUAI Press, 2004.

[51] Carlos Guestrin, Daphne Koller, and Ronald Parr. Solving factored pomdps with linear value functions. Citeseer.

[52] Carlos Guestrin, Daphne Koller, Ronald Parr, and Shobha Venkataraman. Efficient solution algorithms for factored mdps. 2003.

[53] Eric A. Hansen. Solving POMDPs by Searching in Policy Space. In *Fourteenth International Conference on Uncertainty In Artificial Intelligence (UAI-98)*, pages 211–219, 1998.

[54] Eric A. Hansen. Sparse stochastic finite-state controllers for pomdps. 2008.

[55] Milos Hauskrecht. Incremental methods for computing bounds in partially observable markov decision processes. In *AAAI/IAAI*, pages 734–739, 1997.

[56] Onesimo Hernandez-Lerma and Jean Bernard Lasserre. Markov chains and invariant probabilities. In *Progress in mathematics*. Birkhauser Verlag, 2003.

[57] Onsimo Hernndez-Lerma. *Adaptive Markov control processes*. Applied mathematical sciences. Springer, New York, 1989.

[58] Charles Antony Richard Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580, 1969.

[59] Alexander Holt, Er Holt, Ewan Klein, and Claire Grover. Natural language for hardware verification: Semantic interpretation and model checking. In *ILLC, University of Amsterdam*, pages 133–137, 1999.

[60] Gerard J. Holzmann. The theory and practice of a formal method: Newcore. 1994.

[61] Gerard J. Holzmann. The model checker spin. *IEEE Transactions on Software Engineering*, 23:279–295, 1997.

[62] R.A. Howard. *Dynamic Programming and Markov Processes*. MIT Press, 1960.

[63] Nicolas Hudson, Thomas Howard, Jeremy Ma, Abhinandan Jain, Max Bajracharya, Steven Myint, Calvin Kuo, Larry Matthies, Paul Backes, Paul Hebert, et al. End-to-end dexterous manipulation with deliberate interactive estimation. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 2371–2378. IEEE, 2012.

[64] Nicolas Hudson, Jeremy Ma, Paul Hebert, Abhinandan Jain, Max Bajracharya, Thomas Allen, Rangoli Sharan, Matanya Horowitz, Calvin Kuo, Thomas Howard, Larry Matthies, Paul Backes, and Joel Burdick. Model-based autonomous system for performing dexterous, human-level manipulation tasks. *Autonomous Robots*, pages 1–19, 2013.

[65] Michael Huth and Mark Ryan. *Logic in Computer Science: Modelling and reasoning about systems*. Cambridge University Press, 2004.

[66] L.J. Jagadeesan, Carlos Puchol, and J.E. von Olnhausen. A formal approach to reactive systems software: a telecommunications application in esterel. In *Industrial-Strength Formal Specification Techniques, 1995. Proceedings., Workshop on*, pages 132–145, 1995.

[67] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 1998.

[68] Sertac Karaman and Emilio Frazzoli. Sampling-based motion planning with deterministic μ-calculus spefications. In *CDC*, pages 2222–2229, 2009.

[69] Michael N. Katehakis and Arthur F. Veinott. The multi-armed bandit problem: Decomposition and computation. *Mathematics of Operations Research*, 12(2):262–268, 1987.

[70] Lydia E. Kavraki, Petr Svestka, J-C Latombe, and Mark H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *Robotics and Automation, IEEE Transactions on*, 12(4):566–580, 1996.

[71] John G. Kemeny and J. Laurie Snell. *Finite Markov Chains*. Springer-Verlag, 1976.

[72] Joachim Klein. ltl2dstar - ltl to deterministic streett and rabin automata (www.ltl2dstar.de).

[73] Joachim Klein. *Linear time logic and deterministic omega-automata*. PhD thesis, 2005.

[74] Joachim Klein and Christel Baier. Experiments with deterministic omega;-automata for formulas of linear temporal logic. *Theoretical Computer Science*, 363:182–195, 2005.

[75] Marius Kloetzer and Calin Belta. A fully automated framework for control of linear systems from temporal logic specifications. *Automatic Control, IEEE Transactions on*, 53(1):287–297, 2008.

[76] Hadas Kress-Gazit, Georgios E. Fainekos, and George J. Pappas. Where's waldo? sensor-based temporal logic motion planning. In *ICRA*, pages 3116–3121, 2007.

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

[78] Hadas Kress-Gazit, Tichakorn Wongpiromsarn, and Ufuk Topcu. Correct, reactive, high-level robot control. *Robotics & Automation Magazine, IEEE*, 18(3):65–74, 2011.

[79] Hanna Kurniawati, David Hsu, and Wee Sun Lee. Sarsop: Efficient point-based pomdp planning by approximating optimally reachable belief spaces. In *In Proc. Robotics: Science and Systems*, 2008.

[80] K. G. Larsen and A. Skou. Bisimulation through probabilistic testing (preliminary report). In *Proceedings of the 16th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '89, pages 344–352, New York, NY, USA, 1989. ACM.

[81] J.B Lasserre. Conditions for existence of average and blackwell optimal stationary policies in denumerable markov decision processes. *Journal of Mathematical Analysis and Applications*, 136(2):479 – 489, 1988.

[82] Steven M LaValle. Rapidly-exploring random trees: A new tool for path planning.

[83] E. L. Lawler and D. E. Wood. Branch-and-bound methods: A survey. *Operations Research*, 14(4):699–719, 1966.

[84] Olivier Lebeltel, Pierre Bessière, Julien Diard, and Emmanuel Mazer. Bayesian robot programming. *Autonomous Robots*, 16(1):49–79, 2004.

[85] J.J. Leonard and H.F. Durrant-Whyte. Simultaneous map building and localization for an autonomous mobile robot. In *Intelligent Robots and Systems '91. 'Intelligence for Mechanical Systems, Proceedings IROS '91. IEEE/RSJ International Workshop on*, pages 1442–1447 vol.3, 1991.

[86] Feng Lin. Analysis and synthesis of discrete event systems using temporal logic. In *Intelligent Control, 1991., Proceedings of the 1991 IEEE International Symposium on*, pages 140–145. IEEE, 1991.

[87] Scott C. Livingston, Richard M. Murray, and Joel W. Burdick. Backtracking temporal logic synthesis for uncertain environments. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 5163–5170. IEEE, 2012.

[88] Scott C. Livingston, Pavithra Prabhakar, Alex B. Jose, and Richard M. Murray. Patching task-level robot controllers based on a local μ-calculus formula. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 4588–4595. IEEE, 2013.

[89] Johan Lofberg. Yalmip: A toolbox for modeling and optimization in matlab. In *Computer Aided Control Systems Design, 2004 IEEE International Symposium on*, pages 284–289. IEEE, 2004.

[90] Sarah M. Loos, David W. Renshaw, and André Platzer. Formal verification of distributed aircraft controllers. In *Hybrid Systems: Computation and Control (part of CPS Week 2013), HSCC'13, Philadelphia, PA, USA, April 8-13, 2013*, pages 125–130. ACM, 2013.

[91] Tomas Lozano-Perez. Spatial planning: A configuration space approach, 1980.

[92] Tomás Lozano-Pérez and Michael A. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM*, 22(10):560–570, 1979.

[93] Feng Lu and Evangelos Milios. Robot pose estimation in unknown environments by matching 2d range scans. *Journal of Intelligent and Robotic Systems*, 18(3):249–275, 1997.

[94] Armand M. Makowski and Adam Shwartz. On the poisson equation for markov chains: Existence of solutions and parameter dependence by probabilistic methods. Technical report, 1994.

[95] Matthew R. Maly, Morteza Lahijanian, Lydia E. Kavraki, Hadas Kress-Gazit, and Moshe Y. Vardi. Iterative temporal motion planning for hybrid systems in partially unknown environments. In *Proceedings of the 16th International Conference on Hybrid Systems: Computation and Control*, HSCC '13, pages 353–362, New York, NY, USA, 2013. ACM.

[96] Zohar Manna and Amir Pnueli. *The temporal logic of reactive and concurrent systems - specification*. Springer, 1992.

[97] Garth P. McCormick. Computability of global solutions to factorable nonconvex programs: Part iconvex underestimating problems. *Mathematical programming*, 10(1):147–175, 1976.

[98] P. McCullagh and J.A. Nelder. *Generalized Linear Models, Second Edition*. Chapman & Hall/CRC Monographs on Statistics & Applied Probability. Taylor & Francis, 1989.

[99] Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. Pddl-the planning domain definition language. 1998.

[100] Sean Meyn. The policy improvement algorithm: General theory with applications to queueing networks and their fluid models. In *35th IEEE Conference on Decision and Control, Kobe*, 1996.

[101] Sean Meyn and Richard L. Tweedie. *Markov Chains and Stochastic Stability*. Cambridge University Press, New York, NY, USA, 2nd edition, 2009.

[102] Kevin Patrick Murphy. *Dynamic bayesian networks: representation, inference and learning*. PhD thesis, University of California, 2002.

[103] Robin R. Murphy. *Introduction to AI Robotics*. MIT Press, Cambridge, MA, USA, 1st edition, 2000.

[104] Rani Nelken and Nissim Francez. Automatic translation of natural language system specifications into temporal logic. Technical report, 1996.

[105] Van-Duc Nguyen. Constructing force-closure grasps. *The International Journal of Robotics Research*, 7(3):3–16, 1988.

[106] Illah R. Nourbakhsh, Rob Powers, and Stan Birchfield. Dervish - an office-navigating robot. *AI Magazine*, 16(2):53–60, 1995.

[107] Sylvie CW Ong, Shao Wei Png, David Hsu, and Wee Sun Lee. Planning under uncertainty for robotic tasks with mixed observability. *The International Journal of Robotics Research*, 29(8):1053–1068, 2010.

[108] Donald Ornstein. On the existence of stationary optimal strategies. *Proceedings of the American Mathematical Society*, 20(2):pp. 563–569, 1969.

[109] Weiwei Pan. Paradigms of A.I. robotics, http://ftp.stmarys-ca.edu/wp1/robotics.pdf.

[110] Edwin P. D. Pednault. Adl: Exploring the middle ground between strips and the situation calculus. In *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*, pages 324–332, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.

[111] Doron Peled. All from one, one for all: on model checking using representatives. In *Computer Aided Verification*, pages 409–423. Springer, 1993.

[112] K. B. Petersen and M. S. Pedersen. The matrix cookbook, Nov 2012. Version 20121115.

[113] Joelle Pineau, Geoff Gordon, and Sebastian Thrun. Point-based value iteration: An anytime algorithm for POMDPs. 2003.

[114] Nir Piterman and Amir Pnueli. Synthesis of reactive(1) designs. In *In Proc. Verification, Model Checking, and Abstract Interpretation (VMCAI06*, pages 364–380. Springer, 2006.

[115] A. Pnueli. Applications of temporal logic to the specification and verification of reactive systems: A survey of current trends. In *Current Trends in Concurrency*, volume 224 of *Lecture Notes in Computer Science*, pages 510–584. Springer Berlin Heidelberg, 1986.

[116] Amir Pnueli. The temporal logic of programs. In *Foundations of Computer Science, 1977., 18th Annual Symposium on*, pages 46–57. IEEE, 1977.

[117] Amir Pnueli. The temporal semantics of concurrent programs. In *Semantics of Concurrent Computation*, pages 1–20. Springer, 1979.

[118] Pascal Poupart and Craig Boutilier. Bounded finite state controllers. In *NIPS*, 2003.

[119] A.N. Prior. *Time and Modality*. Contributions in Sociology; No. 37. Greenwood Publishing Group, 1955.

[120] International Probabilistic Planning Competition. http://users.cecs.anu.edu.au/ ssanner/ippc_2011/, 2011.

[121] Martin Proetzsch, Karsten Berns, Tobias Schuele, and Klaus Schneider. Formal verification of safety behaviours of the outdoor robot ravon.

[122] Alberto Puggelli, Wenchao Li, Alberto L. Sangiovanni-Vincentelli, and Sanjit A. Seshia. Polynomial-time verification of pctl properties of mdps with convex uncertainties. In *Computer Aided Verification*, pages 527–542. Springer, 2013.

[123] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1994.

[124] Andrea Qualizza, Pietro Belotti, and François Margot. Linear programming relaxations of quadratically constrained quadratic programs. In *Mixed Integer Nonlinear Programming*, pages 407–426. Springer, 2012.

[125] J.P. Queille and J. Sifakis. Specification and verification of concurrent systems in cesar. In *International Symposium on Programming*, volume 137 of *Lecture Notes in Computer Science*, pages 337–351. Springer Berlin Heidelberg, 1982.

[126] D. Revuz. *Markov chains*. North-Holland Pub. Co. ; American Elsevier, Amsterdam : New York :, 1975.

[127] V. I. Romanovsky. Discrete markov chains. Springer-Verlag, 1970.

[128] Victor S. Ryaben'kii and Semyon V. Tsynkov. *A Theoretical Introduction to Numerical Analysis*. Chapman and Hall/CRC, 2006.

[129] Scott Sanner. Relational dynamic influence diagram language (rddl): Language description. http://users.cecs.anu.edu.au/ ssanner/IPPC_2011/RDDL.pdf, 2010.

[130] R.L. Schilling. *Measures, Integrals and Martingales*. Cambridge University Press, 2005.

[131] Rangoli Sharan and Joel Burdick. Finite state control of pomdps with ltl specifications. 2014.

[132] H. Sherali and W. Adams. A hierarchy of relaxations between the continuous and convex hull representations for zero-one programming problems. *SIAM Journal on Discrete Mathematics*, 3(3):411–430, 1990.

[133] Hanif D Sherali and Barbara MP Fraticelli. Enhancing rlt relaxations via a new class of semidefinite cuts. *Journal of Global Optimization*, 22(1-4):233–261, 2002.

[134] H.D. Sherali and W.P. Adams. *A Reformulation-Linearization Technique for Solving Discrete and Continuous Nonconvex Problems*. Nonconvex Optimization and Its Applications. Springer, 1998.

[135] R. Simmons and S. Koenig. Probabilistic robot navigation in partially observable environments. *International Joint Conference on Artificial Intelligence*, 14:1080–1087, 1995.

[136] Kartik Krishnan Sivaramakrishnan. *Linear programming approaches to semidefinite programming problems*. PhD thesis, Rensselaer Polytechnic Institute, 2002.

[137] Richard D. Smallwood and Edward J. Sondik. The optimal control of partially observable markov processes over a finite horizon. *Operations Research*, 21(5):pp. 1071–1088, 1973.

[138] Trey Smith and Reid Simmons. Point-based pomdp algorithms: Improved analysis and implementation. *arXiv preprint arXiv:1207.1412*, 2012.

[139] George Stockman and Linda G. Shapiro. *Computer Vision*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 2001.

[140] Maria Svorenova, Ivana Cerna, and Calin Belta. Optimal control of mdps with temporal logic constraints. *CoRR*, abs/1303.1942, 2013.

[141] Wolfgang Thomas. Automata on infinite objects. In *Handbook Theor. Comp. Sci., Volume B: Formal Models and Semantics*, pages 133–192. 1990.

[142] Antti Valmari. Stubborn sets for reduced state space generation. In *Advances in Petri Nets 1990*, pages 491–515. Springer, 1991.

[143] Alan FT Winfield, Jin Sa, Mari-Carmen Fernández-Gago, Clare Dixon, and Michael Fisher. On formal specification of emergent behaviours in swarm robotic systems. *International journal of advanced robotic systems*, 2(4), 2005.

[144] Eric M. Wolff, Ufuk Topcu, and Richard M. Murray. Robust control of uncertain markov decision processes with temporal logic specifications. In *CDC*, pages 3372–3379, 2012.

[145] Tichakorn Wongpiromsarn and Emilio Frazzoli. Control of probabilistic systems under dynamic, partially known environments with temporal logic specifications. In *CDC*, pages 7644–7651, 2012.

[146] Tichakorn Wongpiromsarn, Ufuk Topcu, and Richard M. Murray. Receding horizon temporal logic planning. *IEEE Trans. Automat. Contr.*, 57(11):2817–2830, 2012.

[147] Tichakorn Wongpiromsarn, Ufuk Topcu, Necmiye Ozay, Huan Xu, and Richard M. Murray. Tulip: A software toolbox for receding horizon temporal logic planning. In *Proceedings of the 14th International Conference on Hybrid Systems: Computation and Control*, HSCC '11, pages 313–314, New York, NY, USA, 2011. ACM.

[148] Ticharkorn Wongpiromsarn. *Formal methods for design and verification of embedded control systems : application to an autonomous vehicle*. PhD thesis, 2010.

[149] A. A. Yushkevich. Blackwell optimal policies in a markov decision process with a borel state space. *Math. Meth. of OR*, 40(3):253–288, 1994.

[150] Bin Zhou, Li Gao, and Yu-Hong Dai. Gradient methods with adaptive step-sizes. *Comp. Opt. and Appl.*, 35(1):69–86, 2006.

[151] M. Zhu, M. Otte, P. Chaudhari, and E. Frazzoli. Game theoretic controller synthesis for multi-robot motion planning - part I : Trajectory based algorithms. In *IEEE International Conference on Robotics and Automation*, Hong Kong, China, May 2014, To Appear.