INSTITUTO SUPERIOR TÉCNICO
Universidade Técnica de Lisboa

Planning, Learning and Control Under Uncertainty Based on Discrete Event Systems and Reinforcement Learning

Gonçalo de Freitas Santos Neto
(Licenciado)

Dissertação para a obtenção do Grau de Doutor em Engenharia Electrotécnica e de Computadores

DOCUMENTO PROVISÓRIO

Julho de 2010

Planeamento, Aprendizagem e Controlo baseado em Sistemas de Eventos Discretos e Aprendizagem por Reforço

Nome: Gonçalo de Freitas Santos Neto
Doutoramento em: Engenharia Electrotécnica e de Computadores
Orientador: Prof. Doutor Pedro U. A. Lima

Resumo: Há um interesse crescente em modelos que combinem planeamento, controlo e aprendizagem para tomada de decisão. O objectivo destes métodos híbridos é de conservar características das diferentes abordagens: a natureza deliberativa de planos pré-programados, a natureza reactiva das políticas de controlo, e a natureza adaptativa de algoritmos de aprendizagem, que permitem abordar problemas com incerteza.

Neste trabalho escolhemos Sistemas de Eventos Discretos, e particularmente Autómatos de Tempo Estocástico, como base para a modelação, abordando problemas em que as mudanças no estado são fruto da ocorrência de eventos. Assim, escolhemos trabalhar em tempo contínuo uma vez que a ocorrência dos eventos não apresenta, de modo geral, uma periodicidade temporal.

Usamos Controlo por Supervisão como abordagem de planeamento para especificar os comportamentos que é permitido a um agente ter, fornecendo restrições de planeamento em vez de planos pré-programados fixos. O controlo do sistema é feito definindo as probabilidades de ocorrência de eventos controláveis, que se considera ocorrerem imediatamente após a decisão; este tipo de eventos está associado, por exemplo, com o início e fim da acção de um robot. Aprendizagem por Reforço é usada para optimizar a política de controlo, dentro dos limites impostos pelo supervisor.

Contribuições originais deste trabalho, para além da abordagem de modelação, incluem as definições de observabilidade total e parcial para Autómatos de Tempo Estocástico; a derivação das condições necessárias para a convergência de Aprendizagem por Reforço, sob observabilidade total e parcial; e a derivação das equações de optimalidade para situações de observabilidade parcial, discussão da relação com Processos de Decisão de Markov Parcialmente Observáveis e a extensão de um método usado para resolver esse tipo de processos ao nosso sistema de eventos.

Resultados de simulação são apresentados para ilustrar a aplicabilidade do método e os efeitos da observabilidade parcial.

Palavras-chave: Sistemas de Eventos Discretos; Autómatos de Tempo Estocático; Controlo por Supervisão; Processos de Decisão Semi-Markov; Aprendizagem por Reforço; Observabilidade.

Planning, Learning and Control Under Uncertainty Based on Discrete Event Systems and Reinforcement Learning

Abstract: There is a growing interest in models that combine planning, control and learning for decision making. The goal of these hybrid methods is to retain characteristics from the different approaches: the deliberative nature of pre-programmed plans, the reactive nature of control policies, and the adaptive nature of learning to be able to tackle uncertainty.

In this work we choose Discrete Event Systems, and particularly Stochastic Time Automata, as the basis for modeling, tackling problems where the changes in the state are driven by the occurrence of events. For this reason, we work in continuous time since the occurrence of these events does not generally show temporal periodicity.

We use Supervisory Control as an offline planning approach to specify the behaviors that the agent is allowed to have, providing planning constraints rather than fixed pre-programmed plans. The control of the system is made by defining the firing probabilities of controllable events which are considered to fire immediately after the decision; this kind of events is, e.g., associated with starting or stopping actions in a robot. Reinforcement Learning is used to optimize the control policy, within the bounds defined by the supervisor.

Novel contributions of this work besides the modeling approach include definitions of full and partial observability for Stochastic Timed Automata (STA); derivation of the conditions on the parameters of the STA that ensure convergence of the reinforcement learning algorithm under several observability conditions; and the derivation of optimality equations under partial observability, discussion of the connection with Partially Observable Markov Decision Processes (POMDP) and extension of a generic POMDP method to work with our event-based system.

Simulation results are shown to illustrate the applicability of the presented method and the effects of partial observability.

Keywords: Discrete Event Systems; Stochastic Timed Automata; Supervisory Control; Semi-Markov Decision Processes; Reinforcement Learning; Observability.

Acknowledgments

Although the road metaphor is often appropriate for an endeavor such as a PhD, while looking back parts of the path are covered with fog, and it seems sometimes walking it felt more like stumbling repeatedly in the middle of a tall grass field than an actual road. Getting across those parts was only possible with the help of a lot of people.

I'd like to start by thanking my supervisor, Pedro Lima. Back in the days of my Licenciatura final year project, when the decision to start a PhD or not came into play, I doubt he imagined the amount of hurdles this process would encounter. Still, he helped me through all those highs and lows and kept believing in the relevance of this work even during the times when I questioned it myself. One of my big wishes for this thesis is to have contributed a bit to create a framework that can serve as common ground for several of the interesting works being done at ISLab, that necessarily gravitate around his vision of the way to model, plan and execute tasks in robot (and particularly multi-robot) systems.

Also, I'd like to extend my thanks to the other members of my committee for the helpful input during the thesis proposal presentation.

Even though the path was not clear at times, it was always comforting to know I was not the only one walking it. So a big thanks to Hugo, it has been a long way since those Applied Electronics classes and I'm sure we'll get to share a road sometime in the future too, wether working or doing some random outdoor adventure. Thanks to Fernanda for being a pillar for a good friend.

I'm certain that whatever obstacles this journey found, overcoming them was also possible largely due to the great working environment I found at ISR. I thank the army of people that I met there, particularly the residents of room 6.15, from the oldies of the Socrob project, sharing nights by a robotic football field frantically trying to get a robot to push a ball, to the recent companions on the trips to Pingo Doce to hunt for dinner. Thanks to Matthijs Spaan for the helpful input and references about POMDPs, to Francisco Melo for the fruitful discussions and general insight on everything related to reinforcement learning and to both for putting up a great reading group.

My martial Way, my personal interpretation of *Budo*, is something so embedded in my life already that it's impossible to picture not following it, so I apologize my friends from Dô Clube for disappearing almost without a word at times, the times I fall down in the grass. Practicing Aikido, or some other path up the mountain, with them is always a pleasure and privilege and I only regret that, for one reason or another, I occasionally deprived myself from doing so. Domo Arigato Gozaimashita.

Whenever I think of a real team of friends, a tight group of people that would go extremes for each other, there's no way to circumvent the role that Ricardo, Bruno, Hugo and Melro have had over the years. Thanks to them for being true comrades in arms even in times of peace, drinking buddies when there's only water and generally being close by for everything even when they are living, or planning to live, quite far.

To Miguel, who has been a brother for longer than I can remember, I can see a huge swell coming for the coast so lets just grab our boards quickly and stop babbling. Big thanks to Pedro, Zé and Ana for all the votes of confidence and friendship throughout the years.

Only an amazing person would be able to stick by me through the process of transforming our older path together to a new one, a wider one but not less meaningful. Only a great friend would also value that friendship so much she would never let it go to waste – that was always the constant in our shared road . Thanks Pipas, in the end our stubbornness payed off.

My family... my father, who always refused letting me settle for half, even though it's an inglorious task most of the times; my mother, who has always supported me in everything, sometimes happily sacrificing some of her own comfort; my sister, the ultimate voice of reason and bodyguard extraordinaire, I am so proud of how she is steering her life; my grandparents, extraordinary friends and role-models who keep us all young.

Finally, some years ago a great Teacher gave, in his doctorate thesis, his thanks to the fields of Alentejo, my home region, for all the inspiration they gave him. I only wish I have had half the dedication to my thesis that he had to his and the opportunity to also have drawn inspiration from those grassy plains. Instead, I'll extend my thanks to every place where I slept under the stars with just a sleeping bag, when I was younger, for leaving an endless lucid dream floating in my head.

Now, Alan, if all else fails and you think you've lost... pretend you've won!
Works for our president.

TWO roads diverged in a yellow wood,
And sorry I could not travel both
And be one traveler, long I stood
And looked down one as far as I could
To where it bent in the undergrowth;

Then took the other, as just as fair,
And having perhaps the better claim,
Because it was grassy and wanted wear;
Though as for that the passing there
Had worn them really about the same,

And both that morning equally lay
In leaves no step had trodden black.
Oh, I kept the first for another day!
Yet knowing how way leads on to way,
I doubted if I should ever come back.

I shall be telling this with a sigh
Somewhere ages and ages hence:
Two roads diverged in a wood, and I?
I took the one less traveled by,
And that has made all the difference.

The Road Not Taken
Robert Frost, 1916

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

The use of autonomous robots or software agents to solve complex problems has become commonplace, with their applicability ranging from the traditional manufacturing robots to web-based autonomous programs. We increasingly start to rely on them to perform tasks that would otherwise be tedious, dangerous or simply impossible for a human and their increased autonomy requires that the three main functions of such agents – perception, reasoning and action – keep up with the needs of the challenging scenarios we set them to work in.

In this work we are concerned mainly with the reasoning aspect of an agent: based on the input the agent gets from his sensors, it chooses from the different available actions so that the task is accomplished. In this context, a plan is simply a sequence of actions to achieve the said task. The representation of agent plans can be made using different approaches (LaValle, 2006). On the extreme sides of the spectrum, the basic planning problem assumes that a plan can be represented as a sequence of pre-programmed actions and, on the other side, that it is no more than a policy function that maps states into actions, with the agent reacting conditionally to the state that it is in.

A pre-programmed plan, which can be generated by well studied classic AI methods (Russell and Norvig, 2003), is ill-equipped to deal with uncertainty in the sensors or in the effects of the actions. In this sense, the reactive representation of a conditional policy seems a more well suited approach to an uncertain environment, since it allows for sudden deviations of the ideal plan to be dealt with. On the other hand, the reactive representation becomes dependent on the ability to successfully identify the state in which the agent is, which in certain instances could prove to be a hindrance, bringing to mind the common expression ”stick to the plan”. In the framework of Partially Observable Markov Decision Processes (POMDPs), (Kaelbling *et al.*, 1998) , this problem is circumvented by making the plan dependent on a probability distribution over the possible states and not the states themselves, but computing the optimal plan for a POMDP is an intractable problem and POMDP solvers are mostly offline methods, with the plans being computed prior to execution.

Learning can be useful to adjust original plans to the particular features of the world where the robot evolves. If the basic planning problem is extended so as to consider decision points in the plan where one of multiple actions are chosen, the best among the conflicting actions can be learned over time. Similarly, the optimal policy can be learned over time for each of the states reachable by the robot, taking into account robot action uncertainties. A popular solution for the latter approach is to use reinforcement learning algorithms (Sutton and Barto, 1998).

Recently, there has been an increasing interest in combining the purely deliberative approach, of which classic AI planning is a good example, the reactive control approach, with the use of conditional policies to decide how to act, and the use of learning algorithms to improve over pre-programmed plans or to adapt to system changes. In fact, hybrid approaches consisting of combining all these different paradigms are, in practice, widely used to solve decision problems, from low level motor control to high level strategy control, but they have been usually obtained in ad-hoc ways that are suited to the problem one wishes to solve but are not easily extendable to other kinds of problems. Besides, since the many of the individual algorithms used for planning, learning and control have been studied extensively for a couple of decades, it is only natural that the interest shifts to the problem of effectively combining these different approaches in a way that retains certain features from each of them to produce an hybrid systematic method.

Another important question regarding the decision problem is when to decide, which is closely related to the way the state changes occur in the system we are considering. Often in situations that require decision making, it is not necessary to analyze the state of the system at each moment but only at key instants in time where an important event happened. In fact, some systems can be modeled with their dynamics being driven not by the time but by the occurrence of said events. Automata and Petri Nets, for example, are two of this sort of models generally called Discrete Event System (DES) models (Cassandras and Lafortune, 2007). The rationale for decision then becomes that, if the state change is restricted to certain moments in time when those events occur, then often it is only at those instants that the agent needs to check if its current action is adequate or if another one is needed.

In these event-based models, an observability problem arises when the agent is not able to observe some of the events the system produces. Additionally, the same event might be associated to different state changes, which causes another observability problem, closer to the one addressed by models such as POMDPs.

Related to these event-based models, where the driver is the occurrence of events, is the question of whether time is considered a continuous or discrete quantity. In fact, although the state changes themselves are driven by which events happened at a certain state, the fact that they do not occur in equally spaced intervals of time might be of critical importance to certain tasks. For example, if the task of a robot is to minimize the time it spends on a certain state since due to energetic constraints, then it becomes important that the cost function it aims to minimize includes a term that depends exactly on the time it spent on that state.

1.2 Objective

The main objective of this thesis is to provide a method for combining:

- A deliberative planning approach, that allows for pre-programmed plans to be provided to the agent in a well formalized way. Ideally, the representation of these plans should be natural and easily understandable by the designer of the agent, or at least obtained from an intermediate representation, in which the designer writes the system specifications, that is natural and easily understandable. The rationale is that a-priori plans should not only be the ones possibly optimized offline by the agent itself, but they might be related to constraints that the designer wants to force on the agent for the problem at hand. A possible application is when a generic robot with a set of possible behaviors is set on a new environment possibly to learn how to act in this new environment, but the designer wants to ensure there are certain sequence of actions it will never try, for safety reasons. Another possible use is to restrict the set of behaviors of the agent by removing the ones the designer knows not make sense to use in certain states. In this sense, it is our goal to introduce pre-programmed plans not so much as pre-determined sequences of actions but as restrictions of the behavior space – instead of telling the agent to do a certain sequence, we aim to use planning to tell him which sequences he should not do, effectively supervising the agent behavior.

- A reactive control approach, which simply provides the agent with the adequate action for a given state. The goal is for the conditional plans induced by this control component to live only in the behavior trajectory space allowed by the planning module.

- A learning approach, which optimizes the control policy based on some reward/cost measure. Again, the optimization is supposed to be done within the boundaries of the set confined by the planning module.

Additionally, we are also concerned with using models that work on top of an event-based approach. Considering we are interested in obtaining a method that is well suited mainly for high level control, most of the applications will only require decisions to be made in specific instants in time, where a key event has happened. In fact, in this sort of abstraction it is common to have discrete states and the decisions only need to be made when the system makes a jump from one state to the other. Furthermore, assuming the state changes are dependent on the occurrence of events and not time allows for the controller to react independently of changes in the rate at which the system is producing events and changing state.

Finally, it is our goal to use a continuous time assumption. Explicitly considering time is important for two reasons: the cost functions to optimize are often time dependent and, on the other hand, although the changes in the system state are driven by the occurrence of events, it is quite common for some of the events to occur driven by some temporal process. The number of costumers in a queue, e.g., is a state variable driven by arrivals, departures and services, and in that sense it is temporally abstract. Nevertheless, the underlying processes of occurrence of each of the events are driven by time.

1.3 Approach

For a DES view only concerned with the untimed (or logical) behavior of the system, Supervisory Control (SC), as introduced in (Ramadge and Wonham, 1987), is an adequate method to steer a plan designed to accomplish the specifications. Stochastic Timed Automata (Glynn, 1989) and Petri nets cover the stochastic (possibly timed) view of DES, which enables modeling uncertain action effects and uncertainty about the robot state. When endowed with controllable actions and under particular conditions, Stochastic Timed Automata are equivalent to Semi-Markov Decision Processes (SMDPs) or even MDPs (Puterman, 1994).

MDPs have been a top choice in the literature to address the problems of sequential decision making under uncertainty in a robotic setup. Solving model-free MDPs can be achieved by the application of reinforcement learning (RL) techniques (Sutton and Barto, 1998) that allow the robot to have no prior knowledge about the environment (apart from its state space and the available actions), learning the optimal course of action from experience. Q-learning (Watkins, 1989) is one of the most popular algorithms to address the RL problem. Semi-Markov Decision Processes (SMDP) are also addressed in (Puterman, 1994) and RL algorithms for such systems can be found in (Bradtke and Duff, 1995). SMDPs are particularly relevant when time can not be considered discrete.

In this thesis, we model the environment as a Stochastic Timed Automata generating a Generalized Markov Decision Process. We extend the STA with controllable events and combine supervisory control of DES and RL to combine a specifications-based approach to the supervision of robot plan execution with a policy controller which uses a RL algorithm to learn the optimal policy over time for the behaviors enabled by the supervisor. Since our method is event-based by design, we chose to consider continuous time because the triggering of events is in general asynchronous. To handle continuous time, we use a Q-learning algorithm extended for SMDPs. We presented this approach in (Neto and Lima, 2008).

Combining RL based controllers with supervision has been addressed before, though not too often. Furthermore, the methods that do exist are not usually concerned with the systematic integration of specifications or a priori planning knowledge with the learning based controller, which is one of the goals of our work.

Additionally, we consider that some events that occur in the environment cannot be observed by the supervisor and by the controller (leading to partial state observability) and that the effects of the observable events are in general non-deterministic (thus modeling uncertain action effects). This poses an additional problem of building an adequate observer to support the decision of the supervisor and controller systems. We build a deterministic observer of the environment based on the methods described in (Cassandras and Lafortune, 2007) and use it to obtain the state estimate the RL-based controller requires. We discuss the conditions necessary for the process to maintain a semi-Markovian property on the states of the observer.

We also address the situation where the effects of the occurrence of an event are uncertain and how that leads to an observability problem since the state is not univocally known. We use a probabilistic DES as an observer and obtain new optimality equations based on the distributed state of the probabilistic observer which, in their form, will be proven to be quite similar to the equivalent optimality equations for POMDPs (Kaelbling *et al.*, 1998). We present the modified optimality equations and describe how to adapt a generic POMDP solver to our problem.

1.4 Contributions

The key contributions of this thesis are:

- The cornerstone contribution is a **novel approach to combining deliberative planning concepts, reactive control and learning by using supervisory control of discrete event systems to provide loosely specified planning options over which a reinforcement learning based controller can optimize**. Building this model was paramount to providing the conditions to obtain specific theoretical and practical results, and it can be used as a starting point to other work sharing the same philosophy.

- The second contribution is the definition of **full observability for a system based on a controllable stochastic timed automata**, from which the agent can only directly see the events produced, and the proof of a result describing the necessary and sufficient conditions for full observability.

- We also show how, under full observability, the model relates to a semi-Markov decision process and present the **conditions that the firing of events needs to follow in order to ensure the convergence of the modified Q-learning algorithm, proving that the algorithm does indeed converge**. The novel aspect of this result is the fact that all the parameters are associated with events, which induces some particular aspects in the optimality equations.

- Under partial observability, we use a known technique to construct an observer given by a finite state automaton, with our contribution being a result **showing that the states of the observer, which are associated with parts of the**

original STA, can be proven to still provide a semi-Markov process which can be used for learning. We show that this happens if the transition to the sub-automaton associated with an observer state is made always to the same state of the original STA. In fact, since each sub-automaton of the STA under a fixed control policy can be thought of as a Markov chain, it can be said that the condition for use of the Q-learning algorithms requires that each of these Markov chains always has the same initial condition.

- Finally, still under partial observability, we **derive the optimality equations for the system when the observer is a probabilistic system and show how we can modify a generic POMDP algorithm to solve our event based problem**.

1.5 Thesis Structure

The thesis is structured as follows:

Chapter 2 covers the basic frameworks and models that support the work done in this thesis. It starts by describing Finite Automata, Supervisory Control of Discrete Event Systems and Stochastic Timed Automata, and some of the linguistic properties of those models. On the subject of Markov Decision Processes, it presents the supporting framework, describes optimality equations and ways to solve them using dynamic programming or reinforcement learning; it explores convergence properties in further detail for the extension of Markov Decision Processes to continuous time. It closes by describing the framework of Partially Observable Markov Decision Processes and going through the optimality equations for the model.

Chapter 3 reviews the most significant literature in the fields related to this thesis, particularly new results in Supervisory Control of Discrete Event Systems, reinforcement learning algorithms in Semi-Markov Decision Processes, combination of supervision with reinforcement learning and recent ways to provide supervisor specifications.

Chapter 4 starts by describing the supporting event-based model for the system we want to control – Stochastic Timed Automata – and introduces some extensions of the basic model to allow events to be classified according to the time it takes for them to fire and their controllability. An observer model is presented and the conditions which the system must meet to achieve full observability are derived. A supervisor and an adaptive controller are presented and optimality equations are derived for the controller, as well as theoretical results that explain under which conditions the update rule of the controller will converge. A case study is described and simulation results presented.

Chapter 5 continues the work on the framework presented in Chapter 4 but the full observability assumptions are dropped. A way to construct a deterministic observer is shown and some results are derived explaining how each state of the observer can be associated with a part of the system behavior, a semi-Markov subchain of the system. The conditions that the observer states must meet to ensure they are also semi-Markovian are derived. The case study from Chapter 4 is extended to include partial observability and the new results are discussed.

Chapter 6 explores a partial observability situation but where the observer is not deterministic. The model for the probabilistic observer is presented and the optimality equations derived. It is show the probabilistic observer induces equations similar to the ones for POMDPs presented in Chapter 2 and a way to modify a generic POMDP solver to this problem is described.

Chapter 7 reviews the thesis by highlighting the main results and contributions and closes by discussing several directions that could be taken to extend this work, in the future.

Readers should start by Chapter 2 unless they are familiar with the concepts presented there. Chapter 3 provides an overview on several of the fields related to this work but is not essential to understand the concepts presented in the thesis. In Chapter 4 the main modelling approach of the thesis framework is presented and Chapters 5 and 6 are critically dependent on it. Chapter 6 mentions some results from Chapter 5 but the chapters can be read independently.

Chapter 2

Background

2.1 Discrete Event Systems

Discrete Event Systems (DES) share the characteristic of having discrete state spaces and their dynamics being driven by events, rather than time. They can be inherently discrete or an abstraction of an hybrid system, where both continuous and discrete dynamics rule the system changes.

The concept of event is that of an instantaneous occurrence that causes the system to change its state. Typically, a set of events E can be interpreted as an alphabet and, in that case, a language over that alphabet describes the *behavior* of the system. Usual operations on languages, such as concatenation or Kleene closure, can be applied here. For more references on the operations on languages a possible source is (Hopcroft *et al.*, 2000).

There are several formalisms that can be used to model these kind of systems – the motivation to use these representations is to express behaviors, which with a language representation would origin infinite sets, in a finite and compact way. This is not always possible and the degree of applicability of each of these *discrete event modeling formalisms* depends on the complexity of the system being represented.

On the logical level, the field of *Formal Language Theory* that stemmed from the hierarchical definitions of Noam Chomsky (Chomsky, 1955), classifies the languages according to their expressive power. The different DES modeling formalisms can then be classified according to the expressive power of the languages they generate[1].

Particularly, the simplest class of languages in this hierarchy is the class of *Regular Languages*, $\mathcal{R}$, and it corresponds to the languages generated by *Finite-State Automata* (FSA). Petri nets are another kind of formalism, which has more expressive power than

[1] Strictly speaking, languages are said to be generated by Grammars. Automata take strings of a language as inputs and are said to accept or recognize the language, if they classify the inputs, e.g., giving a yes/no answer to each input string, or transduce it, if they translate their input language to another. Moore and Mealy automata are of this later kind.

In DES theory, however, since automata can also provide a way to express the production rules associated to grammars, a common abuse of terminology is to consider automata to be language generators. We follow this terminology throughout the thesis.

FSA, thus possibly generating languages not in the $\mathcal{R}$ class.

2.1.1 Finite State Automata

Modeling DES using the complete description of the possible behaviors is at least tedious. Several formal models are frequently used, that correspond to different levels of language complexity, to represent the system in a more compact way. In particular, systems whose behaviors can be described by regular languages can be represented by Finite State Automata (FSA), (Cassandras and Lafortune, 2007). A Finite State Automaton is a tuple $G = (X, E, f, \Gamma, x_0, X_M)$ where:

- X represents a finite state space.
- E represents a finite event set.
- $f : X \times E \to X$ is a possibly partial function representing the state transitions. This function can be extended to strings (sequences of events) in the following way:

$$\begin{aligned} &f(x, \varepsilon) = x \\ &f(x, se) = f(f(x, s), e) \text{ for } s \in E^*, e \in E \end{aligned}$$

- $\Gamma : X \to 2^E$ is the set of enabled events in state x.
- x_0 is the initial state of the system.
- $X_M \subseteq X$ is a set of marked states.

where ε represents an empty string and E^* the Kleene Closure of E, that is, the set of all finite strings of elements of E, including ε. Again, this function is only defined for the strings that the automaton recognizes.

The complete language generated by the automaton denotes all the possible paths that can occur starting from the initial state, that is:

$$\mathcal{L}(G) = \{s \in E^* : f(x_0, s) \text{ is defined } \}$$

Similarly, the set of goal behaviors for the automaton defines the language marked by the automaton:

$$\mathcal{L}_M(G) = \{s \in \mathcal{L}(G) : f(x_0, s) \in X_M\}$$

With the definition presented above, the FSA is said to be a Deterministic FSA (DFA). Although the FSA defined previously is considered a generator, there is no definition of the rules that define the generation of events. This is different from, for example, assuming that all events have the same probability of occurring in a given state, which says that we know what is the generating mechanism but no event is more likely to happen than other.

Two important concepts are those of *accessibility* and *co-accessibility.*

Accessibility A state is said to be accessible if it can be reached by some path (in other words, some string in $\mathcal{L}(G)$) starting from the initial state. The operation $Ac(G)$

takes the accessible part of the automaton G, which is useful since it eliminates states that will never be visited starting from x_0.

The *Ac* operation preserves the languages generated and marked by the automaton, so when working with automata it is usual to assume the automaton is accessible, $G = Ac(G)$. We make that assumption throughout the thesis.

Co-accessibility A state is said to be co-accessible if starting from that state, it is possible to reach some marked state. Similarly, the $CoAc(G)$ operation takes only the co-accessible part of the automaton G. States that are not co-accessible are not desirable since they prevent the automaton from reaching one of the goal states.

The *CoAc* operation shrinks the behavior of the system, possibly removing some of the strings in $\mathcal{L}(G)$. However, it preserves $\mathcal{L}_M(G)$. The automaton is said to be co-accessible if $G = CoAc(G)$, or in other words, the prefixes of all strings in $\mathcal{L}_M(G)$ belong to $\mathcal{L}(G)$ [2].

Co-accessibilty is also useful to model *deadlocks*, a non marked state from where the automaton cannot leave, or *livelocks*, a set of states where the automaton shows dynamic behavior but from where it is not possible to reach a marked state. More generally, an automaton is said to be *blocking* if:

$$\overline{\mathcal{L}_M(G)} \subset \mathcal{L}(G)$$

where the inclusion is strict.

Accessibility and Co-accessibility induce, as seen, two unary operations on automata. Also of interest are binary operations to compose automata. Particularly, the *product* and the *parallel composition*.

Product: Given two automata

$$G_1 = (X_1, E_1, \Gamma_1, f_1, x_{01}, X_{M1})$$

$$G_2 = (X_2, E_2, \Gamma_2, f_2, x_{02}, X_{M2})$$

the product $G_1 \times G_2$ is an automaton

$$(X_1 \times X_2, E_1 \cap E_2, \Gamma_{1\times 2}, f_{1\times 2}, (x_{01}, x_{02}), X_{M1} \times X_{M2})$$

where

$$f_{1\times 2}((x_1, x_2), \sigma) = \begin{cases} (f_1(x_1, \sigma), f_2(x_2, \sigma)) & \text{if } \sigma \in \Gamma_1(x_1) \cap \Gamma_2(x_2), \\ \text{undefined} & \text{otherwise.} \end{cases}$$

and

$$\Gamma_{1\times 2}(x_1, x_2) = \Gamma_1(x_1) \cap \Gamma_2(x_2)$$

[2] The operation of taking all the prefixes of a given language is called *prefix closure* and is usually denoted $\overline{(\cdot)}$. In that sense, a co-accessible automaton has $\mathcal{L}(G) = \overline{\mathcal{L}_M(G)}$.

Parallel Composition: Given two automata $G_1 = (X_1, E_1, \Gamma_1, f_1, x_{01}, X_{M1})$ and $G_2 = (X_2, E_2, \Gamma_2, f_2, x_{02}, X_{M2})$, the parallel composition $G_1 \| G_2$ is an automaton

$$(X_1 \times X_2, E_1 \cup E_2, \Gamma_{1\|2}, f_{1\|2}, (x_{01}, x_{02}), X_{M1} \times X_{M2})$$

where

$$f_{1\|2}((x_1, x_2), \sigma) = \begin{cases} (f_1(x_1, \sigma), f_2(x_2, \sigma)) & \text{if } \sigma \in \Gamma_1(x_1) \cap \Gamma_2(x_2), \\ (f_1(x_1, \sigma), x_2) & \text{if } \sigma \in \Gamma_1(x_1) \setminus E_2, \\ (x_1, f_2(x_2, \sigma)) & \text{if } \sigma \in \Gamma_2(x_2) \setminus E_1, \\ \text{undefined} & \text{otherwise.} \end{cases}$$

and

$$\Gamma_{1\|2}(x_1, x_2) = (\Gamma_1(x_1) \cap \Gamma_2(x_2)) \cup (\Gamma_1(x_1) \setminus E_2) \cup (\Gamma_2(x_2) \setminus E_1)$$

The product of automata corresponds to identifying the behavior shared by both automata, and synchronizes the automata in those events. On the other hand, the parallel composition synchronizes the automata in the events that they share, and does not constrain the events that they do not, with them being executed whenever possible.

It is possible to define a Non-deterministic FSA (NFA) as a tuple $G_{nd} = (X, E \cup \{\varepsilon\}, f_{nd}, \Gamma, x_0, X_M)$ where:

- X represents a finite state space
- $E \cup \{\varepsilon\}$ represents a finite event set with ε denoting an empty string. The transitions labeled with the empty string can be thought of as changes that are not possible to be detected.
- $f_{nd} : X \times E \cup \{\varepsilon\} \to 2^X$ is a possibly partial function with $f_{nd}(x, e)$ representing the states in X that can be reached from x by action of event e. Note that e might also be the empty string. This function might also be extended to strings:

$$f_{nd}(x, se) = \{z : z \in f_{nd}(y, e), y \in f_{nd}(x, s)\}$$

- $\Gamma : X \to 2^E$ is the set of enabled events in state x.
- x_0 is the initial state of the system, which in the case of a NFA might be a subset of X.
- $X_M \subseteq X$ is a set of marked states.

The languages generated and marked by a NFA are:

$$\mathcal{L}(G_{nd}) = \{s \in E^* : \exists_{y \in x_0} f(y, s) \text{ is defined }\}$$

$$\mathcal{L}_M(G_{nd}) = \{s \in \mathcal{L}(G_{nd}) : \exists_{y \in x_0} f(y, s) \cap X_M \neq \emptyset\}$$

Since the objective of the systems we intend to study will be given by a mechanism different than goal states, particularly the use of reward functions and reinforcement learning algorithms, we will drop marked states set X_M from the definition of the automata we use.

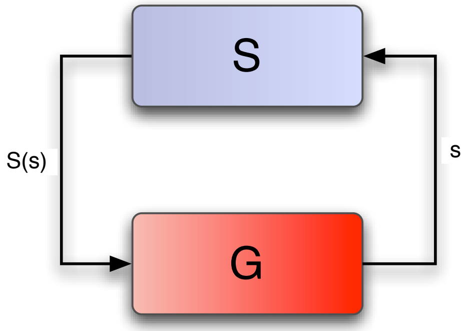


Figure 2.1: Supervisory Control block diagram.

2.1.2 Supervisory Control

In order to control a system modeled as an FSA, the event set is partitioned in a controllable event set E_C and an uncontrollable event set E_{UC} such that $E = E_C \cup E_{UC}$, $E_C \cap E_{UC} = \emptyset$. A supervisor is then defined as a function:

$$S : \mathcal{L}(G) \rightarrow 2^E$$

where $\mathcal{L}$ represents the language generated by the G. The supervisor basically chooses a set of events to enable, for each string generated by the system. For a system starting at state x_0 and after string s has occurred, the set of enabled events is given by $S(s) \cap \Gamma(f(x_0, s))$. Figure 2.1 shows the block diagram for this kind of systems.

It is important to note that the supervisor, in order to be *admissible*, is not allowed to disable uncontrollable events:

$$\forall_{s \in \mathcal{L}(G)} E_{UC} \cap \Gamma(f(x_0, s)) \subseteq S(s)$$

The control model can be further elaborated by adding a notion of partial observability. Essentially, the event set will be partitioned in two other disjoint sets $E = E_O \cup E_{UO}$, $E_O \cap E_{UO} = \emptyset$ where E_O represents the set of observable events and E_{UO} the set of unobservable events. In this case, it is possible to build another automaton, called an observer automaton, that keeps track of an estimate of the state space of the original automaton, but based solely on E_O. The observer is a FSA defined as a tuple $G_{obs} = (X_{obs}, E, f_{obs}, \Gamma_{obs}, x_{0,obs}, X_{M,obs})$ with $X_{obs} \subseteq 2^X \setminus \emptyset$ and $f_{obs} : X_{obs} \times E \rightarrow X_{obs}$ being a deterministic transition function. An algorithm for constructing G_{obs} can be found in (Cassandras and Lafortune, 2007).

Note that the states of the observer are nothing more than subsets of the state space X of the original automaton, i.e., the estimates of the observer correspond to the states in which the original automaton might be in, at a given time.

A projection function $P : E^* \rightarrow E_O^*$ is associated with the original event set and the observable one (or rather, the respective Kleene closures). It essentially erases the unobservable events from a string.
The overall block diagram is represented in Figure 2.2.

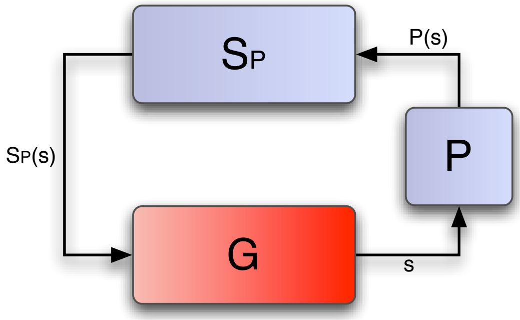


Figure 2.2: Supervisory Control block diagram under partial observation.

The supervisor is now defined over the projection of the language generated by G:

$$S_P : P[\mathcal{L}(G)] \to 2^E$$

This new supervisor will only change its control action after the occurrence of an observable event, and for the same reason if two strings have the same projection it will issue the same control action. This has to be taken into account when designing the supervisor.

The behavior of the supervised system can also be represented by a language, usually denoted by $\mathcal{L}(S/G)$, and in some cases generated by a FSA. We focus particularly on finite state systems, whose restricted and unrestricted behaviors mark regular languages, and can be represented by FSA. If supervisor S is realized by an FSA R, which we assume to have an event set E_R such that $E_R \subset E_G$, then the closed loop controlled system can be interpreted as the parallel composition of the system and the automaton $G \| R$. Such a supervisor will always enable events e such that $e \in E_G \setminus E_R$, by definition of parallel composition, but even if $e \notin \Gamma_G(x)$ for some state $x \in X$, there is no harm in enabling unfeasible events since they will simply never happen. This fact allows for the supervisors to only need to focus on the part of the system behavior they want to restrict and not have to mimic the entire dynamics of the system.

The controllability theorems (under full or partial observability) for supervisory control systems can be found in (Cassandras and Lafortune, 2007; Wonham, 1997).

2.1.3 Modular Supervisory Control

A useful way of constructing supervisors is to have not a single automaton controlling the system but several supervisors in parallel. This method, explained in a block diagram in Figure 2.3 is called in (Cassandras and Lafortune, 2007) *Modular Supervisory Control*(MSC). The *modular* supervisor is defined as a function $S_{modP} : P[\mathcal{L}(G)] \to 2^E$ such that:

$$S_{modP}(s) = \bigcap_{i=1}^{n} S_{iP}(s)$$

Essentially, the only behaviors that the system is allowed to have are those that all the supervisors agree on and, for that reason, we can also say that:

$$\mathcal{L}(S_{modP}/G) = \bigcap_{i=1}^{n} \mathcal{L}(S_{iP}/G)$$

If we assume that each of the supervisor functions S_{iP} realized by a FSA R_i, then

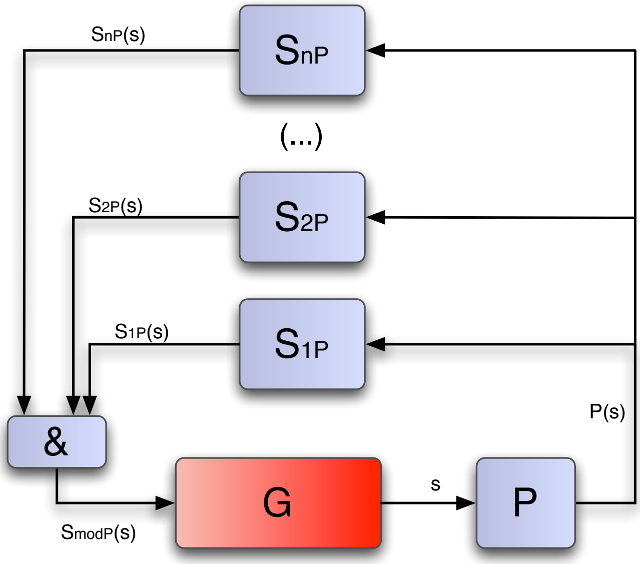


Figure 2.3: Modular Supervisory Control block diagram under partial observation.

another interpretation for MSC is that the system is being controlled by the supervisor induced by the parallel composition of all the automata:

$$R_{1\|\cdots\|n} = R_1\|\cdots\|R_n$$

and the system in closed loop can be interpreted as:

$$G_{closed} = G\|R_{1\|\cdots\|n} = G\|R_1\|\cdots\|R_n$$

Modular Supervisory Control provides a practical way to not only construct an overall supervisor that is the combination of several specifications, each one focusing on a different aspect of the system that they want to control, but it is also a good way to save memory. If $|X_i|$ is the number of states of automaton R_i, then the total number of states that the modular supervisor needs is:

$$\sum_{i=1}^{n} |X_i|$$

since they are stored in parallel. On the other hand, if we note that the state space of $R_{1\|\cdots\|n}$ is, by the definition of parallel composition, potentially equal to $X_1 \times \cdots \times X_n$, then the total number of states needed is potentially equal to:

$$\prod_{i=1}^{n} |X_i|$$

Which shows how the modular approach is much more economic than the explicit construction of an equivalent non-modular supervisor.

2.1.4 Stochastic Timed Automata

As opposed to the previously defined automata, which did not consider time, Stochastic Timed Automaton (STA) (Glynn, 1989) include the notion of (continuous) time associated with each transition. They can be defined as a tuple $(X, E, \Gamma, p, p_0, \mathcal{T})$ where:

- X represents a state space.
- E represents an event set.
- $\Gamma : X \rightarrow 2^E$ is the set of enabled events in state x
- $p : X \times E \times X \rightarrow [0, 1]$ is a transition function defined as:

  $$p(x'|x, e) = P\left[X_{k+1} = x'|X_k = x, E_k = e\right]$$

  The function is possibly partial, not defined for events $e \notin \Gamma(x)$. The indices $k, k+1, \ldots$ are associated with the moments in time where some event occurs.
- $p_0(x)$ is a probability distribution over X that represents the knowledge about the initial state of the system, $P\left[X_0 = x\right]$.
- $\mathcal{T} = \{\mathcal{T}_e : e \in E\}$ is a stochastic clock structure (see (Cassandras and Lafortune, 2007)) for the event lifetimes. Essentially it associates to each event e a stochastic clock sequence $\{V_{e,t}\} = \{V_{e,1}, V_{e,2}, \ldots\}$, which is independent and identically distributed. The index k represents, for each event, the moments in time where the event e either became active, $e \notin \Gamma(x) \wedge e \in \Gamma(x')$, or was triggered.

The STA defined like this is clearly a generator automaton, with the occurrence of events in a given state being driven by the stochastic clock structure and the probability distributions associated with it.

The Markovian property of this system is related to the fact that the state transitions only depend on the current state and the event fired. However, the event lifetimes have arbitrary distributions and, as mentioned above, if an event is active in some state and remains active in the next state without being triggered, its residual lifetime depends not only on the time it has been active in the current state, but also for how long it has been active previously.

Supervisory Control can in fact be applied to systems modeled by STAs, when focusing on the logical behavior of the system, thus ignoring the probabilistic components of the model.

2.2 Markov Decision Processes

Most research on Reinforcement Learning is built on the formalism of *Markov Decision Processes* (Bellman, 1957; Bertsekas, 1983; Howard, 1960; Puterman, 1994), or MDPs. MDPs can be defined as a tuple (X, A, p, r) where:

- A is an action set.
- X is a state space.
- $p : X \times A \times X \longrightarrow [0, 1]$ is a transition function defined as a probability distribution over the states. Hence, we have $p(x'|a, x) = P\left[X_{k+1} = x'|X_k = x, A_k = a\right]$. X_{k+1} is a random variable representing the state of the process at time $k + 1$, X_k the state at time t and A_k the action taken after observing state x_k.
- $r : X \times A \longrightarrow \mathbb{R}$ is a reward function representing the expected value of the next reward, given the current state x and action a: $r(x, a) = E\left[R_k|X_k = x, A_k = a\right]$. In this context R_k is a random variable representing the immediate payoff of the environment to the agent at time k.

This formalism assumes a discretization of time, with decisions being taken at each time step.

The fact that there are no time dependences either on p or on r is due to the stationarity (by definition) of the MDP. Although this is not stated directly, reinforcement learning algorithms without a generalization mechanism are usually designed to work on *finite MDPs*, that is, MDPs in which both the state and the action spaces are finite. So, unless clearly stated otherwise, the term MDP will refer to a finite Markov decision process.

The task of deciding which action to choose in each state is done by a *policy* function. Generally, a policy is a collection of probability distributions, one for each trace of the the process – $\pi(x_k, a_{k-1}, x_{k-1}, a_{k-2}, \dots) \in PD(A)$ – defining the probability that each action will be chosen for that particular trace of the system. However, there is no need to consider other than Markovian policies because the MDP itself is Markovian by construction – it is sufficient to define the policy for each state of the MDP.

A policy can also be thought as a projection transforming the MDP in to an induced discrete-time Markov chain. The interesting thing with this idea is that the Markov chains theoretic paraphernalia becomes available and can act as a performance measure for the current policy. In (Bhulai, 2002) the two frameworks are related and the results applied to fields traditionally related to Markov chains such as control of queuing systems.

Optimality Concepts

The goal of an agent living in an environment that can be modeled as a Markov Decision Process is to maximize the expected reward over time, which on itself aggregates a myriad of formulations. The most common criteria are:

Finite-horizon model: in this scenario the agent tries to maximize the sum of rewards for the following M steps:

$$E\left\{\sum_{k=0}^{M} r(x_k, a_k)\middle| x_0 = x, \pi\right\}$$

The objective is to find the best action, considering there are only M more steps in which to collect rewards.

Infinite-horizon discounted reward model: in this scenario the goal of the agent is to maximize reward at the long-run but favoring short-term actions:

$$E\left\{\sum_{k=0}^{\infty} \gamma^t r(x_k, a_k)\middle| x_0 = x, \pi\right\} \quad , \quad \gamma \in [0, 1[$$

The discount factor γ regulates the degree of interest of the agent: a γ close to 1 gives similar importance to short-term actions and long-term ones; a γ close to 0 favors short-term actions.

Average reward model: in this model the idea is to find actions that maximize average reward on the long-run:

$$\lim_{M\to\infty} E\left\{\frac{1}{M}\sum_{k=0}^{M} r(x_k, a_k)\middle| x_0 = x, \pi\right\}$$

This model makes no distinction between policies which take reward in the initial phases from others that shoot for the long-run rewards.

The first criterion could be used to model systems where there is a hard deadline and the task has to be finished in M steps. In reinforcement learning, usually the adopted model is the *Infinite-horizon discounted reward model*, probably not only because of its characteristics but because it bounds the sum.

A fundamental concept of algorithms for solving MDPs is the *state value function*, which is nothing more than the expected reward (in some reward model) for some state, given the agent is following some policy:

$$V^{\pi}(x) = E\left\{\sum_{k=0}^{\infty} \gamma^k r(x_k, a_k)\middle| x_0 = x, \pi\right\} \tag{2.1}$$

Similarly, the expected reward given the agent takes action a in state x and following policy π could also be defined:

$$Q^{\pi}(x, a) = E\left\{\sum_{k=0}^{\infty} \gamma^k r(x_k, a_k)\middle| x_0 = x, a_0 = a, \pi\right\} \tag{2.2}$$

This function is usually know as *Q-function* and the corresponding values as *Q-values.*

From Equation (2.1) a relation can be derived, which will act as the base of much of the ideas behind dynamic programming and reinforcement learning algorithms to solve MDPs.

$$\begin{aligned} V^{\pi}(x) &= E\left\{\sum_{k=0}^{\infty}\gamma^k r(x_k,a_k)\middle| x_0=x,\pi\right\} \\ &= E\left\{r(x_0,a_0)+\gamma\sum_{k=1}^{\infty}\gamma^{k-1}r(x_k,a_k)\middle| x_0=x,\pi\right\} \\ &= \sum_a \pi(x,a)\left[r(x,a)+\gamma\sum_{y\in X}p(y|a,x)E\left\{\sum_{k=0}^{\infty}\gamma^k r(x_k,a_k)\middle| x_0=y,\pi\right\}\right] \\ &= \sum_a \pi(x,a)\left[r(x,a)+\gamma\sum_{y\in X}p(y|a,x)V^{\pi}(y)\right] \end{aligned} \tag{2.3}$$

Note that the term $\pi(x,a)$ is used to represent the probability of taking action a in state x. In some situations, the term $\pi(x)$ will be used to represent the action selected by policy π. Generally, this a probabilistic outcome, except for deterministic policies where π can be thought as $\pi : x \rightarrow A$.

The resulting equation, called the *Bellman equation*, has a unique solution for each policy which is the state value function for that policy (Howard, 1960).

As previously stated, the goal of solving a MDP is usually to find a policy that guarantees a maximal reward, with some given reward criterion. Using state values, a policy π' is said to dominate a policy π if and only if, for every x in the state space, $V^{\pi'}(x) > V^{\pi}(x)$. An optimal policy is one which is *undominated* in the sense that no other can expect to do better, in any state. An optimal policy π^* is always guaranteed to exist[3] and sometimes even more than one, although they share the same value function, which can be defined as:

$$V^*(x) = \max_{\pi} V^{\pi}(x) \ \ , \ \forall_{x\in X} \tag{2.4}$$

It can be show (Puterman, 1994) that there is always at least one undominated deterministic policy that satisfies the previous relationship. Knowing that, we obtain the *Bellman optimality equation*:

$$V^*(x) = \max_a \left[r(x,a)+\gamma\sum_{y\in X}p(y|a,x)V^*(y)\right] \tag{2.5}$$

or in terms of the Q-values:

$$Q^*(x,a) = r(x,a)+\gamma\sum_{y\in X}p(y|a,x)\left[\max_{b\in A}Q^*(y,b)\right] \tag{2.6}$$

[3] This can be proven by construction, choosing at each state the action that maximizes V for that state and discarding the rest of the policy. On the other hand, a maximum value of V always exists in the infinite-horizon discounted reward model due to $\gamma < 1$.

2.2.1 Dynamic Programming

Solving MDPs has always been closely tied to the idea of *dynamic programming*[4], which was introduced by (Bellman, 1957) as a possible solution to a wide variety of problems. From then on, a lot of research has been done in that area and extensive treatments of the subject can be found in texts like (Bertsekas, 1987; Bertsekas and Tsitsiklis, 1996; Ross, 1983). An interesting approach is the one of (Cassandras and Lafortune, 2007) which relates dynamic programming with other methods for controlling *discrete event systems*. It has been also widely used in *optimal control* applications.

Two classical dynamic programming methods for MDPs are *value iteration* and *policy iteration*.

Value Iteration

As stated previously, a way of finding an optimal policy is to compute the optimal value function. *Value iteration* is an algorithm to determine such function, which can be proved to converge to the optimal values of V. The core of the algorithm is:

$$V_{k+1}(x) = \max_a \left[r(x,a) + \gamma \sum_{y \in X} p(y|a,x) V_k(y) \right] \tag{2.7}$$

Note that the expression was obtained by turning the Bellman optimality Equation (2.5) into an update rule. An important result about value iteration is that it is guaranteed to find an optimal greedy policy in a finite number of steps, even though the optimal value function may not have converged – in fact, usually the optimal policy is found long before the value function has converged. In (Bertsekas, 1987) those questions are discussed, along with some convergence proofs.

One thing lacking in the definition of the algorithm is the termination rule – it is not obvious when the algorithm should stop. A typical stopping condition bounds the performance as a function of the *Bellman residual* of the current value-function. That is the approach taken by (Sutton and Barto, 1998) in their description of the algorithm

[4] The term *reinforcement learning* can be associated with the body of methods intended to compute policies that maximize a given expected long term reward criterion, dealing with the problems of delayed credit assignment and how to back-propagate such reward signals. In this sense, dynamic programming can be thought of as a kind of reinforcement learning.

Similarly, classical reinforcement learning algorithms can be thought of as following a dynamic programming approach that uses stochastic approximation techniques and sampled paths of the system to learn the intended policies – in (Bertsekas and Tsitsiklis, 1996) they are known as *neuro-dynamic programming*.

So, the border between the two terms is not always distinct and most of the times they are interchangeable, but throughout this thesis we generally refer to algorithms that use stochastic approximation techniques, and are generally done online, as reinforcement learning, and consider that the term dynamic programming refers to the more traditional algorithms that are generally done offline and require a model of the system. The chosen terminology does not reflect any fundamental considerations about what names are more appropriate to each situation but rather a pragmatical choice of succinctly distinguish between two different approaches for the same problem.

and it states that if:

$$\max_{x \in X} |V_{k+1}(x) - V_k(x)| < \epsilon$$

then

$$\forall_{x \in X} \quad |V_{k+1}(x) - V^*(x)| < \frac{2\epsilon\gamma}{1-\gamma}$$

Another stopping criterion, which may lead the algorithm to stop some iterations earlier, is discussed in (Puterman, 1994).

Although the algorithm assumes a full sweep through the state space before passing to the next iteration, the assignments of V do not need to be done through successive sweeps. *Asynchronous dynamic programming* algorithms, originally proposed by (Bertsekas, 1982; Bertsekas, 1983) who also called them *distributed dynamic programming* algorithms, back up the values of the states in an indefinite order and, in fact, the value of a state can be backed up several times before the value of another one gets backed up even once. The condition for convergence is that all states are backed up infinitely often. These algorithms were further discussed in (Bertsekas and Tsitsiklis, 1989).

Policy Iteration

Another way of finding an optimal policy in a finite MDP is by manipulating the policy directly rather than finding it through the state values. A simple algorithm for doing that is based on the idea of alternating two different steps: a *policy evaluation* step plus a *policy improvement* step. Since there is a finite number of deterministic policies, the algorithm must converge to the optimal policy in a finite number of steps of policy iteration. However, the policy evaluation step is itself an iterative procedure but starting it with the value from the previous policy evaluation step generally increases the speed of convergence.

In the policy evaluation step, the state values corresponding to the starting policy are computed based on an iterative expression which, similarly to value iteration, is taken from the Bellman equation, although this time not for the optimal values:

$$V^{\pi}_{k+1}(x) = \sum_{a} \pi(x,a) \left[r(x,a) + \gamma \sum_{y \in X} p(y|a,x) V^{\pi}_{k}(y) \right] \tag{2.8}$$

The discussion of when to stop the algorithm is the same as for value iteration. A different option is to use the Bellman equation directly and solve a set of linear equations to find the value function corresponding to the policy being used. This could not be done so easily for value iteration because the equations are not linear (due to the max operator).

The policy improvement step is accomplished by choosing the action that maximizes the recently updated value function at each state:

$$\pi(x) = \arg\max_{a} \left[r(x,a) + \gamma \sum_{y \in X} p(y|a,x) V^{\pi}(y) \right] \tag{2.9}$$

Considering that the number of deterministic policies over a finite MDP is also finite (in fact the number of policies is equal to $|A|^{|X|}$), and knowing the policy improvement step always strictly improves the policy, there is a bound in the total number of iterations.

Convergence of Dynamic Programming

Looking at the Bellman Optimality equation, we can define an operator $\mathbf{L}$ over functions of type $v : X \to \mathbb{R}$ as:

$$(\mathbf{L}v)(x) = \max_a \left[r(x,a) + \gamma \sum_{y \in X} p(y|a,x)v(y) \right]$$

With this in mind, the Bellman equation can be rewritten as:

$$\mathbf{L}v = v$$

A solution to the Bellman equation is guaranteed to exist and be unique by the Banach Fixed Point Theorem, if $\mathbf{L}$ is a contraction mapping, which can be proved to be true for space of value functions with the $\|\cdot\|_\infty$ norm.

Both the policy value iteration algorithm and the policy improvement step of policy iteration end up being just variations on the application of the previous result. A more detailed explanation can be seen in (Puterman, 1994).

2.2.2 Learning with Model-free methods

The dynamic programming methods presented in the previous section are intended to learn optimal value functions (and from them find optimal policies) in the presence of a model of the system. In reinforcement learning, generally it is assumed the model is unknown and, in this situation, two approaches can be pursued. A *model-based* approach tries to learn the model explicitly and then uses methods like dynamic programming to compute the optimal policy with respect to the estimate of the model. On the other hand, a *model-free* approach concentrates on learning the state value function (or the Q-value function) directly and obtaining the optimal policy from these estimates. Two popular model-free reinforcement learning methods are Q-learning and Sarsa, which will be summarily described.

Figure 2.4 shows the agent/environment loop for reinforcement learning systems.

Q-learning: off-policy control

The *Q-learning* method, proposed by (Watkins, 1989), is perhaps the most popular and widely used form of reinforcement learning, mainly due to the ease of implementation. It is an *off-policy* method which learns optimal Q-values, rather than state-values, and

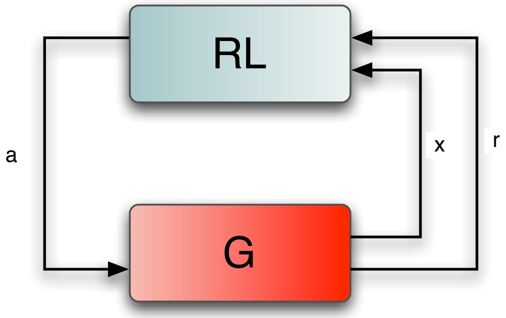


Figure 2.4: Reinforcement learning block diagram.

simultaneously determines an optimal policy for the MDP. The update rule for Q-learning is:

$$Q_{k+1}(x_k, a_k) = Q_k(x_k, a_k) + \alpha_k\big(r_k + \gamma \max_{b\in A} Q_k(x_{k+1}, b) - Q_k(x_k, a_k)\big) \tag{2.10}$$

which is based on the idea that $r_k + \gamma \max_{b\in A} Q_k(x_{k+1}, b)$ is a better approximation for $Q^*(x_k, a_k)$ than $Q_k(x_k, a_k)$. In (Watkins and Dayan, 1992) the method is proven to converge to the Q-values for the optimal policy, Q^*, if two convergence conditions are met:

1. Every state-action pair has to be visited infinitely often.
2. α must decay over time such that $\sum_{k=0}^{\infty} \alpha_k = \infty$ and $\sum_{k=0}^{\infty} {\alpha_k}^2 < \infty$.

Q-learning is in fact a particular case of a broader range of algorithms known as *stochastic approximation algorithms*. In fact, the convergence conditions are general convergence conditions of stochastic approximations theorems, which are used as an alternative proof to (Watkins, 1989) by (Jaakkola *et al.*, 1994). A detailed discussion on stochastic approximation can be read in (Kushner and Yin, 2003).

Sarsa: on-policy control

Sarsa was first introduced by (Rummery and Niranjan, 1994) as an *on-policy* method for learning the optimal policy while controlling the MDP. In this situation, the algorithm behaves according to the same policy which is being improved and, so, the update rule is as follows:

$$Q_{k+1}(x_k, a_k) = Q_k(x_k, a_k) + \alpha_k\big(r_k + \gamma Q_k(x_{k+1}, a_{k+1}) - Q_k(x_k, a_k)\big) \tag{2.11}$$

The method relies on information about the variables $x_k, a_k, r_k, x_{k+1}, a_{k+1}$ and that explains the origin of the name Sarsa (the state is commonly represented as s instead of x), introduced by (Sutton, 1996).

Convergence results can be seen in (Singh *et al.*, 2000*b*) and, basically, require that each state-action pair is visited infinitely often and that the policy being used converges to a greedy policy[5]. Sarsa is another instance of a stochastic approximation algorithm, and the convergence proof cited uses a similar theorem to the proof of Q-learning.

2.2.3 Exploration vs Exploitation

The methods presented previously, when used for learning and control, all share a common characteristic: for the algorithm to converge to the optimal policy, all the possible combinations of state and action have to visited infinitely often. This condition allows the convergence to avoid being stuck on sub-optimal policies.

With an off-policy method this is always achieved in the limit by using *soft* behavior policies, meaning that no action has a 0 probability of being chosen. On-policy methods, on the other hand, require that the policy itself converges to a greedy one, while performing exploration.

In both situations, the dilemma of the learning method is wether to focus on using the already acquired information for control or trying to get new information, which could lead to better policies and a better control. The bottom line is: reinforcement learning algorithms have to act while learning.

A simple option for balancing exploration and exploitation is to have the behavior policy be a mixture of an *exploration policy* and an *exploitation policy* as stated by:

$$\Pi_{behavior} = \Gamma \cdot \Pi_{explore} + (1 - \Gamma) \cdot \Pi_{exploit} \tag{2.12}$$

where $0 < \Gamma < 1$ is a parameter which can be adjusted to allow more exploration or more exploitation. For example, on-policy methods just have to make $\Pi_{exploit}$ a greedy policy and modify so the parameter so that $\Gamma \to 0$ and, in the end, the remaining policy is greedy and is optimal because the corresponding values would also have converged to the optimal ones.

For off-policy methods, the general balancing expression can also be used and Γ may or not be modified, according to the specific needs of the algorithm. This and other considerations regarding exploration are surveyed in (Thrun, 1992).

ϵ-greedy policies

This kind of policies fit in the general description of Equation (2.12) and are basically a mixture of a greedy policy (for exploitation) and a uniform policy (for exploration). The parameter ϵ can be made arbitrarily small and drive the convergence to a greedy

[5] The exact kind of policies needed for the convergence to occur are denominated GLIE policies, which stands for *Greedy in the Limit with Infinite Exploration.* A policy is said to be GLIE if it satisfies the mentioned conditions of visiting each state-action pair infinitely often plus and additional condition:

In the limit, the learning policy is greedy w.r.t. the Q-function with probability 1 (and optimal) policy. Equation (2.13) describes such a policy.

$$\pi(x,a) = \begin{cases} \dfrac{\epsilon}{|A|} & \text{if } a \text{ is a greedy action,} \\ 1-\epsilon+\dfrac{\epsilon}{|A|} & \text{otherwise.} \end{cases} \tag{2.13}$$

Softmax policies

Although ϵ-greedy policies are widely used in reinforcement learning, they have the disadvantage of giving equal weights to the non-greedy actions. Actually, some of them could be performing incredibly better than others, although they are not greedy. A way to avoid this problem is to use an utility function over the actions to better distinguish between them. This kind of methods, called *softmax*, still give the highest probability to the greedy action but do not treat all the others the same way.

In the reinforcement learning context, the Q-function seems like a natural utility to use. The most common softmax method used in reinforcement learning relies on a Boltzmann distribution and controls the focus on exploration through a temperature parameter $\tau > 0$, as defined in Equation (2.14).

$$\pi(x,a) = \frac{e^{Q(x,a)/\tau}}{\sum_{b\in A} e^{Q(x,b)/\tau}} \tag{2.14}$$

Similarly to ϵ-greedy policies, when $\tau \to 0$ the policy becomes greedy making the method adequate for the use with on-policy algorithms.

2.2.4 Semi-Markov Decision Processes

Semi-Markov Decision Process (SMDP) (Howard, 1963; Puterman, 1994) can be considered an extension of MDPs to continuous time and, besides the quantities defined previously, it introduces:

$$F(t|x,a) = P\left[Y < t | X = x, A = a\right]$$

which represents the probability of a state change occurring in less than time t, given the system was in state x and chose action a. Additionally, instead of the reward function $r(x,a)$, two functions are considered:

- $\kappa : X \times A \longrightarrow \mathcal{R}$ is defined as the expected immediate reward the agent gets from executing action a in state x.
- $\mu : X \times A \longrightarrow \mathcal{R}$ is defined as the expected reward rate the agent gets from executing action a for as long as he stays in state x.

As in MDPs, the objective for infinite-horizon discounted problems, in SMDPs, can be written as:

$$V^{\pi}(x) = E\left\{ \sum_{k=0}^{\infty} e^{-\beta\sigma_k} \left[\kappa(x_k,a_k) + \int_{\sigma_k}^{\sigma_{k+1}} e^{-\beta(\tau-\sigma_k)} \mu(x_k,a_k) d\tau \right] \middle| x_0 = x, \pi \right\} \tag{2.15}$$

where π is a policy that assigns actions to states, $\sigma_0, \sigma_1, \cdots$ represent the times where the state of the system changes and a new action is chosen and β is the discount factor. The corresponding Bellman equation for policy π can also be derived (Puterman, 1994):

$$\begin{aligned} V^{\pi}(x) &= E\left\{\sum_{k=0}^{\infty} e^{-\beta\sigma_k} r(x_k, a_k) \middle| x_0 = x, \pi\right\} \\ &= \sum_{b\in A} \pi(x,b)\left[r(x,b) + E\left\{e^{-\beta\tau} V^{\pi}(x_1) \middle| \pi\right\}\right] \\ &= \sum_{b\in A} \pi(x,b)\left[r(x,b) + \sum_{y\in X}\left(\int_0^{\infty} e^{-\beta\tau} H(d\tau, y|x,b)\right) V^{\pi}(y)\right] \end{aligned} \tag{2.16}$$

where $H(dt, x'|x,a) = F(dt|x,a)p(x'|x,a)$ represents the probability of moving from state x to state x' in less than t time, after having taken action a. The notation $F(dt|x,a)$ represents a time differential and, if the density associated with the distribution F exists, $F(dt|x,a) = f(t|x,a)dt$. Additionally:

$$\begin{aligned} r(x,a) &= \kappa(x,a) + \int_0^{\infty}\left[\int_0^{\tau} e^{-\beta t}\mu(x,a)dt\right] F(d\tau|x,a) \\ &= \kappa(x,a) + \mu(x,a)\int_0^{\infty} \frac{1}{\beta}(1 - e^{-\beta\tau})F(d\tau|x,a) \end{aligned} \tag{2.17}$$

and in terms of Q-values:

$$Q^*(x,a) = r(x,a) + \sum_{y\in X} M(y|x,a) \max_{b\in A_x} Q^*(y,b) \tag{2.18}$$

where

$$M(y|x,a) = \int_0^{\infty} e^{-\beta\tau} H(d\tau, y|x,a)$$

An important assumption on the model, that will ensure the convergence of both the dynamic programming and Q-learning algorithms, is that fact that, in any interval of finite length, the probability of infinite state transitions occurring is zero. This is discussed in (Puterman, 1994) and can be expressed in the following way:

Assumption 2.2.1. *There exist* $\epsilon > 0$ *and* $\delta > 0$ *such that*

$$F(\delta|x,a) < 1 - \epsilon$$

$\forall_{x\in X}$ *and* $\forall_{a\in A}$, *where* $F(\delta|x,a) = \sum_{y\in X} H(t,y|x,a)$.

Under the conditions of Assumption 2.2.1, Equation (2.18) will also have a unique solution. In fact, the optimal Q-function is a fixed point of the operator $\mathbf{H}$, defined for functions of type $q : X \times A \to \mathbb{R}$ as:

$$(\mathbf{H}q)(x,a) = r(x,a) + \sum_{y\in X} M(x,a,y) \max_{b\in A_y} q(y,b) \tag{2.19}$$

and this operator is guaranteed to have a unique fixed point since it is a contraction mapping in the sup-norm:

$$\begin{aligned}
&\|\mathbf{H}q_1 - \mathbf{H}q_2\|_\infty = \\
&= \max_{x,a} \left| \sum_{y \in X} M(x,a,y) \left(\max_{b \in A_y} q_1(y,b) - \max_{b' \in A_y} q_2(y,b') \right) \right| \leq \\
&\leq \max_{x,a} \sum_{y \in X} M(x,a,y) \cdot \max_{z,b} |q_1(z,b) - q_2(z,b)| = \\
&= \max_{x,a} \sum_{y \in X} M(x,a,y) \cdot \|q_1 - q_2\|_\infty
\end{aligned}$$

It remains to show that $\max_{x,a} \sum_{y \in X} M(x,a,y)$ is strictly bounded by 1. This condition is met because of the Assumption 2.2.1, in the following way:

$$\begin{aligned}
&\max_{x,a} \sum_{y \in X} M(x,a,y) = \\
&= \max_{x,a} \sum_{y \in X} \int_0^\infty e^{-\beta t} H(dt, y|x,a) = \\
&= \max_{x,a} \int_0^\infty e^{-\beta t} F(dt|x,a) = \\
&= \max_{x,a} \left[\int_0^\delta e^{-\beta t} F(dt|x,a) + \int_\delta^\infty e^{-\beta t} F(dt|x,a) \right] \leq \\
&\leq \max_{x,a} \left[\int_0^\delta F(dt|x,a) + e^{-\beta\delta} \int_\delta^\infty F(dt|x,a) \right] = \\
&= \max_{x,a} \left[F(\delta|x,a) + e^{-\beta\delta}(1 - F(\delta|x,a)) \right]
\end{aligned}$$

and from Assumption 2.2.1

$$\begin{aligned}
&\max_{x,a} \sum_{y \in X} M(x,a,y) \leq \\
&\leq e^{-\beta\delta} + (1-\epsilon)(1 - e^{-\beta\delta}) < 1
\end{aligned}$$

If we set $\gamma^* = e^{-\beta\delta} + (1-\epsilon)(1 - e^{-\beta\delta})$, we have:

$$\|\mathbf{H}q_1 - \mathbf{H}q_1\|_\infty \leq \gamma^* \|q_1 - q_2\|_\infty$$

with $0 \leq \gamma^* < 1$, which proves that operator $\mathbf{H}$ is, in fact a contraction mapping in the sup-norm.

The fact that the operator defined in Equation (2.19) is indeed a contraction mapping in the sup-norm can be used to prove the convergence of the modified Q-learning update rule for SMDPs.

The optimal Q-function can be estimated using a stochastic approximation strategy similarly to the MDP case. We apply the modified Q-learning rule described on (Bradtke and Duff, 1995) for SMDPs:

$$Q_{k+1}(x_k, a_k) = (1 - \alpha_k)Q_k(x_k, a_k) + \alpha_k \tilde{Q}_{k+1}(x_k, a_k) \tag{2.20}$$

with

$$\tilde{Q}_{k+1}(x_k, a_k) = r_k + e^{-\beta\tau_k} \max_{b \in A_{x_{k+1}}} Q_k(x_{k+1}, b)$$

$$r_k = \kappa_k + \frac{1 - e^{-\beta\tau_k}}{\beta}\mu_k$$

where $A_{x_{k+1}}$ represents the subset of actions from which the algorithm can choose when it reaches state x_{k+1}, and τ_k the time it takes to change from state x_k to state x_{k+1}.

Note that, besides a state, an action, a successor state and a reward (x_k, a_k, r_k, x_{k+1}), the experience tuples for the SMDP case also include the time elapsed between those states, which implicitly samples the distribution $F(t|x_k, a_k)$ associated with the state-action pair (x_k, a_k). For this reason, in the SMDP case, the experienced samples used in each step of the modified Q-learning algorithm are (x, a, r_k, τ, x').

The convergence of the update rule from Equation (2.20) can be proven using the following stochastic approximation theorem:

Theorem 2.2.1. *The random process* $\{\Delta_k\}$ *taking values in* $\mathbb{R}^n$ *and defined as*

$$\Delta_{k+1}(x) = (1 - \alpha_k(x))\Delta_k(x) + \beta_k(x)F_k(x)$$

converges to zero with probability 1 under the following assumptions:

- $x \in X$ *where* X *is finite.*
- $\sum_k \alpha_k(x) = \infty$ *and* $\sum_k \alpha_k^2(x) < \infty$*;*
- $\sum_k \beta_k(x) = \infty$ *and* $\sum_k \beta_k^2(x) < \infty$*;*
- $E\{\beta_k(x)|\mathcal{F}_k\} \leq E\{\alpha_k(x)|\mathcal{F}_k\}$ *uniformly over* x *with probability 1;*
- $\|E\{F_k(x)|\mathcal{F}_k\}\|_W \leq \gamma\|\Delta_k\|_W$*, with* $0 < \gamma < 1$*;*
- $var\{F_k(x)|\mathcal{F}_k\} \leq C(1 + \|\Delta_k\|_W^2)$*, for* $C > 0$.

Where $\mathcal{F}_k$ *refers to the past history of the system up to* k *and* $\|\cdot\|_W$ *denotes some weighted maximum norm.*

Proof. See (Jaakkola *et al.*, 1994). □

In fact, this theorem is more powerful than what is needed to prove the convergence of Q-learning. A simpler lemma can instead be used:

Lemma 2.2.2. *The random process $\{\Delta_k\}$ taking values in $\mathbb{R}^n$ and defined as*

$$\Delta_{k+1}(x) = (1 - \alpha_k(x))\Delta_k(x) + \alpha_k(x)F_k(x)$$

converges to zero with probability 1 under the following assumptions:

- *$x \in X$ where X is finite.*
- *$0 \leq \alpha_k(x) \leq 1$, $\sum_k \alpha_k(x) = \infty$ and $\sum_k \alpha_k^2(x) < \infty$;*
- *$\|E\{F_k(x)|\mathcal{F}_k\}\|_W \leq \gamma\|\Delta_k\|_W$, with $0 < \gamma < 1$;*
- *$var\{F_k(x)|\mathcal{F}_k\} \leq C(1 + \|\Delta_k\|_W^2)$, for $C > 0$.*

Where $\mathcal{F}_k$ refers to the past history of the system up to t and $\|\cdot\|_W$ denotes some weighted maximum norm.

Proof. Make $\beta_k(x) = \alpha_k(x)$ for every x and every t and apply Theorem 2.2.1. □

These stochastic approximation results stem from a broader theorem supported in an Ordinary Differential Equations (ODE) method. A thorough discussion of the connections between these results and their implications in learning can be seen in (Melo, 2007). Applying them will allow us to prove the update rule from Equation (2.20) will in fact converge with probability one, to the optimal Q-function (in the discounted formulation). This can be written as a theorem:

Theorem 2.2.3. *Given a finite SMDP (X, A, H, k, c) with discount rate β, the Q-learning sequence $\{Q_k\}$, given by the update rule:*

$$Q_{k+1}(x_k, a_k) = (1 - \alpha_k(x_k, a_k))Q_k(x_k, a_k) + \alpha_k(x_k, a_k)\left(r_k + e^{-\beta\tau_k} \max_{b \in A_{x_{k+1}}} Q_k(x_{k+1}, b)\right)$$

with

$$r_k = \kappa_k + \frac{1 - e^{-\beta\tau_k}}{\beta}\mu_k$$

converges with probability 1 to the optimal Q-function if:

- *Assumption 2.2.1 holds.*
- *$0 \leq \alpha_k(x_k, a_k) \leq 1$, $\sum_k \alpha_k(x_k, a_k) = \infty$ and $\sum_k \alpha_k^2(x_k, a_k) < \infty$;*
- *k_k and c_k are bounded.*

Proof. We start by noting that the notation x_{k+1} is equivalent to the continuous time-based notation x_{t+t_k} if we consider decision epochs. In that case, $n : \mathbb{R}^+ \rightarrow \mathbb{N}$ is a function that matches the continuous time values to the decision epochs, for some run of the algorithm: in essence, if $n(t) = k$ and the next state change occurs after τ_k, then $n(t + \tau_k) = k + 1$.

We now define:

$$\Delta_k(x_k, a_k) = Q_k(x_t, a_k) - Q^*(x_k, a_k)$$

and

$$F_k(x_k, a_k) = r_k + e^{-\beta\tau_k} \max_{b \in A_{x_{k+1}}} Q_k(x_{k+1}, b) - Q^*(x_k, a_k)$$

and subtracting $Q^*(x_k, a_k)$ from both sides of the update rule equation we get:

$$\begin{aligned} Q_{k+1}(x_k, a_k) - Q^*(x_k, a_k) =& (1 - \alpha_k(x_k, a_k))Q_k(x_k, a_k) + \\ &+ \alpha_k(x_k, a_k)\left(r_k + e^{-\beta\tau_k} \max_{b \in A_{x_{k+1}}} Q_k(x_{k+1}, b)\right) - Q^*(x_k, a_k) \\ \Delta_{k+1}(x_k, a_k) =& (1 - \alpha_k(x_k, a_k))\Delta_k(x_k, a_k) + \alpha_k(x_k, a_k)F_k(x_k, a_k) \end{aligned}$$

The update rule now reduces to the form presented in Lemma 2.2.2. It remains to be shown that the conditions required by that result are met for this case. The first two conditions trivially hold by assumption, so the last two conditions must be verified.

- $E\{F_k(x_k, a_k)|\mathcal{F}_k\} = E\{r_k(\tau_k)|\mathcal{F}_k\} + E\{e^{-\beta\tau_k} \max_{b \in A_y} Q_k(y, b)|\mathcal{F}_k\} - Q^*(x_k, a_k)$. From Equation (2.17) we know that:

$$\begin{aligned} E\{r_k|\mathcal{F}_k\} &= E\left\{\kappa_k + \mu_k \int_0^\infty \frac{1}{\beta}(1 - e^{-\beta\delta})F(d\delta|x, a)\mathcal{F}_k\right\} \\ &= E\{\kappa_k|\mathcal{F}_k\} + E\{\mu_k|\mathcal{F}_k\} \int_0^\infty \frac{1}{\beta}(1 - e^{-\beta\delta})F(d\delta|x, a) \\ &= \kappa(x_k, a_k) + \mu(x_k, a_k) \int_0^\infty \frac{1}{\beta}(1 - e^{-\beta\delta})F(d\delta|x, a) \end{aligned}$$

  And again from Equation (2.17) we can write:

$$\begin{aligned} E\{r_k|\mathcal{F}_k\} &= \kappa(x_k, a_k) + \mu(x_k, a_k) \int_0^\infty \frac{1}{\beta}(1 - e^{-\beta\delta})F(d\delta|x, a) \\ &= r(x_k, a_k) \end{aligned}$$

  As for the expected value of successor state, following a greedy policy, we can write:

$$\begin{aligned} E\{e^{-\beta\tau_k} \max_{b \in A_{x_{k+1}}} Q_k(x_{k+1}, b)\} &= \sum_{y \in X} \int_0^\infty e^{-\beta\delta} \max_{b \in A_y} Q_k(y, b) H(d\delta, y|x_k, a_k) \\ &= \sum_{y \in X} M(x_k, a_k, y) \max_{b \in A_y} Q_k(y, b) \end{aligned}$$

  and from this we can see that $E\{F_k(x_k, a_k)|\mathcal{F}_k\} = (\mathbf{H}Q_k)(x_k, a_k) - Q^*(x_k, a_k)$. Using the fact that the optimal function is a fixed point of the operator $\mathbf{H}$, as seen before, we have:

$$E\{F_k(x_k, a_k)|\mathcal{F}_k\} = (\mathbf{H}Q_k)(x_k, a_k) - (\mathbf{H}Q^*)(x_k, a_k)$$

and it follows that:

$$\|E\{F_k(x_k, a_k)|\mathcal{F}_k\}\|_\infty \leq \gamma^* \|Q_k(x_k, a_k) - Q^*(x_k, a_k)\| = \gamma^* \|\Delta_k(x_k, a_k)\|_\infty$$

proving the condition holds, since it was shown before how to obtain γ^* and that this value would always be strictly inferior to 1, as long as we are within the conditions of Assumption 2.2.1, which is achieved by assumption.

- As for the variance condition

$$\begin{aligned} &var\{F_k(x_k, a_k)|\mathcal{F}_k\} = \\ &= E\left\{\left(r_k + e^{-\beta\tau_k} \max_{b\in A_y} Q_k(y, b) - Q^*(x_k, a_k) - (\mathbf{H}Q_k)(x_k, a_k) + (\mathbf{H}Q^*)(x_k, a_k)\right)^2 |\mathcal{F}_k\right\} = \\ &= E\left\{\left(r_k + e^{-\beta\tau_k} \max_{b\in A_y} Q_k(y, b) - (\mathbf{H}Q_k)(x_k, a_k)\right)^2 |\mathcal{F}_k\right\} = \\ &= var\{r_k + e^{-\beta\tau_k} \max_{b\in A_y} Q_k(y, b)|\mathcal{F}_k\} \end{aligned}$$

Considering we assume the rewards μ_k and κ_k to be bounded, so will be r_k[6], and that will make the last condition of the theorem hold, which concludes the proof.

□

A similar proof was first presented in Ronald Parr's PhD thesis (Parr, 1998*a*). In our proof we did not rely on the factorization of function $H(t, y|x, a) = p(y|x, a)F(t|x, a)$ since, as it will be shown in Chapter 4, that is not always the case for the model proposed. Additionally, although strictly not in the proof, we have described how Assumption 2.2.1 guarantees that Bellman operator is contractive and not just a short map[7]. We closely followed the proofs in (Jaakkola *et al.*, 1994) and (Melo, 2007) for discrete-time Q-learning.

2.2.5 Partially Observable Markov Decision Processes

MDPs and SMDPs, as described previously, assume that it is possible to have a complete access to the state of the system at each decision point. While for some applications this approach might suffice, it is not always possible to have a complete and unequivocal knowledge about the state of the system and generally it is only possible to try to

[6] In fact, even if the time between decisions is very large, the discount factor guarantees that the part of the reward due to μ_k remains bounded.

$$\lim_{\tau\to\infty} \kappa + \frac{1 - e^{-\beta\tau}}{\beta}\mu = \kappa + \frac{\mu}{\beta}$$

[7] A *short map*, *weak contraction* or *non-expansive map* is one that satisfies a Lipschitz condition with $K = 1$. See Appendix A.

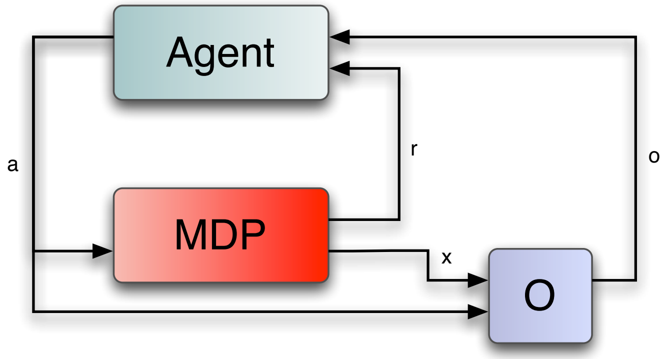


Figure 2.5: Agent acting in a POMDP

estimate it from specific signals that hint on what the state might be. The framework of *Partially Observable Markov Decision Processes* (POMDP) (Aström, 1965; Aoki, 1965; Sondik, 1971; Kaelbling *et al.*, 1998) models this sort of problem where the state is not directly accessible, even though there is still an underlying Markovian process regulating the dynamics of the system.

POMDPs can be defined as a tuple (X, A, O, p, p_O, r) where X, A, p and r have the same meaning as in MDPs and:

- O is an observation set.
- $p_O : O \times X \times A \longrightarrow [0, 1]$ is an observation model defined as a probability distribution over the observations. Hence, we have

  $$p_O(o|x, a) = P\left[O_{k+1} = o|X_{k+1} = x, A_k = a\right]$$

  where X_{k+1} is a random variable (r.v.) representing the state of the process at time $k+1$, O_{k+1} is a r.v. that represents the observation made at time $k+1$ and A_k is an r.v. representing the action taken at time k.

An agent acting in an environment modeled as a POMDP does not have a direct access to the state of the system. Instead, it only receives observation signals from the set O and has to base his decision of what action to take in the history of actions and observations. The underlying process is still Markovian but the observations themselves, in the general case, do not provide a Markov signal that the agent can use to support the dynamic programming algorithms (or in a stochastic approximation based version, the reinforcement learning algorithms) that were accessible directly in a MDP setting. Figure 2.5 shows how an agent receives signals and interacts with the environment, in a POMDP.

Even though the observations are not Markovian and there is no direct access to the state, it is possible to construct an infinite-space MDP from the POMDP, which will support the derivation of optimality equations. A key concept in this transformation is that of *belief*, which is nothing more than a probability distribution over the states of the POMDP. Put another way, the belief at instant k is:

$$b_k(x) = P\left[X_k = x | \mathcal{F}_k\right]$$

for each state in X. $\mathcal{F}_k$ represents the history of the process up to instant k.

The belief can be updated from another belief and the model parameters using Bayes' rule.

$$b^{ao}(y) = \frac{1}{\eta}\, p_O(o|y,a) \sum_{x \in X} p(y|x,a) b(x) \tag{2.21}$$

where η is a normalizing constant, also given by:

$$\eta = \sum_{y \in X} p_O(o|y,a) \sum_{x \in X} p(y|x,a) b(x)$$

The belief can be thought off as a vector, for some ordering of state X that defines the index associated with each state, and because it is a probability distribution over the states the belief space will be a subset of $[0,1]^N$ with $N = |X|$. Particularly, it will be a simplex Δ^N living in that hypercube:

$$\Delta^N = \left\{ b \in \mathbb{R}^N | \sum_{i=0}^{N} b_i = 1 \right\}$$

This simplex will be the state space for the infinite-state MDP associated with the POMDP. The transition function, which we will denote as p_Δ to differentiate from the POMDP one, is given by:

$$\begin{aligned} p_\Delta(b'|b,a) &= \sum_{o \in O} P\left[B' = b' | B = b, A = a, O = o\right] \;\; P\left[O = o | A = a, B = b\right] \\ &= \sum_{o \in O} \mathbf{1}_{\{b^{ao}\}}(b') \sum_{y \in X} p_O(o|y,a) \sum_{x \in X} p(y|x,a) b(x) \end{aligned}$$

where $\mathbf{1}_{\{b^{ao}\}}$ is an indicator function with $\{b^{ao}\}$ being a singleton including the belief obtained by updating b after executing a and observing o, using Equation (2.21). Finally, a reward function for the infinite-state MDP can also be obtained, with:

$$r_\Delta(b,a) = \sum_{x \in X} b(x) r(x,a)$$

and the MDP will be the tuple $(\Delta^N, A, p_\Delta, r_\Delta)$.

As in finite state MDPs, a policy over the beliefs can be defined and a Bellman equation derived, as in Section 2.2.

$$\begin{aligned} V^\pi(b) &= E\left\{ \sum_{k=0}^{\infty} \gamma^k r_\Delta(b_k, a_k) \middle| b_0 = b, \pi \right\} \\ &= \sum_{a} \pi(b,a) \left[r_\Delta(b,a) + \gamma \int_{\Delta^N} p_\Delta(b|a,b') V^\pi(b') db' \right] \end{aligned} \tag{2.22}$$

And the corresponding optimal equation given by:

$$V^*(b) = \max_a \left[r_\Delta(b,a) + \gamma \int_{\Delta^N} p_\Delta(b|a,b')V^*(b')db' \right] \tag{2.23}$$

Since a policy in the infinite-state MDP is a map over a non-countable state space, it is clear that computing the policy for every belief is generally not possible. However, the MDP associated with the POMDP has specific properties that will allow for methods to find solutions for the optimal equation using value iterations algorithms. The optimality equation can also be written as:

$$V^*(b) = \max_a \left[r_\Delta(b,a) + \gamma \sum_{o\in O} p_O(o|a,b)V^*(b^{ao}) \right] \tag{2.24}$$

where b^{ao} is given by the bayesian update rule of Equation (2.21).

If instead of considering the planning problem over an infinite horizon, we consider the finite horizon one, it can be shown (Sondik, 1971; Smallwood and Sondik, 1973; Cassandra, 1998) that every n-step value function is *piecewise linear and convex* (PWLC) and the infinite horizon value function can be arbitrarily approximated by a PWLC function. Essentially, every intermediate n-step value function can be given by:

$$V_n(b) = \max_{\alpha\in\mathcal{A}_n} b\cdot\alpha \tag{2.25}$$

with the number of vectors in $\mathcal{A}_n$ being finite for every finite horizon value function. In fact, if we consider the immediate decision problem, the number of vectors will be limited to the number of actions.

$$V_0^*(b) = \max_{a\in A} b\cdot r_a$$

where $r_a(i) = r(x_i,a)$ for the same ordering of X used in the beliefs. Essentially, $\mathcal{A}_0 = \{r_a\}_A$ and each belief will have an optimal action which is the one corresponding to the vector α that maximizes the value function at that particular belief. In general, if the problem has a more extensive planning horizon, the number of vectors will still be bounded by all the possible action-observation sequences in that horizon. Since the planning horizon is finite, the action set is finite and the observation set is finite the number of vectors will also be finite. The policy-tree representation explained in (Kaelbling *et al.*, 1998) reflects this idea of finite planning options in a finite horizon, and relates it to the finite nature of the vector set and the PWLC aspect of the optimal value function in that finite horizon.

Following the derivation described in (Spaan, 2006), Equation (2.24) can be combined with Equation (2.25) and the expression for the belief update to obtain:

$$\begin{aligned} V_{n+1}(b) &= \max_a \left[r_\Delta(b,a) + \gamma \sum_{o \in O} p_O(o|a,b) \max_{\alpha \in \mathcal{A}_n} b^{ao} \cdot \alpha \right] \\ &= \max_a \left[b \cdot r_a + \gamma \sum_{o \in O} p_O(o|a,b) \max_{\alpha \in \mathcal{A}_n} b^{ao} \cdot \alpha \right] \\ &= \max_a \left[b \cdot r_a + \gamma \sum_{o \in O} \max_{\alpha \in \mathcal{A}_n} \sum_{y \in X} p_O(o|y,a) \sum_{x \in X} p(y|x,a) b(x) \alpha(y) \right] \end{aligned} \tag{2.26}$$

and so we can say that

$$V_{n+1}(b) = \max_a \left[b \cdot r_a + \gamma \sum_{o \in O} \max_{\{g^i_{ao}\}_i} b \cdot g^i_{ao} \right] \tag{2.27}$$

with

$$g^i_{ao}(x) = \sum_{y \in X} p(o|y,a)\, p(y|x,a) \alpha^i(y) \tag{2.28}$$

for every vector $\alpha^i \in \mathcal{A}_n$.

Finally, since $\max_j b \cdot a_j = b \cdot \arg\max_j b \cdot \alpha_j$, we can define a *backup* function for each belief point given by:

$$\text{backup}(b) = \arg \max_{\{g^b_a\}_{a \in A}} b \cdot g^b_a \tag{2.29}$$

with

$$g^b_a = r_a + \gamma \sum_{o \in O} \arg \max_{\{g^i_{ao}\}_i} b \cdot g^i_{ao} \tag{2.30}$$

The backup vector for belief b essentially represents the hyperplane that maximizes the value function for that belief, after having received information from taking action a and receiving observation o. Essentially, the inner product of the backup and the corresponding belief will give the value of V_{n+1} at that belief point:

$$V_{n+1}(b) = b \cdot \text{backup}(b)$$

Considering the finite horizon value functions are PWLC as discussed before, and can be described by a finite number of vectors, the challenge in constructing a value iteration algorithm for POMDPs consists in determining the regions that share the same maximizing vector. Once a region is determined, every belief point inside that region will produce the same backup(b) vector. A belief point b is called a witness point for the region where backup(b) is the maximizing vector. There is a large body of work on exact algorithms to solve POMDPs. Some algorithms focus on identifying the regions for each step of the algorithm, such as (Sondik, 1971)'s One-Pass algorithm, (Cheng, 1989)'s Linear Support algorithm or the popular Witness algorithm (Littman, 1994).

Alternatively, other algorithms focus on spanning a set of possible vectors and then prune useless ones. The idea of pruning is that, in most problems, not all vectors corresponding to a given action strategy (or policy tree in the representation used by (Kaelbling *et al.*, 1998)) will ever be the maximizing vector in some region of the belief space. For this reason, there is no need to take these vectors into account when calculating all the possible g_{ao} vectors at a given step of the algorithm or, consequently, performing the backup operation. Vectors that are not associated with any region of the belief space are said to be *dominated*. Examples of prunning algorithms include (Monahan, 1982)'s Enumeration algorithm, which was also mentioned in (Sondik, 1971), and Incremental Prunning (Cassandra *et al.*, 1997).

Apart from exact algorithms, it is also possible to approximate the dynamic programming operator and obtain approximate methods that generally have the advantage of being faster and/or performing better for larger state space. An approach to approximate dynamic programming for POMDPs is that of (Lovejoy, 1991) or more recently the following works of (Pineau *et al.*, 2003; Spaan and Vlassis, 2004).

2.3 Summary

In this chapter, we described the basic frameworks and models that support the work in this thesis.

We started by addressing Discrete Event Systems in general and particularly the model of Finite State Automata, as well as some common operations on automata such as the product and parallel composition. The classical framework of Supervisory Control of DES, proposed by (Ramadge and Wonham, 1984), was then described. The key idea of this theory is that there are events in a system that can be enabled/disabled by a supervisor, which can steer the behavior of the system, by keeping it bounded within the limits imposed by the supervisor's specifications. We also considered the addition of time to DES models and described Stochastic Timed Automata, where the occurrence of event is governed by time cumulative distribution functions associated with each event.

We then described Markov Decision Processes (Puterman, 1994), which are the support model for most of the reinforcement learning work, and the optimality equations associated with MDPs. Two classical dynamic programming algorithms were presented, as well as two on-line reinforcement learning ones. Furthermore, the extension of MDPs to continuous time, Semi-Markov Decision Processes, was described and the modified optimality equations were presented. A more detailed discussion on the convergence of learning methods was made for SMDPs.

Still in the field of Markov Decision Processes (Kaelbling *et al.*, 1998), the chapter closes by making an overview of Partially Observable MDPs and their associate optimality equations. The idea is that the state is no longer accessible by the agent who, instead, receives observations that, according to an observation model, can provide information about the state of the system. The concept of *belief*, a probability distribution over the states of the system, becomes a key concept and the optimality equations are written in terms of the belief.

Chapter 3

Related Work

3.1 Supervisory Control of Discrete Event Systems

The problem of controlling Discrete Event Systems, i.e. ensuring that the sequence of events produced by a given system is within specifications, had been addressed by different methods up until the introduction of Supervisory Control in (Ramadge and Wonham, 1984) – a consequence of the thesis work presented in (Ramadge, 1983) and whose ideas had been previously mentioned in (Ramadge and Wonham, 1982*b*; Ramadge and Wonham, 1982*a*) – but a systematic theory of DES supervisions was missing. With the results briefly described in Section 2.1.2 and extensively described in (Cassandras and Lafortune, 2007) or (Wonham, 1997), an effective theory of control of DES was introduced.

The kind of control mechanism implemented by SC is one that, as described previously, ensures that the possible behaviors, or event strings, generated by the system are within specifications. This makes it an ideal choice to use in this work since it is our goal to combine a controller that effectively chooses the best action possible at any given time with another controller that limits the behaviors to a given subset. It is not the goal of this work to extend the research on Supervisory Control but rather to use its results as a tool to achieve a kind of offline planning that does not specify full plans (although it can do so) but rather planning options for the controller to optimize. The main results of SC are described in (Cassandras and Lafortune, 2007) in detail.

Extending the initial SC idea, in (Wonham and Ramadge, 1988; Lin and Wonham, 1990) the control of a DES was considered in a decentralized way, as shown in Figure 3.1 which essentially corresponds to saying the decentralized supervisor function is the conjunction of all the individual supervisors:

$$S_{dec}(s) = \bigcap_{i=1}^{n} S_i(P_i(s))$$

This idea has been further developed in (Rudie and Wonham, 1992) and the fundamental work on *Decentralized Supervisory Control* (DecSC) surveyed by the same author in (Rudie, 2002). The fundamental difference from DecSC to MSC, as described in Section

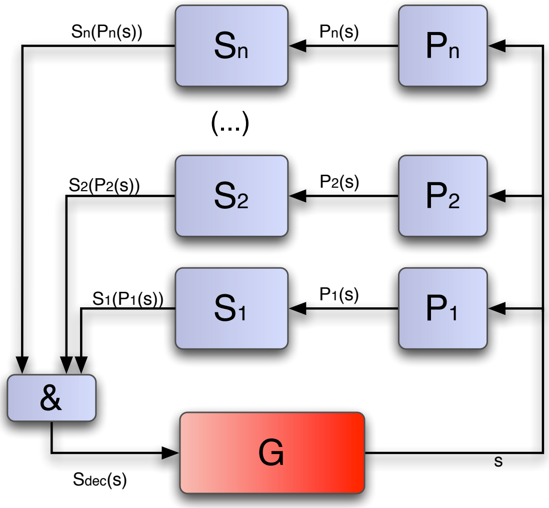


Figure 3.1: Decentralized Supervisory Control block diagram.

2.1.3 is the fact that in the decentralized approach each supervisor can observe a different set of events, and that is exactly what makes DecSC a more challenging problem – how to design and combine supervisors in a way that they can compensate each other's observability faults.

In (Yoo and Lafortune, 2000; Yoo and Lafortune, 2002*b*) a different way to combine the individual supervisors is explored – instead of a *conjunctive* approach they admit the supervisors might be combined by union instead of intersection, which corresponds to:

$$S_{dec}(s) = \bigcup_{i=1}^{n} S_i(Pi(s))$$

if all the supervisors are combined by union. The conjunctive architecture corresponds to focusing on a set of events to enable, meaning that all the supervisors have to agree on the events to enable, while the disjunctive architecture focuses instead on which events to disable, meaning that all the supervisors must disable the same events for them not to be a part of $S_{dec}(s)$.

In (Yoo and Lafortune, 2004) this idea is further explored based on the work presented in (Yoo and Lafortune, 2002*a*), and the logic functions used to combine the supervisors are assumed to be more complex, particularly allowing for conditional decisions where one supervisor might enable (or disable) an event if none of the others enables (or disables) it. Furthermore, it is shown that the more general conditional architecture can achieve a larger class of languages by appropriate choice of the combination function.

Our work in this thesis can also be extended to explicitly consider decentralized supervisors and controllers, and such an extension will necessarily be closely tied to the previously mentioned works on DecSC.

Other extensions of the DecSC approach include that of (Rudie *et al.*, 2003), where an agent P function does not allow him access to a certain set of unobservable events but it can be informed about their occurrence by the other agents through a direct communication function, or the work of (Moor and Raisch, 2002) where discrete supervisors are used to control continuous processes modeled as Ordinary Differential Equations.

3.1.1 Probabilistic Supervisory Control

After the introduction of the Supervisory Control framework, and eventually its decentralized version, some authors extended it by considering supervisors that were not deterministic, i.e., with the enabling and disabling of events being governed by some stochastic process rather than deterministic rules (usually represented, or realized, by a FSA).

In (Hennet, 1993), the SC approach is applied to the control of *Semi-Markov Chains* (SMC), bearing resemblance to the work done in this thesis. The author's idea was to create a *Semi-Markov Decision Process* (SMDP) where the actions consist of removing specific transitions from the underlying chain in order to prevent it from reaching undesirable/unsafe states. It is still very much rooted in SC since it aims not to find optimal paths within the chain but to obtain a supervised chain that is not susceptible to reach specific forbidden states. Figure 3.2 explains the idea graphically.

One of the differences from this approach to our own is that our adaptive part of the system is intended to find the optimal path in the supervised DES we are trying to control. The other difference is that we explicitly consider the notion of event and assume that all information about the system state has to be extracted from the string of events produced by the system since there is no notion of a direct access to the state in our formulation[1].

In the same year, (Lawford and Wonham, 1993) introduced another extension to the SC formalism meant to control *Probabilistic Discrete Event Systems* (PDES), which are similar to STA in form but where the events are not generated by a clock structure but by a matrix of emission probabilities, i.e., there is no explicit notion of time in such models (see Chapter 6 for further reference on PDES). This work has been continued in (Kumar and Garg, 1998; Kumar and Garg, 2001; Postma and Lawford, 2004) and was recently summarized in (Pantelic *et al.*, 2009). The main idea is that not only is the system modeled by a PDES but the supervisor itself issues probabilistic control actions, with events being enabled or disabled according to a given probability measure.

With deterministic supervisors like the ones detailed in (Cassandras and Lafortune, 2007) and briefly described in Section 2.1.2, the supervisor function $S : \mathcal{L}(G) \rightarrow 2^E$ attributes to each possible string generated by the system a subset of the event set E

[1] Of course, it is always possible to consider events that univocally identify the state that has been reached, and in a way this is what happens if we try to model a classical MDP as described in Section 2.2 with an event-based model.

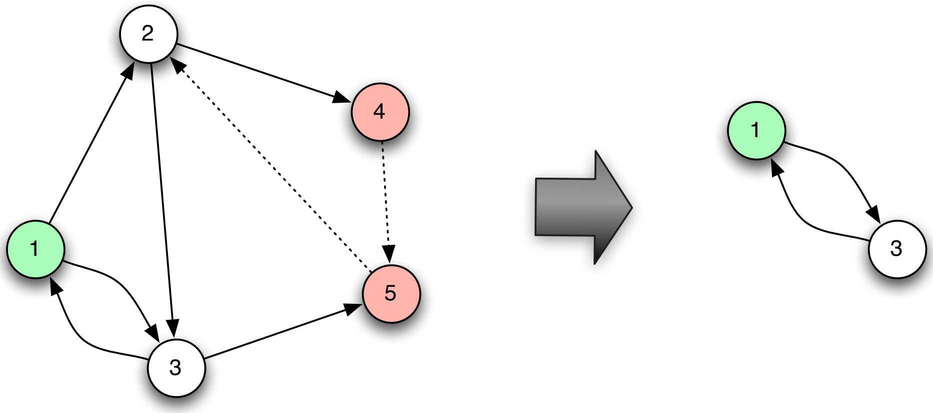


Figure 3.2: Supervisory Control of a Markov Chain.
The initial state is $\{1\}$, the forbidden states are $\{4, 5\}$ and the controllable arcs are $\{(1, 2), (1, 3), (2, 3), (2, 4), (3, 1), (3, 5)\}$. There are several solutions that respect the safety constraints but the proposed one optimizes the cost of cutting arcs, for some cost function.

of events to enable. Put another way, if we define a function:

$$V : \mathcal{L}(G) \to [0, 1]^E$$

that essentially does $V(s)(e) = P[$ Enabling event e after string s $]$, we have:

$$V(s)(e) = \begin{cases} 1 & e \in S(s) \\ 0 & \text{otherwise} \end{cases}$$

In the case of a probabilistic supervisor, this function changes to:

$$V(s)(e) = \begin{cases} 1 & e \in E_{uc} \\ x(s)(e) & \text{otherwise, with } x(s)(e) \in [0, 1] \end{cases}$$

Essentially, the only events that are always enabled w.p.1 are uncontrollable ones, or the supervisor would not be admissible. The others are enabled with probability given by $x(s)(e)$ and so, for the same string produced by the system, the enabling of a given event is not deterministic.

In our approach we assume deterministic supervisors but, nevertheless, it would be possible to combine a probabilistic supervisor with our controllers. The challenges arise from that fact that, as it will be shown in Chapter 4, the action sets for the controller depend on the supervisor action and if the supervisor action is not deterministic not only do the action sets change with the state of the supervisor but they might also be different for the same state of the supervisor, creating an additional source of uncertainty that the controller would have to deal with. In either case, since the class of languages achieved by a probabilistic supervisor is different than that of a deterministic one, it would be particularly interesting to study such combination of probabilistic supervisor and reinforcement learning based controller, although that is out of the scope of this work.

In Chapter 6, we use PDES not as a model of the system of the supervisor, but as the supporting model for a probabilistic observer.

3.1.2 Learning Supervisors

An interesting combination of SC of DES and reinforcement learning is the one that was introduced in (Yamasaki and Ushio, 2003) and further consolidated in (Yamasaki and Ushio, 2005). The key idea of that work is to construct optimal supervisors using reinforcement learning, with the reward function being associated with enabling or disabling events. The advantage of such procedure is that the specifications of a supervisor can then be relegated to defining cost measures for the construction of the supervisor, which in some situations might prove less tedious than describing the full supervisor function.

In (Yamasaki *et al.*, 2005) the approach is combined with the language measure introduced in (Wang and Ray, 2002; Ray, 2005) and it is show how such a language measure is associated with a Bellman equation, with a supervisor being the optimal solution of the equation. The authors apply the approach to a dining philosophers example.

This approach has been studied further on several publications that deal with other extensions of the basic Supervisory Control framework, (Yamasaki and Ushio, 2008; Kajiwara and Yamasaki, 2009; Kajiwara and Yamasaki, June 2010), but these were all published in a japanese conference and are only available in japanese.

3.2 Temporal Abstraction in Reinforcement Learning

Although the SMDP model had been previously defined by some time (Howard, 1963; Puterman, 1994), the need for methods that extended reinforcement learning to include temporally abstract actions stemmed several approaches that, by their own nature, would be supported on SMDPs. Three of these approaches are particularly meaningful: learning with Hierarchies of Machines (Parr, 1998*b*; Parr, 1998*a*; Parr and Russel, 1998), the options framework (Sutton *et al.*, 1999; Precup, 2000) and learning with the Max-Q decomposition (Dietterich, 2000).

In first approach, (Parr and Russel, 1998), the idea was to introduce hierarchical learning by defining subtasks with a non-deterministic finite state controller, called a *Hierarchical Abstract Machine* (HAMs). This approach shares some features with our work since it is the point of the HAM to provide a partial policy at each state, restricting the original choice of actions of the underlying MDP to the ones allowed by the substask being done at the moment. The actions from the associated MDP are often referred to as *primitive actions* and, depending on the level of the hierarchy, a HAM can have as options the execution of a given action or the execution of another HAM corresponding to some subtask of the problem.

On a formal level, a HAM is not much different from a NFA and one of the distinctive parts of the use of the machines for reinforcement learning lies on the interpretation of each state. Essentially, there can be 4 different kinds of states:

Action Executes an action on the environment.

Call Calls another machine as a subroutine.

Choice Chooses the next state according to some policy.

Stop Stops the execution of the current machine and returns to the level of **Call** state that caused it to start.

The decision points correspond to **Choice** states and it is the available choices at each one of these states that will provide the actions for a learning algorithm. And example of modeling with HAMs can be seen in Figure 3.3.

Based on this setup, the authors propose a reinforcement learning update rule referred to as *HAM-Q*:

$$Q(x, m, c) \leftarrow (1 - \alpha)\, Q(x, m, c) + \alpha\, (r_c + \gamma_c V(x', m'))$$

where $x, x' \in X$ are states of the environment, $m, m' \in M$ are states of the machine and c is the decision made at the choice point corresponding to state m of the HAM. This rule is only applied at decision points but the rewards and discount factors need to be update for every change in the environment, with $r_c \leftarrow r_c + \gamma_c r$ and $\gamma_c \leftarrow \gamma\gamma_c$.

With the environment modeled as an MDP and the addition of a HAM to the decision process, the resulting constrained process with decisions restricted to choice states can be proved to be an SMDP and the rule defined by HAM-Q, under the regular conditions on the evolution of α, can be proved to make the value function converge to the optimal value w.p.1, as show on (Parr, 1998*a*). Note that value obtained by the algorithm is generally not optimal from the point of view of an unconstrained system, but it does in fact obtain the best possible decisions when then policy is being partially specified by a HAM.

Another approach to introduce hierarchical learning and temporal abstraction in MDPs are the *options* concept (Sutton *et al.*, 1999; Precup, 2000). The idea is for the programmer to define subtasks by specifying the policy that must be followed while executing such subtasks. These subtasks, called *options* can be formally defined as a tuple $o = (\mathcal{I}, \pi, \beta)$ where:

- $\mathcal{I} \in X$ is an initiation set such that the option o can only be initiated if the system is in a state belonging to $\mathcal{I}$.

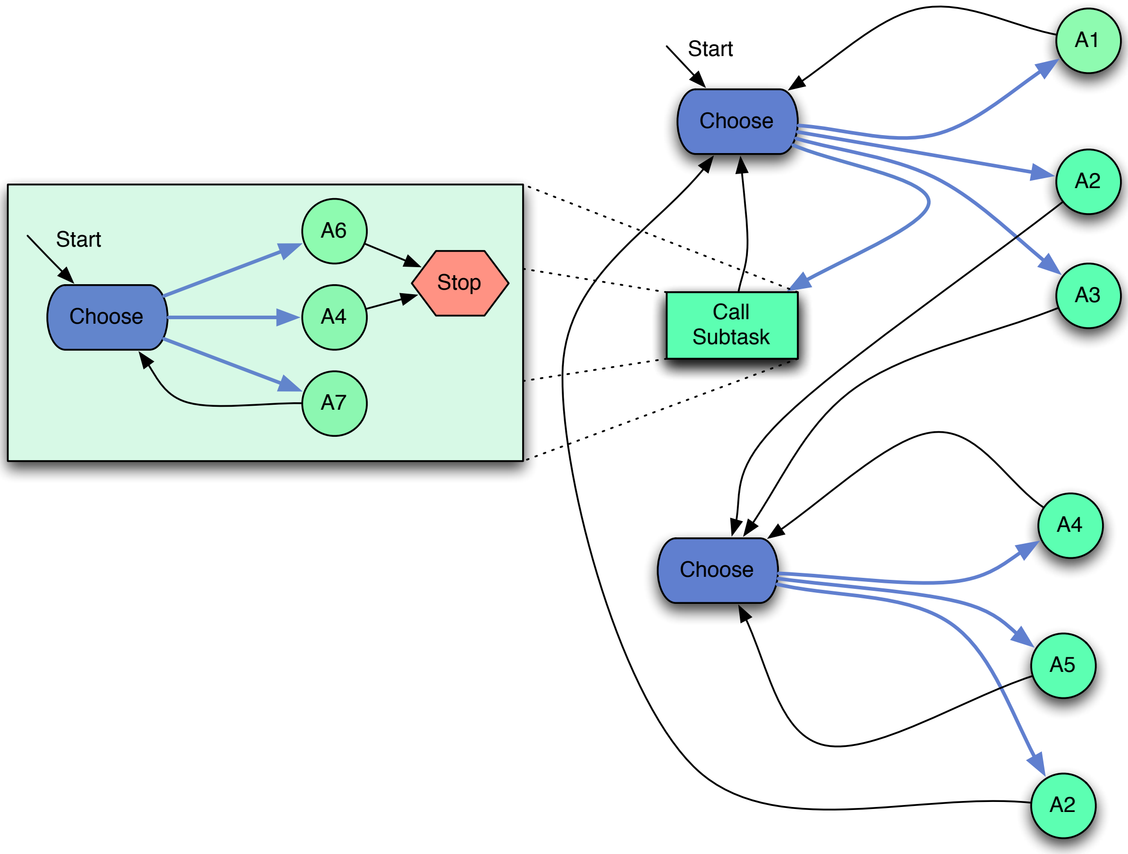


Figure 3.3: Example of a Hierarchical Abstract Machine.
The decisions are made in specific states and the policy to determine which choice to take can be obtained using reinforcement learning.

- $\pi : X \times A \rightarrow [0, 1]$ is a policy to be followed whenever the agent is executing option o.
- $\beta : X \times [0, 1]$ is a stochastic termination condition. At each state, if option o is being executed, it might be terminated with probability $\beta(s)$.

Them simplest kind of options are individual primitive actions, such that the option will terminate after exactly one time step, corresponding to the duration of the associated action. Figure 3.4 shows the differences between the decision over options and over primitive actions. In (Precup, 2000) it is shown that an MDP endowed with a fixed set of options is in fact an SMDP, for which optimality equations can be written as shown in Chapter 2. A Q-learning update rule for the induced SMDP is:

$$Q(x, o) \leftarrow (1 - \alpha)\, Q(x, o) + \alpha\, (r + \gamma^k \max_{o' \in O_{x'}} Q(x', o'))$$

where $x, x' \in X$ are states of the system and $o, o' \in O$ are options. Note that k corresponds to the duration of option o as perceived by the learner; since the termination

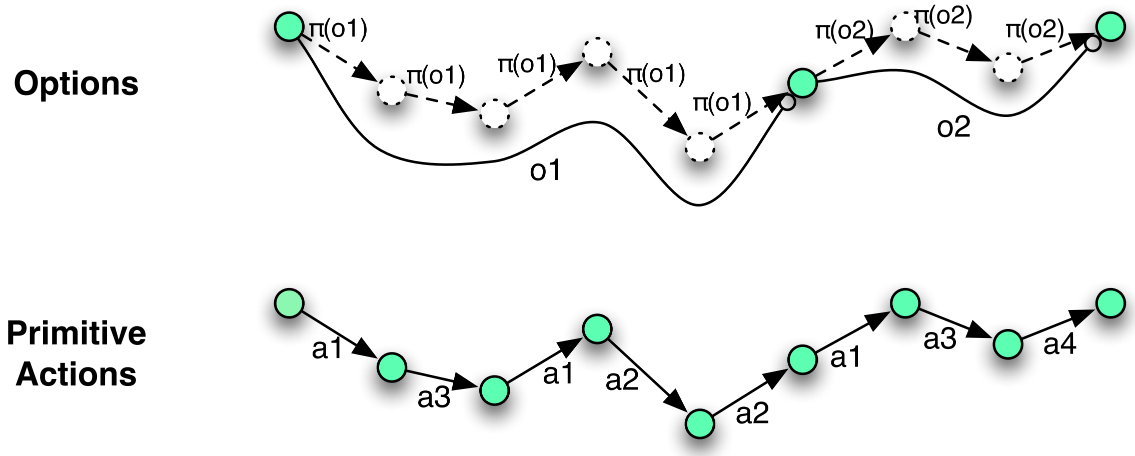


Figure 3.4: Decision making using options.
After an option is started, a new decision after the terminating condition says that the option being run stop. While executing an option, primitive actions are determined by the associated policy.

function is stochastic this duration is not fixed, which is exactly what happens with the actions in an SMDP as defined in Section 2.2.4. The state dependent option set is given by:

$$O_{x'} = \{o \in O : x' \in \mathcal{I}_o\}$$

Additional work on options by the same authors include the possibility of interrupting an option by some mean not part of the termination condition and learning the policy associated with a given option, called *intra-option learning*. Furthermore, (Stolle and Precup, 2002) proposed a method for identifying possible options/subtasks based on empirical state visitation counts and (Wolfe and Singh, 2006) studied how the framework of *Predictive State Representations* (PSRs) could be extended to systems where the decisions consisted of options, and for this reason having a variable duration.

A third approach to implement temporal abstraction and hierarchical learning is that taken by (Dietterich, 1998; Dietterich, 2000). In this approach, each subtask is defined in terms of a termination predicate and additionally there is an associated reward function. A hierarchy of tasks is defined by a graph called the *MaxQ* graph in which the actions available to perform a certain task can either be primitive actions or other subtasks. The termination predicate for a given subtask will consist of a set of states. Figure 3.5 schematizes a hierarchical MaxQ graph. Each task has, for a given state of the environment, the choice between the associated subtasks which might be primitive actions, in the lower level of the hierarchy. Also, it is important to note that not only there are individual value functions for each of the subtasks but they also depend on the caller parent task. In the example of Figure 3.5 both subtasks **S1** and **S5** have

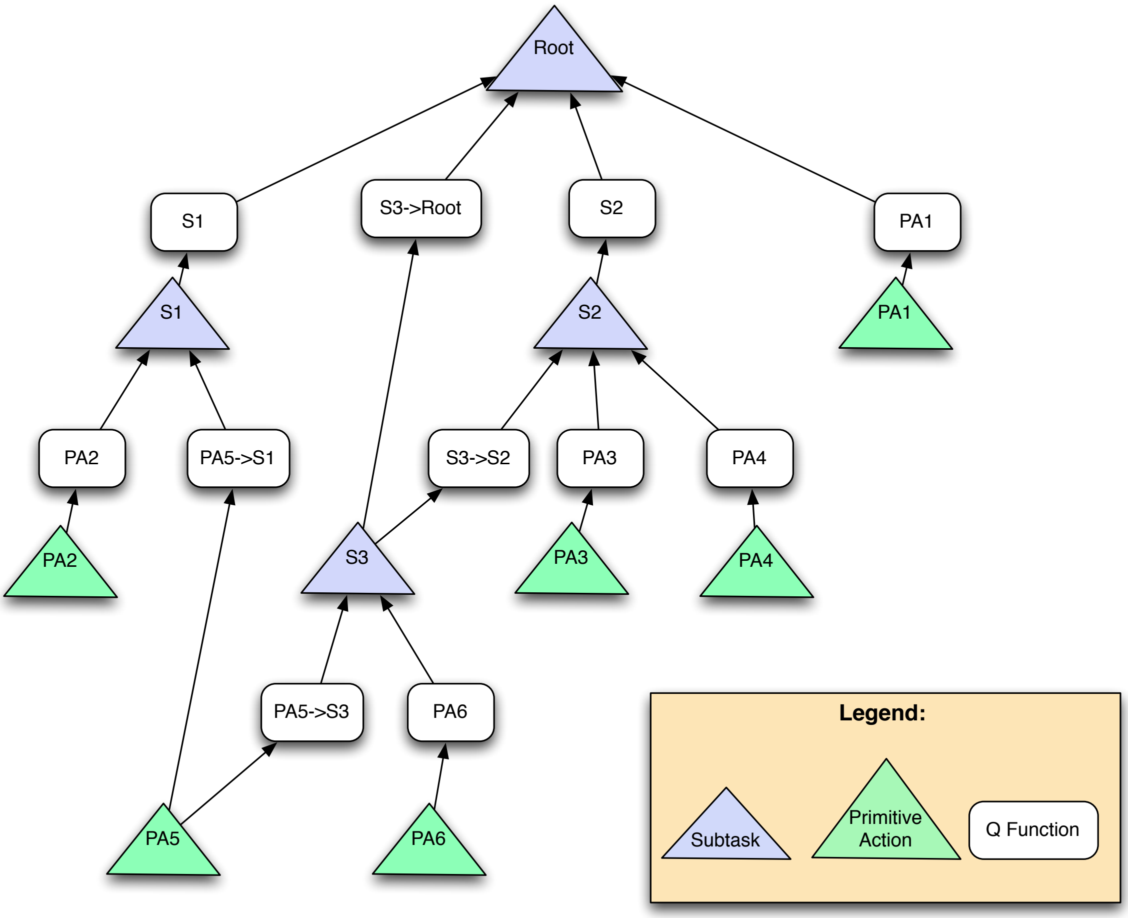


Figure 3.5: MaxQ hierarchical graph for subtasks.
A task can have as subtasks primitive actions or other tasks and will rely on their value functions to compute its own.

the primitive action **PA5** as a possible choice of action and there are separate value functions for when **PA5** is called from either one of the parent tasks.

In all 3 approaches described here the underlying process is a discrete time MDP and the variable duration of each option/subtask origins in the hierarchical nature of the problem. It would be possible to apply these approaches to continuous time MDPs and SMDPs with some modifications but, as they are, even though the resulting process in hierarchical decision is an SMDP, it still has discrete time, which reflects in the discount factor for each option/subtask being given by a power of γ.

On the other hand, in our model continuous time is an assumption from the start that stems from the event-based nature of the model and even without any kind of supervision or eventual hierarchical decomposition that could be considered, the system is already not fully markovian. For this reason, the update rule that we use is similar to both Equation (2.20) and the ones presented here but the discount factor varies with continuous time.

3.3 Reinforcement Learning under Supervision

Combining RL based controllers with supervision has been addressed before, though not too often. The works cited in the previous section achieve some supervision by the definition of subtasks and the available primitive actions in a given subtask. This hierarchical approach is already a way to introduce *a priori* knowledge and limit the number of actions that are available to the agent at any given time, according to the state of the system.

Another approach, proposed in (Rosenstein and Barto, 2004), extends the *actor-critic* model for reinforcement learning (Sutton and Barto, 1998) by using a supervisor to provide a choice of action and an error signal for the learner. Figure 3.6 shows the block diagram of the approach: This approach has been applied in simulation to the control of a robotic arm. Other methods use classical control approaches to restrict the possible actions in each state, typically in systems with continuous states and actions, such as in (Perkins and Barto, 2003) Lyapunov functions or (Morimoto and Doya, 2005) with robust control algorithms.

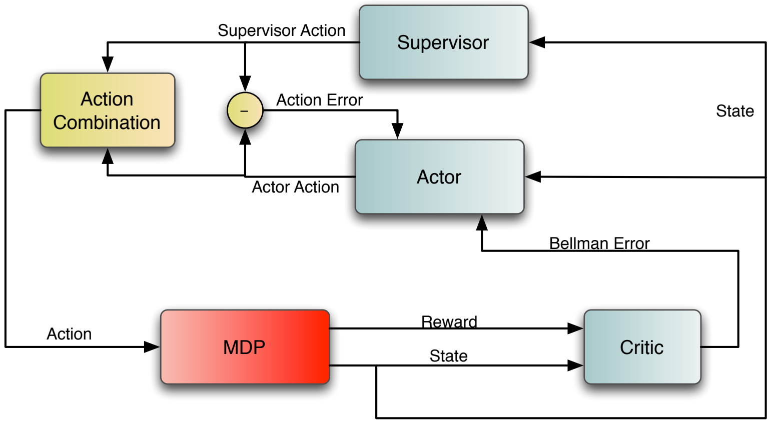


Figure 3.6: Supervised Actor-Critic reinforcement learning.

In (Gabel and Riedmiller, 2008), Reinforcement Learning methods are used for solving Decentralized-MDPs, and the actions of each of the agents have dependencies on the other agents, so in a sense the entire agent collective acts as a supervisor for each of the individual agents. A dependency graph illustrates those dependencies. Additionally, if the action set for agent i is represented as A_i with:

$$A_i = \{\alpha_{i1}, \cdots, \alpha_{ik_i}\}$$

then each agent is assumed to work with an action set A_i^r where:

$$A_i^r = A_i \cap \{\alpha_0\}$$

where α_0 represents an idling action. The idling action is the only one that is always available to each agent, and the one necessarily chosen if none of the others is enabled.

Including idling actions is also an important part of our work, which is particularly relevant when the changes in the environment are considered to be dependent not only on the actions of the agent but on other parallel environmental processes, that might be due to the presence of other agents or intrinsic to the system – sometimes the best thing to do is waiting for something to happen.

3.4 Extensions to POMDPs

3.4.1 Partially Observable Semi-Markov Decision Processes

Although the amount of literature on POMDPs is extensive and has been increasing fast in the last years, particularly with extensions that explicitly consider the presence of several agents and decentralized approaches like Partially Observable Stochastic Games (POSG) (Hansen *et al.*, 2004) or Decentralized POMDPs (Dec-POMDPs) (Bernstein *et al.*, 2000; Nair *et al.*, 2003; Oliehoek *et al.*, 2008), there are not many addressing continuous time semi-markov problems under partial observability.

In (Mahadevan, 1998), the author explains the basic formalism of Partially Observable Semi-Markov Decision Processes (POSMDPs) as an extension of SMDPs (Puterman, 1994) to partially observable problems endowed with an observation model and hints about the possible applications of such framework to distinct areas such as robotics, manufacturing or biological and cognitive behavior modeling. The transition parameters in this model are the same as in SMDPs, as detailed in Section 2.2.4, but the belief update is slightly more complex than the one described by Equation (2.21) if we also take as an observational input the time between a state and its successor.

Knowing it took τ time under the effect of action a to leave state x will influence the belief in the following way:

$$b^{a\tau}(x) = \frac{f(\tau|x,a)b(x)}{\sum_{x\in X} f(\tau|x,a)b(x)} \tag{3.1}$$

where $f(\tau|x,a) = \frac{dF(t|x,a)}{dt}|_{t=\tau}$. Note that this belief refers to the state of the system before the transition as occurred.

The belief about the state of the system after the transition and after having observed o will then be given by the same expression of Equation (2.21) but with the previous belief being given by $b^{a\tau}$ instead of b, i.e.:

$$b^{a\tau o}(y) = \frac{1}{\eta}\, p_O(o|y,a) \sum_{x\in X} p(y|x,a)b^{a\tau}(x) \tag{3.2}$$

where η is a normalizing constant, also given by:

$$\eta = \sum_{y \in X} p_O(o|y, a) \sum_{x \in X} p(y|x, a) b^{a\tau}(x)$$

Put another way, we can write:

$$b^{a\tau o}(y) = \frac{1}{\eta'} p_O(o|y, a) \sum_{x \in X} p(y|x, a) f(\tau|x, a) b(x) \tag{3.3}$$

where η' is a normalizing constant.

Applications of the POSMDP model include (Mahadevan and Khaleeli, 1999) with planning for robust mobile robot navigation or more recently (Jung and Pedram, 2006), which used POSMDPs to develop a Stochastic Dynamic Thermal Management (DTM) technique that aims on regulating voltage and frequency in *Very Large Scale Integration* (VLSI) circuits in order to minimize power dissipation and on-chip temperature. In (Yu, 2006) approximate methods to solve POSMDPs are proposed.

In general, it is possible to adapt the equations underlying most POMDP solvers to work in a semi-markov situation. In Chapter 6 we derive the equations and propose a generic solver for a system that bears some resemblance to a POSMDP but where the transition parameters and the time between states are dependent on events, which will allow the equations to take distinctive aspects w.r.t. the POMDP ones.

3.4.2 Mixed Observability Markov Decision Processes

In POMDPs it is assumed that the only way to have access to an estimate about the state is through the beliefs, which are update based on actions taken and observations made. If, however, we assume there are state variables which are not subject to partial observability and can be directly accessed by the agent, the process can be defined as a *Mixed Observability Markov Decision Process* (MOMDP), which is the approach taken in the recent work by (Ong *et al.*, 2009).

If we assume the state of such a process can be factorized by a set of fully observable and a set of partially observable state variables, with $X = X_p \times X_f$, where X_p corresponds to the partially observable state variables and X_f to the fully observable ones, then it will only make sense to maintain a belief about the partially observable factor of the full state, which we will represent by b_p.

In Chapter 4 we saw that the value function of a POMDP could be represented by a set of vectors, which for a n-step horizon planning yields:

$$V_n(b) = \max_{\alpha \in \mathcal{A}_n} b \cdot \alpha$$

What is proposed in (Ong *et al.*, 2009) by the MOMDP model is that the value function is represented not by a set of vectors but by several ones, one for each of the states in X_f. The value function is then given by:

$$V_n(x_f, b_p) = \max_{\alpha \in \mathcal{A}_n(x_f)} b_p \cdot \alpha$$

where $\mathcal{A}_n(x_f)$ represents the set of vectors corresponding to state x_f.

In our work we independently derived, in Chapter 6, a similar expression that comes from the fact that we use a supervisor to constraint the behavior of the system, and the agent always has full access to the state of the supervisor. In that sense, the supervisor state acts as the X_f and the system state acts as X_p in the MOMDP model. In our case, the transition functions from the supervisor and system are also completely separable, which slightly simplifies the optimality equations.

3.5 Beyond Specifications with Automata

Throughout this thesis we use several kinds of finite automata to model different aspects of the system, ranging from the assumptions about the underlying model of the environment to the agent's modules that observe its output to produce an estimate of the state, and that supervise the system, restricting the possible behaviors of the agent in the environment. From a modeling perspective, finite automata are powerful enough to model systems that produce *regular languages* which are often adequate for a large number of decision-theoretic applications. Nevertheless, even though there are composition operations like the ones described in Section 2.1.1 which allow for parallel representation of different aspects of the system, automata are not always the more expressive option from the point of view of the agent's designer.

A popular option for modeling DES that not only has a higher expressive power, being able to represent a broader class of languages than finite automata in finite space, but usually provides a more intuitive way for programmers to model systems are Petri Nets (PN) (Petri, 1966; Cassandras and Lafortune, 2007). Particularly, a Marked Ordinary Petri Net (MOPN), (Murata, 1989; Cassandras and Lafortune, 2007), can be defined as a tuple $PN = (P, T, I, O, m_0)$ where:

- P is a finite set of places.
- T is a finite set of transitions.
- $I \subset P \times T$ is a set of input arcs from places to transitions.
- $O \subset P \times T$ is a set of output arcs from transitions to places.
- m_0 is an initial marking.

The state of the MOPN is represented by a quantity called a *marking* such that at some time k, the marking is a function $m : P \to \mathbb{N}_0$ that associates a natural number, called the number of *tokens*, to each of the places of the MOPN[2]. The initial marking is given by m_0 and the dynamics of the MOPN are regulated by the transitions: each transition removes tokens from input places and adds tokens to output places. Much of the expressive power of MOPNs comes from the fact that state is represented in

[2] For some ordering of P the marking can also be represented in vector form.

a distributed way, which is particularly useful in problems where there are parallel processes, and the fact that it is possible to add a counter to each place, making the graphical representation of typical counting problems like *birth-death Markov Chains* finite.

Another useful extensions of PNs are *Generalized Stochastic Petri Nets* (GSPNs), (Murata, 1989) which add the possibility of having both transitions that occur immediately and others that take an amount of time regulated by cumulative distribution functions (cdfs). Strictly speaking, in the case of GSPNs the time a non-immediate transition takes to fire is regulated by a Poisson Process and the cdfs are exponential, which makes the system modeled by a GSPN completely markovian. GSPNs are strongly connected to the model of STAs presented in Section 2.1.4 as both model the same kind of stochastic process, under certain conditions.

Recently, in (Costelha and Lima, 2008; Costelha and Lima, 2010) MOPNs and GSPNs have been used to model robotic tasks. The authors take a hierarchical approach to modeling based on 4 levels as represented in Figure 3.7. All the levels of the hierarchy are used for analysis of the controlled system but the execution is done only using some parts of the model. This approach could easily be combined with ours as long as the petri nets used have a finite number of markings, which makes them equivalent to STA. Although this is not strictly the case, roughly speaking the PNs used for *Execution* correspond to our *Controller* and *Supervisor* models that will be defined in Chapter 4 and ultimately, since GSPNs with finite markings have the same expressive power as STAs, it is possible to translate from one to the other, which makes combining the use of PNs with our work fairly straightforward. Particularly, although in (Costelha and Lima, 2010) the focus is on modeling, analysis and execution and not the synthesis

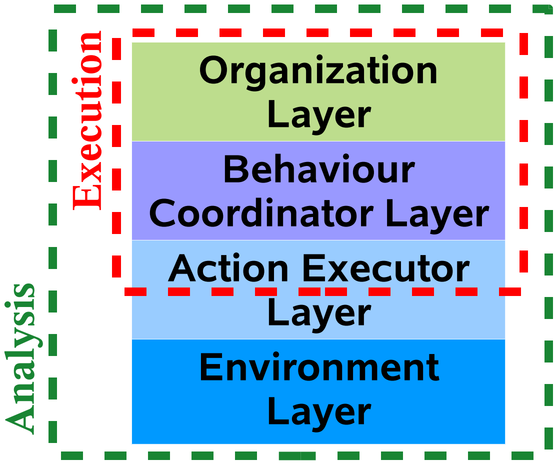
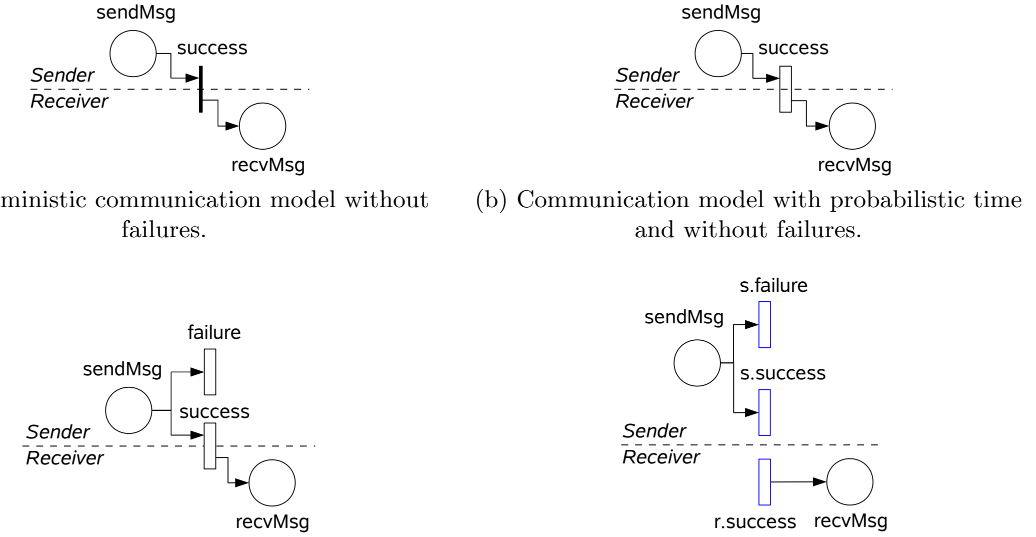


Figure 3.7: Hierarchical representation of robotic task modeling with Petri Nets. As proposed in (Costelha and Lima, 2010) (reproduced with permission from the author). Each of the levels is modeled using Petri Nets.

of PN plans, a possible approach is to associate decisions to immediate transitions where there is a conflict solved probabilistically, generally called *random switches*, which is very similar to what we do in this work, as will be explained in Chapter 4 – if the probabilities of a random switch can be controlled by the agent, it is possible to use reinforcement learning to obtain them. Alternatively, and considering weights can be associated to PN arcs, controlling the weights of some of the transitions is another way to define actions whose choice can also be optimized using reinforcement learning methods.

Another interesting aspect of the modeling approach by the same authors is the explicit modeling of multi robot tasks, particularly how to use PNs to model communication between different robots, in order to enable coordination and synchronization between them. Figure 3.8 shows different models for a communication action. An extension to our work to a distributed multi agent case will necessarily have to rely on coordination and communication events, for which the approach taken in (Costelha and Lima, 2010) is quite suitable.



(a) Deterministic communication model without failures.

(b) Communication model with probabilistic time and without failures.

(c) Full communication model with probabilistic time and failures.

(d) Separate view of the full communication model.

Figure 3.8: Communication models in robotic task modeling with Petri Nets. As proposed in (Costelha and Lima, 2010) (reproduced with permission from the author).

Another recent approach to model specification focuses particularly on the supervisor and how can it be constructed from a specification language closer to natural language. In (Lacerda and Lima, 2008), the authors choose *Linear Temporal Logic* (LTL) as a starting point to constructing a supervisor represented by a DFA. If we denote by Π a set of propositional symbols and $L_{LTL}(\Pi)$ the set of LTL formulas over Π, whose definition is explained more thoroughly in (Lacerda and Lima, 2008), the for two formulas $\varphi, \psi \in$

$L_{LTL}(\Pi)$, there are 5 temporal operators that can be defined:

Next (X): φX meaning that φ will be true until the next state.

Until (U): $\varphi U \psi$ meaning that φ will always be true until ψ becomes true.

Releases (R): $\varphi U \psi$ meaning that ψ must always be true until φ becomes true (or "releases" ψ).

Eventually (F): $F\psi$ meaning that sometime in a future state ψ will be true. Essentially an abbreviation for $true\, U\psi$

Always (G): $G\psi$ meaning that ψ will be true in all future states. Essentially an abbreviation for $false\, R\psi$.

With these operators as well as the extension of the usual propositional logic ones to a temporal setting, LTL offers a powerful way to express complex specifications in a compact way. In (Lacerda and Lima, 2008), after specifications are made using LTL, they are used to build a *Büchi Automaton* (BA) and finally it is explained how this kind of automaton can be used for supervision, particularly since BA are equivalent to NFA and these always have a deterministic equivalent.

3.6 Summary

In this chapter we review literature from several areas that are strongly connected to the work of this thesis.

Supervisory Control of Discrete Event Systems: In this section we briefly mention the key works on Supervisory Control since its introduction by (Ramadge and Wonham, 1984). Particularly, in more recent years, the work of (Yoo and Lafortune, 2004) establishes new ways to combine supervisor actions. Probabilistic Supervisory Control is an extension of the initial SC framework that has had some development, initially in the control of Markov Chains by (Hennet, 1993), and more recently through the use of Probabilistic Discrete Event Systems to implement probabilistic supervisors by (Pantelic *et al.*, 2009) after a line of work beginning in (Lawford and Wonham, 1993).

Using reinforcement learning to compute optimal supervisors for a given system, according to a language measure, is a recent approach studied initially in (Yamasaki and Ushio, 2005) and further developed in following works.

Temporal Abstraction in Reinforcement Learning: This section is closely related to learning in Semi-Markov Decision Processes and mainly addresses 3 different ways to implement temporally abstract actions in MDPs: learning with Hierarchical Machines (Parr, 1998*a*), learning with options (Sutton *et al.*, 1999) and the Max-Q decomposition(Dietterich, 1998). Each of these approaches creates a discrete time SMDP by abstracting over an MDP. We arrive similar results for our framework but in continuous time and starting from an event-based system.

Reinforcement Learning under Supervision: Combining a reinforcement learning algorithm with a supervisor has been done in different areas, usually having specific applications in mind. (Rosenstein and Barto, 2004) propose a Supervised Actor-Critic architecture, while the works (Perkins and Barto, 2003) and (Morimoto and Doya, 2005) are closely related to using classical control results to supervise reinforcement learning. In (Gabel and Riedmiller, 2008), a multi-agent scenario is studied where actions are restricted through dependencies from other agents.

Extensions to POMDPs: The extension of POMDPs to continuous time is explored by (Mahadevan, 1998), whose results are closely related to the equations we derive in Chapter 6, although the fact that our framework is event-based gives it unique properties. Some applications of POSMDPs are reffered. The recent framework of Mixed Observability Markov Decision Processes, (Ong *et al.*, 2009), studies situations where the state spaces can be factored in a fully observable part and a partially observable one. Our framework in Chapter 6 includes mixed observability since the state of the supervisor is always fully observable.

Beyond Specifications with Automata: Novel ways to model the system and provide specifications to a supervisor are reviewed in this section. (Costelha and Lima, 2010) uses Petri Nets to model robotic tasks while in (Lacerda and Lima, 2008) Linear Temporal Logic is used as a language for supervisor specification.

Additionally to the work mentioned, there has been an increasing trend in studying models and methods that integrate planning, learning and control. Particularly, the *1st International Workshop on Hybrid Control of Autonomous Systems*, (Ferrein *et al.*, 2009), was dedicated exactly to this theme.

Chapter 4

Full Observability

When choosing the sequence of actions for an agent, a major drawback of pre-programmed plans is the fact that they are not prepared to deal with unforeseen changes in the environment. Additionally, usually there is a need for complete environment models in order to be able to compute plans *a priori*. Reinforcement learning, and particularly model-free reinforcement learning, addresses this issue by having the agent compute an optimal value function, from which a policy can be obtained, while interacting with the environment. Unfortunately, the convergence speed of such methods is not always fast enough to grant good results in a reasonable time. Furthermore, value functions are not the most intuitive way for a programmer to represent *a priori* knowledge or to include safety restrictions to the agent's decision making process.

Ideally, an hybrid controller that provides the agent with pre-programmed planning alternatives while letting it optimize the plans by interacting with the environment can achieve a middle ground between both ends of the spectrum and obtain better results. In fact, supervising the action choices at each decision moment effectively reduces the space in which an optimal policy needs to be searched. Furthermore, such approach provides a close link between the agent and the programmer through the supervisor – there is a high practical value in integrating the knowledge about unsafe or undesired paths into the system from the start, and not having to wait for the agent to find out about their undesirability.

Our goal is to provide a systematic way to combine the supervisory control approach, which we can use to provide weakly specified pre-programmed plans, with Q-learning, which implements the adaptive nature of the framework, for a continuous-time system under partial observability. For that we integrate several components as represented in Figure 4.1.

Block **G** represents the uncontrolled system, including the model for the effects of the robot's actions, **Obs** provides an observer which transforms the strings of events generated by the system in a state based representation to be used by the learner, **Sup** supervises the control process, providing the action restrictions to the controller according to specifications, and **RL** is a Reinforcement Learning based controller, which optimizes the behavior of the system using policy π. **P** simply represents the projection which filters the events that are observable by the robot.

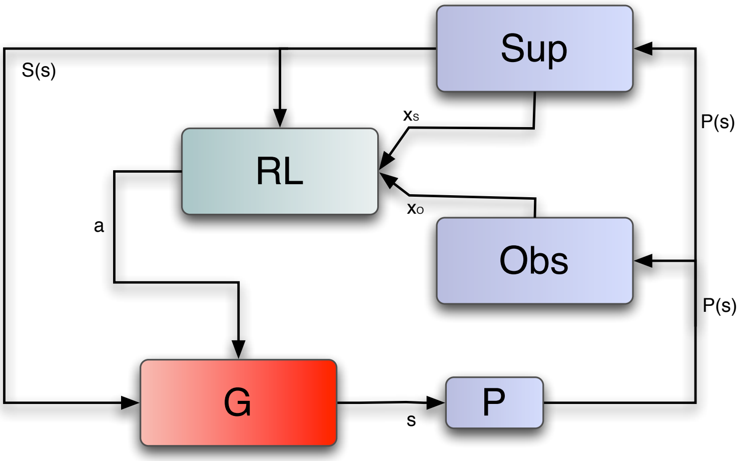


Figure 4.1: Supervised event-based Q-learning block diagram.

For analysis or simulation purposes, it is important to know the model of the underlying system and the type of stochastic process that it describes. On the other hand, for action execution purposes, the actual complete model of **G** does not need to be known. The controller and supervisor act based on the events that the environment generates, without a direct access to its state.

4.1 Environment Model

We assume the discrete event process we are trying to control is a Stochastic Process known as Generalized Semi-Markov Process (GSMP) (Glynn, 1989) and, so, can be identified by a STA, as defined in Section 2.1.4. We consider some extensions to the STA formalism:

- $E = E_U \cup E_T$, with $E_U \cap E_T = \emptyset$ and $E_T \neq \emptyset$. The names E_T and E_U come from *timed* and *untimed*, respectively.
- $X_{rs} \subset X$, with $X_{rs} \neq X$, is the set of states x for which $\Gamma(x) \cap E_U \neq \emptyset$.
- The events in E_T have an associated stochastic clock structure, $\mathcal{T} = \{\mathcal{T}_i : i \in E_T\}$.
- The events in E_U have no time associated with them and are considered to fire immediately.
- $p_{rs} : X_{rs} \times E_U \to [0, 1]$ is generally a partial function defined only for the events active in state x. When all events are in E_U, $\sum_{i \in \Gamma(x) \cap E_U} p_{rs}(x, i) = 1$ but if there are active timed events in $\Gamma(x)$ and $\sum_{i \in \Gamma(x) \cap E_U} p_{rs}(x, i) < 1$, the remaining probability mass corresponds to waiting for one of the timed events to happen[1].

We call this kind of automaton a *Generalized Stochastic Timed Automaton* (GSTA) to emphasize the fact that it has both timed and immediate (untimed) transitions.

[1] The index rs derives from the designation *random switch*, commonly used in stochastic petri nets and which applies to markings which enable immediate transitions.

In general, the automaton that identifies the GSMP of the system is the result of a parallel composition (Cassandras and Lafortune, 2007) of several smaller automata. These represent different aspects of the system and, as defined by the parallel composition operation (see Section 2.1.2), range from being fully synchronized with every event to running in parallel without any relation with each other, all depending on the E sets of each STA.

4.1.1 Inputs

We introduce controllable events E_C, with $E_C \in E$ and denote the GSTA with controllable events as a *Controllable GSTA* (C-GSTA). As in Section 2.1.2, we assume there is a supervisor function defined over the language generated by the system $\mathbf{G}$:

$$S : \mathcal{L}(G) \rightarrow 2^E$$

We assume the supervisor to be admissible and will refer to S as a *supervisor policy* and $S(s)$ as a *supervisor action.* The set of active events under supervision is $\Gamma_S(x, s) = \Gamma(x) \cap S(s)$, when the system is in state x and has, so far, output string s. We will further elaborate on properties of the supervisor in Section 4.3 but for now it is sufficient to assume it exists and is admissible.

Additionally, we consider that, for events that are both in E_C and E_U, and for that reason occur immediately after a state change, it is possible to control directly the function p_{rs}. We define:

$$\pi : \mathcal{L}(G) \times E_A \rightarrow [0, 1]$$

where $E_A = E_C \cap E_U$.

Note that, just as the supervisor function, this one depends on the string generated by the system, meaning that it is a dynamic map which can provide different values for the same state of the system, depending on the history represented by s. In any case, π is generally a partial function defined only for the immediate controllable events which are enabled by the supervisor for a given string, i.e. $\Gamma_S(x, s) \cap E_A$, noting that the other controllable and immediate events are not enabled due to supervisory restrictions. We refer to π as the *controller policy.*

Equivalently, we can assume the controller itself chooses an event from $\Gamma_S(x, s) \cap E_A$, using the policy π, and passes that event to the system. This one event, which we will denote as a and refer to as the *controller action*, despite being immediate might not be the one to fire due to the possible existence of other immediate events that are not controllable, i.e. events in $\Gamma_S(x, s) \cap (E_U \setminus E_C)$. The function p_{rs} for a, will be given by:

$$p_{rs}(x, a) = 1 - \sum_{e \in (\Gamma_S(x,s) \cap (E_U \setminus E_C))} p_{rs}(x, e) \tag{4.1}$$

Events in E_A can be interpreted, in a robotics setting, as commands that start or stop a given behavior of the robot. In fact, even though such commands do not occur immediately in reality, their time scale is usually several degrees smaller than that of the actual behaviors. It is usually these kind of events that we are interested in controlling or, put another way, the objective of an agent modeled using this approach is to determine when to start and when to stop behaviors internally. For this reason, unless stated otherwise, we make a small simplification to the model and assume that all controllable events are immediate, $E_C \subseteq E_U$ and, consequently, $E_A = E_C$.

Nevertheless, this simplification is mostly a modeling choice and the majority of the results we present do not depend on it and could easily be extended or modified to address the most general case. We will explicitly state so if a result depends on the fact that all controllable events are immediate.

Additionally, the most common case where some events are immediate but uncontrollable is when they correspond to start/stop commands of other robots, and although it is not a rule, for a single robot case it is common that $E_C = E_A = E_U$.

Finally, it is not mandatory for the controller to actively pick an event in E_A at each decision point. In fact, a possible decision might be to wait for the occurrence of an uncontrollable event, i.e, choosing an *idling action* or ε_a. Nevertheless, it is important that an idling action does not lead to a situation where the system has no other option but to stay indefinitely in the same state. For a system in state x and after having generated string s we must have:

$$\Gamma_S(x,s) \setminus E_A \neq \emptyset \tag{4.2}$$

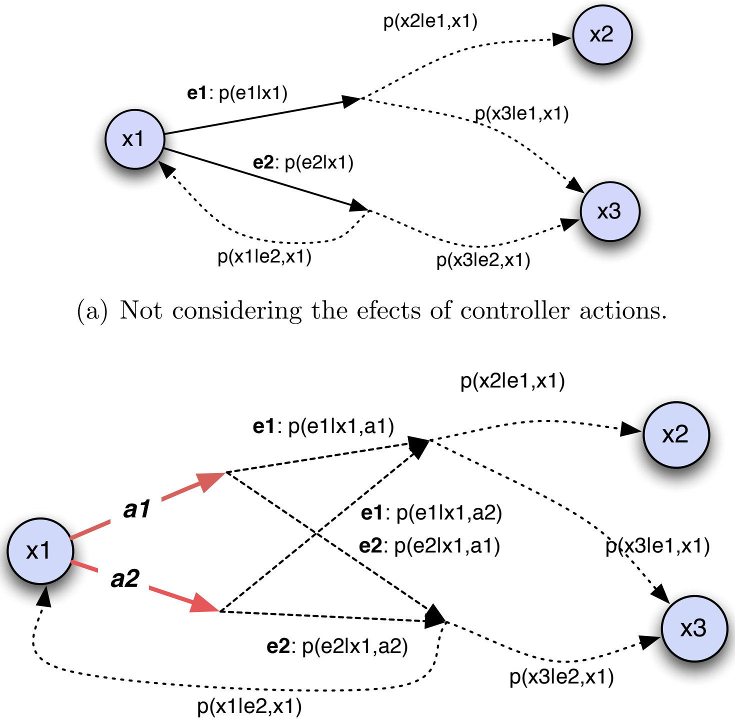
4.1.2 Transitions

The transitions in this model are driven by two different aspects: which event will fire in a given state and, knowing that, which state will the system jump to. Figure 4.2 schematizes this two aspects. The first aspect is associated with the probability that a given event will fire in state x, that is $p(e|x)$, or, considering the effects of the controller and supervisor actions, $p(e|x,S,a)$. For states in X_{rs} this quantity is easily derived. We have:

$$p(e|x,S,a) = \begin{cases} p_{rs}(x,e) & e \in \Gamma_S(x,s) \cap (E_U \setminus E_C) \\ 1 - \sum_{i \in (\Gamma_S(x,s) \cap (E_U \setminus E_C))} p_{rs}(x,i) & e = a \wedge e \in \Gamma_S(x,s) \\ 0 & \text{otherwise} \end{cases} \tag{4.3}$$

In Equation (4.3) we are assuming that a is not the idling action. If we do have $a = \varepsilon_a$, the system will behave as if $x \notin X_{rs}$ and will look at the time structure associated with the events in E_T to determine the next event to fire. Essentially, we need to determine $p(e|x,S,\varepsilon_a) = P[E = e|X = x, S]$, for all events in E_T that are active in state x and after occurrence of string s, and we start by pointing out that writing $E = e$ is equivalent to:

$$Y_e \leq \min_{\substack{j \in (\Gamma_S(x,s) \cap E_T) \\ j \neq e}} \{Y_j\}$$



(a) Not considering the efects of controller actions.

(b) Considering the effects of controller actions.

Figure 4.2: Diagram of the STA model parameters.

where Y_j is a random variable corresponding to the next potential firing time of event j[2]. Essentially we are saying that the firing time of event e must be smaller than that of other events in E_T. We now define a random variable W_e representing the minimum firing time of any event other than e.

$$W_e = \min_{\substack{j \in (\Gamma_S(x,s) \cap E_T) \\ j \neq e}} \{Y_j\}$$

and we have that $P[E = e|X = x] = P[Y_e \leq W]$ and applying the rule of total probability:

$$P[Y_e \leq W] = \int_0^\infty P[Y_e \leq W | W = t] \cdot dF_W(t)$$

Assuming the firing times between different events are independent, it can be proven

[2] Strictly speaking, Y_e is a different random variable every time the system reaches a state x which is dependent on the time event e has been active before the system reached that state, called the *age* of event e. For simplicity of notation, we do not explicitly write the dependence of the random variables Y_j on the age of the events when the system reaches x since we will introduce restrictions to the model that allow us to abstract from that additional memory variable, but it is important to note that, in general and unless stated otherwise, every time we write Y_e there is an implicit dependency on the age of e.

that:

$$F_{W_e}(t) = 1 - \prod_{\substack{j \in (\Gamma_S(x,s) \cap E_T) \\ j \neq e}} (1 - P[Y_j \leq t])$$

and so, we obtain:

$$p(e|x, S, \varepsilon_a) = \int_0^\infty P[Y_e \leq W_e | W_e = t] \cdot dF_{W_e}(t) \tag{4.4}$$

Also of importance is determining the joint probability of moving to state x within a time frame of t. First, we define a random variable Y^* describing the next interevent time.

$$Y^* = \min_{j \in (\Gamma_S(x,s) \cap E_T)} \{Y_j\}$$

Note that it is similar to W_e except that it takes into consideration all events. Now we have[3]:

$$\begin{aligned}
H(t, x'|x, S, \varepsilon_a) &= P\left[Y^* \leq t, X_{k+1} = x' | X_k = x, S, \varepsilon_a\right] \\
&= \sum_{e \in \Gamma_S(x,s) \cap E_T} P\left[Y^* \leq t, X_{k+1} = x' | E = e, X_k = x, S, \varepsilon_a\right] \; P\left[E = e | X_k = x, S, \varepsilon_a\right] \\
&= \sum_{e \in \Gamma_S(x,s) \cap E_T} P\left[Y^* \leq t | E = e\right] \; P\left[X_{k+1} = x' | X_k = x, E = e\right] \; P\left[E = e | X_k = x, S, \varepsilon_a\right] \\
&= \sum_{e \in \Gamma_S(x,s) \cap E_T} P\left[Y^* \leq t | E = e\right] \; p(x'|x, e) \, p(e|x, S, \varepsilon_a)
\end{aligned}$$

Since the event is conditioned to be e, the firing time will be only dependent on e and $P\left[Y^* \leq t | E = e\right] = P\left[Y_e \leq t\right]$. We have:

$$H(t, x'|x, S, \varepsilon_a) = \sum_{e \in \Gamma_S(x,s) \cap E_T} P\left[Y_e \leq t\right] \; p(x'|x, e) \, p(e|x, S, \varepsilon_a) \tag{4.5}$$

If we condition the probability to an event e, we have:

$$\begin{aligned}
H(t, x'|x, e) &= P\left[Y^* \leq t, X_{k+1} = x' | X_k = x, E_k = e\right] \\
&= P\left[Y^* \leq t | E_k = e\right] P\left[X_{k+1} = x' | X_k = x, E_k = e\right] \\
&= P\left[Y_e \leq t\right] \; p(x'|x, e)
\end{aligned} \tag{4.6}$$

and consequently:

$$H(t, x'|x, S, \varepsilon_a) = \sum_{e \in \Gamma_S(x,s) \cap E_T} H(t, x'|x, e) \, p(e|x, S, \varepsilon_a) \tag{4.7}$$

[3] Note that although Y^* is not dependent on the state itself, it does depend on the ages of all the events in $\Gamma_S(x, s)$ but, again, we will not explicitly write this dependancy.

respecting the law of total probability.

The problem with the previous expressions is that, at each state, the residual firing times Y_e may be different because the corresponding events might have been enabled for a long time already before the system reached that state, i.e. the age of the event is not necessarily known. Considering we intend to use a reinforcement learning based controller, it is crucial that, at least when an event fires and the state changes, the system does not need additional memory other than the state to which it changed. That way, even though time is continuous, it might be possible to end up with a process similar to SMDPs, described in Section 2.2.4. To accomplish that we need to make an assumption about the system:

Assumption 4.1.1. *We will assume one of the two simplifications:*

i. At each state, a reset is made to the firing times of every event, i.e., the age of all events is assumed to be reset to zero at every state change.

OR

ii. The occurrence of a number of events in a given interval can be modeled as a Poisson process (we can say we have a Poisson clock structure).

Note that the second assumption is a special case of the first but one that, due to the nature of Poisson processes, transforms the *Generalized Semi Markov Decision Process* induced by the C-GSTA in a *Continuous Time Markov Decision Process* (CTMDP).

Clock resets

In this case the residual times at some state always follow the same distribution, since there is no bias due to the time an event has already been active. So, Equation (4.4) can be rewritten as :

$$p(e|x, S, \varepsilon_a) = \int_0^\infty F_e(y) \cdot dF_{W_e}(y) \tag{4.8}$$

where F_e represents the firing time distribution of event e, assumed to be the same for every state.

We also have:

$$F_{W_e}(t) = 1 - \prod_{\substack{j \in \Gamma_S(x,s) \\ j \neq e}} (1 - F_j(t)) \tag{4.9}$$

On the joint probability of a state and a time interval, from Equations (4.5) and (4.8) we have:

$$H(t, x'|x, S, \varepsilon_a) = \sum_{e \in \Gamma_S(x,s)} F_e(t)\, p(x'|x, e) \int_0^\infty F_e(y) dF_{W_e}(y) \tag{4.10}$$

and

$$H(t, x'|x, e) = F_e(t)\, p(x'|x, e) \tag{4.11}$$

Poisson clock structure

With a Poisson clock structure, all the firing times will be distributed according to an exponential distribution, i.e., the distributions $F_e(t)$ will be given by:

$$F_e(t) = 1 - e^{-\lambda_e\, t}$$

In this situation we have:

$$F_{W_e}(t) = 1 - e^{-\Lambda_e(x,s)\, t}$$

with $\Lambda(x,s) = \sum_{j \in \Gamma_S(x,s) \cap E_T} \lambda_j$ and $\Lambda_e(x,s) = \Lambda(x,s) - \lambda_e$ (only for $e \in \Gamma_S(x,s) \cap E_T$).

On the other hand the memoryless property allows us to simplify $P[Y_e \leq W_e | W_e = y]$ simply to $F_e(y)$ and we have:

$$\begin{aligned} p(e|x, S, \varepsilon_a) &= \int_0^\infty (1 - e^{-\lambda_e y}) \cdot \Lambda_e(x,s) \cdot e^{-\Lambda_e(x,s)\, y} dy \\ &= \frac{\lambda_e}{\Lambda(x,s)} \end{aligned}$$

As for the joint probability, we have:

$$H(t, x'|x, S, \varepsilon_a) = \sum_{e \in \Gamma_S(x,s) \cap E_T} (1 - e^{-\lambda_e t}) p(x'|x, e) \frac{\lambda_e}{\Lambda(x,s)} \tag{4.12}$$

and

$$H(t, x'|x, e) = (1 - e^{-\lambda_e t}) p(x'|x, e) \tag{4.13}$$

The Poisson clock structure effectively removes the need to have a memory of the time elapsed, with the event firing distributions depending solely on the state of the process. As mentioned previously, in this case we end up with a Continuous-Time Markov Decision Process (CTMDP) (Howard, 1960; Bertsekas, 1987), also known as a controlled Continuous-Time Markov Chain.

4.2 Observer Model

So far, since we only described the environment model, no assumptions were made about the observability of the system. In this chapter, we will assume it is fully observable w.r.t. the system state and what remains to be determined are the implications on the model parameters of such an assumption. First, it is important to note that the output of the system is a string of events and the other blocks described in Figure 4.1 do not have a direct access to the state of the system but, instead, must be able to either directly work on the string of events or use it to obtain an estimate of the state.

Particularly, what we want from an observer of the system is to be able to obtain an exact state estimation from the event string output.

Let us represent the observer by a function

$$O : P(\mathcal{L}(G)) \rightarrow \hat{X}$$

where $\hat{X}$ denotes the space of state estimations. We define full observability w.r.t. the state as:

Definition 4.2.1. *An STA,* $\boldsymbol{G}$*, is said to be* fully observable w.r.t. to the state *if it is possible to construct a state estimation set* $\hat{X}$*, a function* $O : P(\mathcal{L}(G)) \rightarrow \hat{X}$ *and a bijection* $b : X \rightarrow \hat{X}$ *such that for every* $s \in \mathcal{L}(G)$ *and* $x \in X$*, if* s *is a possible trace from* x_0 *to* x *then* $(b^{-1} \circ O)(P(s)) = x$*. Particularly* $(b^{-1} \circ O)(\varepsilon) = x_0$

Essentially, there are two necessary (and sufficient) conditions that will guarantee full observability:

- An observer must be able to distinguish between different strings.
- The same string of events must not potentially lead the system to different states.

The first condition is met if there are no unobservable events and the second one is met if the transition function p is deterministic. Figures 4.3 and 4.4 show how if these conditions are not met the observer has not way of distinguishing between two different states.

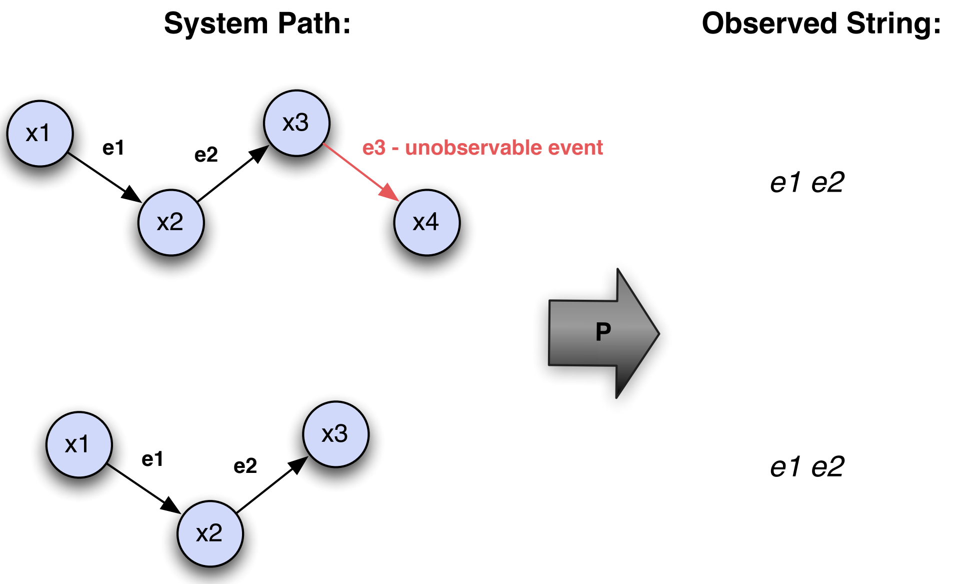


Figure 4.3: Breaking full observability from the existence of unobservable events.

This is expressed in the following theorem:

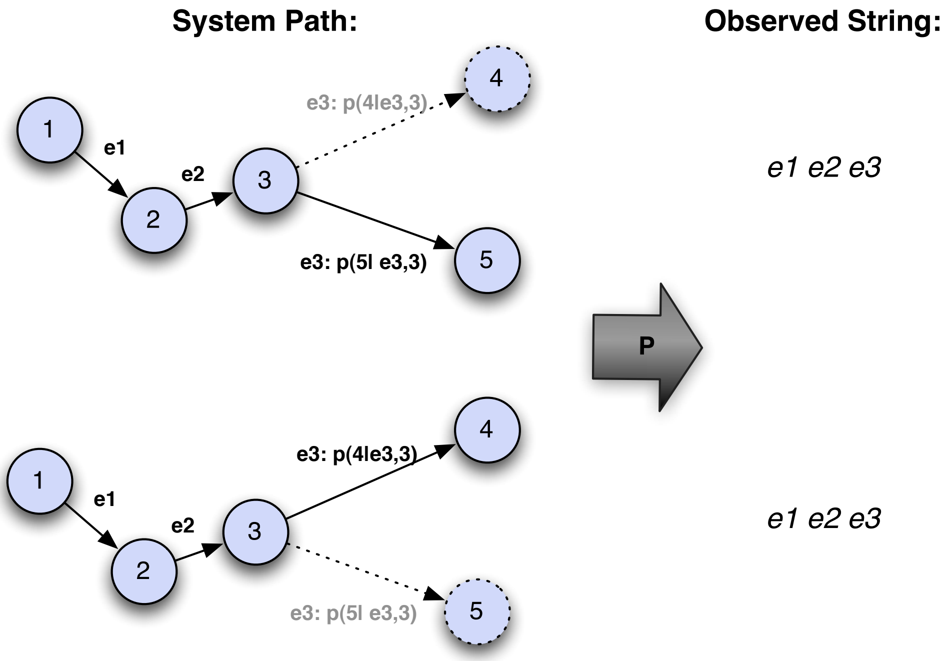


Figure 4.4: Breaking full observability from ambiguity in the events effects.

Theorem 4.2.1. *An STA, $\boldsymbol{G}$, is fully observable w.r.t. the state iff:*

1. $E_{uo} = \emptyset$
2. $\exists_{x_0 \in X}\, p_0(x_0) = 1$
3. $\forall_{x \in X} \forall_{e \in \Gamma(x)} \exists_{x' \in X}\, p(x'|x, e) = 1$

Proof. We start by noting that conditions 2 and 3 are equivalent to having a transition function $f : X \times E \to X$ as for FSA defined in Section 2.1.1. In fact, another way of stating conditions 2 and 3 is to say that the underlying transitions of the STA are regulated by a deterministic FSA. Additionally, we note that since all the events are assumed to be observable the projection of the language generated by the system is the language itself, i.e., $P(\mathcal{L}(G)) = \mathcal{L}(G)$. We will now construct $\hat{X}$ such that it is isomorphic with X by some bijection b. Since b has an inverse by construction, if we set:

$$O(P(s)) = (b \circ f)(x_0, P(s))$$

and considering $x = f(x_0, s)$, we have:

$$(b^{-1} \circ O)(P(s)) = f(x_0, s) = x$$

showing that the presented conditions are sufficient to ensure full observability in the terms of Definition 4.2.1.

To show they are also necessary conditions, we will see what happens if any of them is broken. We assume the system is indeed fully observable w.r.t. the state:

- $E_{uo} \neq \emptyset$. So let us assume that, after outputing string s, the system has reached a state x such that $\Gamma(x) \cap E_{uo} \neq \emptyset$, i.e. a state that has active unobservable events. If the next event to fire is e such that $e \in E_{uo}$ and considering there is a $x' \neq x$ such that $x' = f(x, e)$, we have that it is possible that after $s' = se$ the system is in state x'.
  Since the system is fully observable, $(b^{-1} \circ O)(P(s)) = x$ and $(b^{-1} \circ O)(P(s')) = x'$ for some observer function O and some bijection b. But considering $P(s) = P(s')$ because e is unobservable, we have that

  $$x = (b^{-1} \circ O)(P(s)) = (b^{-1} \circ O)(P(s')) = x'$$

  which is impossible since we already know $x \neq x'$.

- $\neg(\exists_{x_0 \in X}\, p_0(x_0) = 1)$. In that case, there are at least two states x_{01} and x_{02} that have initial non-null probability. We will have that $O(\varepsilon) = b(x_{01})$ since $P(\varepsilon) = \varepsilon$. But we could as easily write $O(\varepsilon) = b(x_{02})$, which means that $b(x_{01}) = b(x_{02})$ and since $x_{01} \neq x_{02}$ the function b cannot be a bijection.

- $\neg(\forall_{x \in X} \forall_{e \in \Gamma(x)} \exists_{x' \in X}\, p(x'|x, e) = 1)$. Similarly to the initial state condition, let us take a state x and an event e such that there are x'_1 and x'_2 with $x'_1 \neq x'_2$ and for which $0 < p(x'_1|x, e) < 1$ and $0 < p(x'_2|x, e) < 1$, which are known to exist by assumption. We will have that $O(P(se)) = b(x'_1)$ but we could as easily write $O(P(se)) = b(x'_2)$, which means that $b(x'_1) = b(x'_2)$ and since $x'_1 \neq x'_2$ the function b cannot be a bijection.

It is necessary that all conditions are satisfied for the system to be fully observable, which concludes the proof.

□

It is important to note that the firing of events is still regulated by a stochastic process and, for this reason, the system is not deterministic, which is reflected on the quantities $p(e|x, S, a)$.

We saw that the easiest way to construct an observer for a fully observable STA is to make use of the transition function f and some bijection. In fact, on fully observable systems $g(s) = f(x_0, s)$ is isomorphic of $O(s)$ and a good way to construct an observer is simply to use a deterministic FSA:

$$\mathbf{Obs} = (\hat{X}, E, f_O, \Gamma_O, \hat{x}_0)$$

where $\hat{X} = b(X)$ and particularly $\hat{x_0} = b(x_0)$, for some bijection, and $f_O(\hat{x}, e) = (b \circ f)(b^{-1}(\hat{x}), e)$ which is extended to strings in the usual way.

In fact, this observer automaton recognizes the same language that is generated by the system, and takes the events generated by $\mathbf{G}$ as inputs and sends the state estimate as an output. The state estimate can then be used by a controller to aid the decision process, as shown in Figure 4.1, which is particularly relevant if the controller relies on a state-based representation like the reinforcement learning algorithms we use. It is also important to note that, since we are assuming **G** to have a finite state space, the observer automaton **Obs** will also have a finite state space – the language generated by **G** and recognized by **Obs** is regular.

4.3 Supervisor Model

In general, considering the presence of unobservable events, a supervisor will be defined by a function:

$$S : P(\mathcal{L}(G)) \rightarrow 2^E$$

but since we are assuming full observability, the projection P will simply be an identity function and we have:

$$S : \mathcal{L}(G) \rightarrow 2^E$$

We assume this supervisor to be admissible, as defined in Section 2.1.2. Ideally, it will also be constructed according to the theorems defined in (Cassandras and Lafortune, 2007) to avoid leading the system to unintended locks. Nevertheless, in this work we are concerned particularly with integrating the supervisor and adaptive controller together and will, for the most part, assume the supervisors are also well specified.

We know by assumption that the language generated by **G** is regular and the system has a finite state space. However, this does not guarantee that the language generated by the supervised system, $\mathcal{L}(S/G)$, is also regular. However, since this will be the case if the supervisor is itself realized by a FSA, we will make the following assumption:

Assumption 4.3.1. $\mathcal{L}(S/G)$ *is regular, and* S *can be realized by a FSA.*

Still, the realization of the supervisor could be done using other techniques as long as they respect the condition of keeping the language regular. However, the overall state of the supervised now depends not only on the state of **G** but also of **Sup** and, for this reason, having a state based representation for the supervisor is also important for the controller to use. We use FSAs as the realization for our supervisors:

$$\mathbf{Sup} = (X^S, E^S, f_{\mathbf{Sup}}, \Gamma_{\mathbf{Sup}}, x_0^S)$$

Some of the functions previously defined in terms of the output string can, then, be redefined in terms of the supervisor state. Particularly, we have that:

$$\Gamma_S(x, s) = \Gamma_S(x, x^S) \text{ and } S(s) = S(x^S)$$

with $x^S = f_{\mathbf{Sup}}(x_0^S, s)$.

We also have that $\Gamma_S(x, x^S) = \Gamma_{\mathbf{G}\|\mathbf{Sup}}(x, x^S)$ or $\Gamma_S(x, x^S) = \Gamma_{\mathbf{G}\times\mathbf{Sup}}(x, x^S)$, depending on the composition operation we are using. Typically, even if we use parallel composition, the event set over which the supervisor works will still be a subset of the event set of the system, $E^S \subset E$, and we will have:

$$\begin{aligned}\Gamma_S(x, x^S) &= (\Gamma_{\mathbf{G}}(x) \cap \Gamma_{\mathbf{Sup}}(x^S)) \cup (\Gamma_{\mathbf{G}}(x) \setminus E^S) \\ &= \Gamma_{\mathbf{G}}(x) \cap (\Gamma_{\mathbf{Sup}}(x^S) \cup E \setminus E^S)\end{aligned}$$

Since the controller does not have a direct access to the state of the system and it relies on the observer for a state estimate, the state space on which a reinforcement learning algorithm works will be a subset of the cartesian product of $\hat{X}$ and X^S.

We extend the notation referring to the system state to include the state of the supervisor and, unless it is unambiguous whether we are talking about the supervised or unsupervised system, will begin to denote the state of the system as x^G and we will have that $x = (x^G, x^S)$. Similarly, if we denote the state of the observer as $\hat{x}^G$ we will have that the state estimate of the supervised system will be denoted as $\hat{x} = (\hat{x}^G, x^S)$ (note that the controller has a direct access to the state of the supervisor so its estimate is the exact state).

We can now redefine a lot of the quantities defined in Section 4.1 in terms of the extended state, which is particularly useful to avoid cluttered notation. The probability of an event firing given a control action and supervisor action still only depends on the state of **G**.

$$p(e|x, S, a) = p_{\mathbf{G}}(e|x^G, S, a)$$

The transition probability depends on both the system state and the supervisor state:

$$p(x'|x, e) = p_{\mathbf{G}}(x^{G'}|x^G, e)p_{\mathbf{Sup}}(x^{S'}|x^S, e)$$

and under the assumptions of full observability and representation of the supervisor by a deterministic FSA [4] :

$$f(x, e) = (f_{\mathbf{G}}(x^G, e), f_{\mathbf{Sup}}(x^S, e))$$

Unless stated otherwise, we will assume that generally we are talking about the augmented state representation of the supervised system, or its observer estimate. We will also drop the explicit cartesian product notation from the argument of Γ_S to write simply $\Gamma_S(x) = \Gamma_S(x^G, x^S)$, but we will maintain the subscript to emphasize we are dealing with the supervised system.

4.4 Controller Model

Rewards

The objective of the controller is to choose an action at each decision moment so that the performance of the robot is optimized, according to some reward function. So, it

[4] In reality, $f_{\mathbf{G}}$ and $f_{\mathbf{Sup}}$ might not be defined for some of the events, which also depends on the type of composition we are considering. The expression for the transition function can then easily be adapted accordingly as described in Section 2.1.1

is only natural that the reward function needs to be defined previously. We define the rewards in a similar way to Section 2.2.4, except that they depend on events directly and not on actions:

- $\kappa : X \times E \times X \longrightarrow \mathcal{R}$ is defined as the expected immediate[5] reward the agent gets after moving from state x to state x' due to the firing of event e.
- $\mu : X \longrightarrow R$ is defined as the expected reward rate the agent gets for as long as he stays in state x.

Note how this formulation differs from the one of Section 2.2.4 essentially due to the fact that the rewards depend on the events and not the actions, and events occur at the end of a transition from x to x' and not at the beginning. For this reason, the agent receives the immediate reward at the end of the transition and, on the other hand, it would break causality to have rate c depend on an event that has not occurred so it only depends on the state the system is in during the entire transition from x to x'.

The actual state space over which the rewards are defined depends strongly on the way those rewards are interpreted. If we consider them to be supplied directly by the environment, than there is a dependence on the actual state of $\mathbf{G}$, possibly augmented by the state of the supervisor, x^S. If, on the other hand, we consider they are obtained by some process posterior to the observation, and are thus subject to the same observational constraints of the controller, it is more natural to define them in terms of the observer state, $\hat{x}$; we will use this last interpretation. Note that under full observability there is a bijection between X and $\hat{X}$ induced by the observer and so it is equivalent using either of the interpretations.

Although κ depends on the state for which the system moves to, we can redefine it to only depend on the state from which it starts and the event that fired, by doing:

$$\kappa(x, e) = \sum_{y \in X} p(y|x, e)\kappa(x, e, y)$$

In any case, in our formulation of the reward scheme this immediate reward is only received at the end of transition.

4.4.1 Optimality Equations

Having defined the rewards, we can now obtain equations for the expected discounted reward, much like what was done for SMDPs in Section 2.2.4. The objective is, as before, to obtain a value for each state that reflects how much we can expected to obtain if we start in such a state.

$$V^{S,\pi}(x) = E\left\{ \sum_{k=0}^{\infty} e^{-\beta\sigma_k} \left[e^{-\beta\tau_k}\kappa(x_k, e_k) + \int_0^{\tau_k} e^{-\beta t}\mu(x_k, e_k)dt \right] \middle| x_0 = x, S, \pi \right\} \quad (4.14)$$

[5] Also known as lump reward/cost, and we will use the two terms interchangeably.

where S is the supervisor policy and π is the controller policy, τ_k represents the sojourn time in for state x_k and β is the discount factor. Additionally, we have $\sigma_0 = 0$ and $\sigma_k = \sum_{i=0}^{k-1} \tau_k$ for $k > 0$[6].

Similarly to Section 2.2.4, we can write:

$$r(x,e) = \int_0^\infty e^{-\beta\tau} k(x,e) + \frac{1-e^{-\beta\tau}}{\beta} c(x,e)\, F(d\tau|x,e) \tag{4.15}$$

The Bellman equation for controller policy π and supervisor policy S can be written as:

$$\begin{aligned} V^{S,\pi}(x) &= E\left\{ \sum_{k=0}^{\infty} e^{-\beta\sigma_k} r(x_k,e_k) \middle| x_0 = x, S, \pi \right\} \\ &= \sum_{e\in\Gamma_S(x)} p(e|x,S,\pi) E\left\{ \left[r(x,e) + e^{-\beta\tau} V^{S,\pi}(x_{k+1}) \middle| \pi \right\} \right] \\ &= \sum_{e\in\Gamma_S(x)} p(e|x,S,\pi) \left[r(x,e) + \sum_{y\in X} \left(\int_0^\infty e^{-\beta\tau} H(d\tau,y|x,e) \right) V^{S,\pi}(y) \right] \end{aligned} \tag{4.16}$$

where $H(t,x'|x,e) = F_e(t)p(x'|x,e)$ represents the probability of moving from state x to state x' in less than t time, knowing that the change state change happened due to event e, as defined in Equation (4.11). We can define $\gamma_e = \int_0^\infty e^{-\beta\tau} F(d\tau|x,e)$ and end up with the following expression for the Bellman equation:

$$V^{S,\pi}(x) = \sum_{e\in\Gamma_S(x)} p(e|x,S,\pi) \left[r(x,e) + \gamma_e \sum_{y\in X} p(y|e,x) V^{S,\pi}(y) \right] \tag{4.17}$$

which is quite similar to Equation (2.3), with the difference that we are adding over events instead of actions and it is as if each of the events has an independent discount factor given by γ_e.

During these derivations no restriction was made on the nature of the transition probabilities, although the observability conditions of Theorem 4.2.1 require that:

$$\forall_{x\in X} \forall_{e\in\Gamma(x)} \exists_{x'\in X}\; p(x'|x,e) = 1$$

However, if the controller did have a direct access to the state instead of through the strings of events generated by the system, then there would be no need to impose such conditions – the observability question is a problem of the *observer* and not the *controller* who only requires an correct state estimate but is not concerned with the way this estimate was obtained, and for that reason we always left the term $p(y|e,x)$ in the equations instead of restricting the sum right away to the only state $y \in X$ that, under observability and a for given $x \in X$ and $e \in \Gamma_S(x)$, verifies $p(y|x,e) = 1$.

[6] And consequently $\tau_k = \sigma_{k+1} - \sigma_k$.

We will continue to leave the transition arbitrarily distributed in the optimality expressions, particularly because it will be of use in other chapters, but if we combined Equation 4.17 with the condition of Theorem 4.2.1 on the transition probabilities, we would get:

$$\begin{aligned} V^{S,\pi}(x) &= \sum_{e\in\Gamma_S(x)} p(e|x,S,\pi)\left[r(x,e)+\gamma_e V^{S,\pi}(f(x,e))\right] \\ &= r^{\pi,S}(x,e) + \sum_{e\in\Gamma_S(x)} \gamma_e\, p(e|x,S,\pi)V^{S,\pi}(f(x,e)) \end{aligned}$$

with $r^{\pi,S}(x,e) = \sum_{e\in\Gamma_S(x)} p(e|x,S,\pi)r(x,e)$. We can see from this equation that the successor state is not determined by a policy because the event that will fire is unknown from the start, even though the transitions w.r.t. the events are deterministic under full observability.

It is important to underline where the supervisor and controller influence the state value function, on the previous equation. In fact, their effect is patent in the events over which we are summing and on the quantities $p(e|x,S,\pi)$, which can be defined based on Equation (4.4):

$$p(e|x,S,\pi) = \sum_{a\in A_x} \pi(a,x)p(e|x,S,a) \tag{4.18}$$

with

$$A_x = \begin{cases} \Gamma_S(x)\cap E_A & \Gamma_S(x)\setminus E_A = \emptyset \\ (\Gamma_S(x)\cap E_A)\cup\{\varepsilon_a\} & otherwise \end{cases}$$

The optimality equation can be written as:

$$V^{S,*}(x) = \max_{a\in A_x}\left[\sum_{e\in\Gamma_S(x)} p(e|x,S,a)\left[r(x,e)+\gamma_e\sum_{y\in X} p(y|e,x)V^{S,*}(y)\right]\right] \tag{4.19}$$

and in terms of Q-values we can write:

$$Q^{S,*}(x,a) = \sum_{e\in\Gamma_S(x)} p(e|x,S,a)\left[r(x,e)+\gamma_e\sum_{y\in X} p(y|e,x)\max_{b\in A_y} Q^{S,*}(y,b)\right] \tag{4.20}$$

which takes the same form as Equation (2.18) if we make:

$$r(x,a) = \sum_{e\in\Gamma_S(x)} p(e|x,S,a)r(x,e)$$

and

$$\begin{aligned} M(y|x,a) &= \sum_{e\in\Gamma_S(x)} p(e|x,S,a)\,\gamma_e\, p(y|e,x) \\ &= \int_0^\infty e^{-\beta t} H(dt,y|x,S,a) \end{aligned}$$

In fact, as mentioned previously the clock reset assumptions at each state change transform the GSMDP induced by the C-GSTA into a SMDP, since no further memory of the system is needed at each decision point, particularly there is no need to store temporal values prior to the state change.

In Section 4.1 we defined actions as corresponding to controllable events that fire immediately, with the exception of the idling action that may or may not correspond to the immediate firing of events depending on whether there are uncontrollable and immediate events active in the state or not – with the assumption that $E_U = E_C$ then an idling action will, in fact, correspond to waiting for an uncontrollable timed event to fire. Particularly, states that do not correspond to random switches, i.e. states not in X_{rs} can be thought of as only having the idling action ε_a available. For the states, the optimality equation acquires a particular form with:

$$Q^{S,*}(x,a) = \sum_{e\in\Gamma_S(x)} p(e|x,S,a)\left[k(x,e) + \sum_{y\in X} p(y|e,x)\max_{b\in A_y} Q^{S,*}(y,b)\right] \tag{4.21}$$

which clearly shows how there is no discount since the state change occurs infinitesimally right after the decision.

It is handy to separate states with random switches from the ones that take some time, particularly because the firing of events is directly determined by a probability mass function and not by an event race. Nevertheless, we could unify the firing times of events in E_U and events in E_T by saying that all events in E_U will have the Heaviside function as their distribution:

$$F_e(t) = \begin{cases} 1 & t \geq 0 \\ 0 & t < 0 \end{cases}, \quad \forall e \in E_U$$

Intuitively, it is clear this causes problems for convergence since, as show previously, there will be no discount factor to bound the rewards. On the other hand, it is easy to construct a system where infinite immediate events occur in succession in zero time. This means that some additional assumptions need to be made in order to keep the system well behaved, which will be discussed further ahead.

Poisson Clock Structure

Under a Poisson clock structure, the expressions for the optimality equations take a particular form. In fact, if we consider that for the idling action

$$p(e|x,S,\varepsilon_a) = \frac{\lambda_e}{\Lambda(x)} \tag{4.22}$$

with $\Lambda(x) = \sum_{j\in\Gamma_S(x)\cap E_T} \lambda_j$ then Equation (4.19) takes the form:

$$V^{S,*}(x) = \sum_{e\in\Gamma_S(x)} \frac{\lambda_e}{\Lambda(x)}\left[r(x,e) + \gamma_e \sum_{y\in X} p(y|e,x) V^{S,*}(y)\right] \tag{4.23}$$

for states $x \notin X_{rs}$, i.e., states where the only available action is the idling action. Additionally, we need to calculate the event specific discount factors:

$$\gamma_e = \int_0^\infty e^{-\beta\tau} F(d\tau|x,e) = \frac{\lambda_e}{\beta + \lambda_e}$$

and

$$V^{S,*}(x) = \sum_{e \in \Gamma_S(x)} \frac{\lambda_e}{\Lambda(x)} \left[r(x,e) + \frac{\lambda_e}{\beta + \lambda_e} \sum_{y \in X} p(y|e,x) V^{S,*}(y) \right] \tag{4.24}$$

again for $x \notin X_{rs}$.

4.4.2 Properties of the Bellman Operator

Ultimately, the goal in this chapter is to use a reinforcement learning algorithm in the controller, so it can optimize the control policy within the bounds imposed by the supervisor, and also to derive some convergence results about such algorithm. For that, we will start by exploring some properties about the Bellman operator $\mathbf{H} : \mathcal{Q} \to \mathcal{Q}$, with $\mathcal{Q}$ representing the space of all bounded q-functions, given by:

$$(\mathbf{H}q)(x,a) = \sum_{e \in \Gamma_S(x)} p(e|x,S,a) \left[r(x,e) + \gamma_e \sum_{y \in X} p(y|e,x) \max_{b \in A_y} q(y,b) \right] \tag{4.25}$$

Ideally, we would want this operator to be a contraction mapping in some norm, similarly to what was shown for SMDPs in Chapter 2. However, that is not always the case because of the existence of immediate events which induce an undiscounted transition, as described in Equation (4.21). Put another way we can say that:

$$\gamma_e = 1 \quad , \quad e \in E_U$$

As for the other timed events, let us make a similar assumption as to the one in Section 2.2.4.

Assumption 4.4.1. *For each event $e \in E_T$, there exist $\delta_e > 0$ and $\epsilon_e > 0$ such that:*

$$F_e(\delta_e) < 1 - \epsilon_e$$

Under the conditions of Assumption 4.4.1 we can prove the following result:

Lemma 4.4.1. *Suppose Assumption 4.4.1 holds. Then $\gamma_e < 1$ for every $e \in E_T$*

Proof. We have that $\gamma_e = \int_0^\infty e^{-\beta t} F_e(dt)$ and following a similar derivation from the one in Section 2.2.4:

$$\begin{aligned}\gamma_e &= \int_0^\infty e^{-\beta t} F_e(dt) = \\ &= \int_0^{\delta_e} e^{-\beta t} F_e(dt) + \int_{\delta_e}^\infty e^{-\beta t} F_e(dt) \leq \\ &\leq \int_0^{\delta_e} F_e(dt) + e^{-\beta\delta_e} \int_{\delta_e}^\infty F_e(dt) = \\ &= F_e(\delta_e) + e^{-\beta\delta_e}(1 - F_e(\delta_e)) = \\ &= e^{-\beta\delta_e} + F_e(\delta_e)(1 - e^{-\beta\delta_e}) < \\ &< e^{-\beta\delta_e} + (1 - \epsilon_e)(1 - e^{-\beta\delta_e}) < \\ &< 1\end{aligned}$$

□

So we have that $\gamma_e = 1, e \in E_U$ and $\gamma_e < 1, e \in E_T$. For this reason, we are in the condition to say that the operator $\mathbf{H}$ as defined in Equation (4.25) is a non-expansive map.

Theorem 4.4.2. *Suppose we have a C-GSTA with every $e \in E_T$ satisfying Assumption 4.4.1. Then the operator $\mathbf{H}$:*

$$(\mathbf{H}q)(x, a) = \sum_{e\in\Gamma_S(x)} p(e|x, S, a) \left[r(x, e) + \gamma_e \sum_{y\in X} p(y|e, x) \max_{b\in A_y} q(y, b) \right]$$

satisfies a Lipschitz condition with $K = 1$ for the distance defined by the sup norm.

Proof. We have:

$$\begin{aligned}&\|\mathbf{H}q_1 - \mathbf{H}q_2\|_\infty = \\ &\quad = \max_{x,a} \left| \sum_{e\in\Gamma_S(x)} p(e|x, S, a)\gamma_e \sum_{y\in X} p(y|e, x) \left(\max_{b_1\in A_y} q_1(y, b_1) - \max_{b_2\in A_y} q_2(y, b_2) \right) \right| \leq \\ &\quad \leq \max_{x,a} \sum_{e\in\Gamma_S(x)} p(e|x, S, a)\gamma_e \sum_{y\in X} p(y|e, x) \cdot \max_{z,b} |q_1(z, b) - q_2(z, b)| =\end{aligned}$$

Since $\gamma_e \leq 1$ for all events:

$$\begin{aligned}\|\mathbf{H}q_1 - \mathbf{H}q_2\|_\infty &\leq \max_{x,a} \sum_{e\in\Gamma_S(x)} p(e|x, S, a) \sum_{y\in X} p(y|e, x) \cdot \max_{z,b} |q_1(z, b) - q_2(z, b)| = \\ &= \max_{x,a} \sum_{e\in\Gamma_S(x)} p(e|x, S, a) \cdot \max_{z,b} |q_1(z, b) - q_2(z, b)| = \\ &= \max_{x,a} \|q_1 - q_2\|_\infty = \\ &= \|q_1 - q_2\|_\infty\end{aligned}$$

□

Particularly, we can also prove that $\|\mathbf{H}^n q_1 - \mathbf{H}^n q_2\|_\infty \leq \|q_1 - q_2\|_\infty$ for any $n > 0$.

This is not enough to guarantee the existence of a unique fixed point of the operator, and the value iteration algorithm given by:

$$q^{n+1} = \mathbf{H}q^n$$

is not guaranteed to converge. So, we need additional conditions on the system, particularly concerning the immediate events which are the ones that cause the convergence problems. In fact, like mentioned previously, it is possible to construct a system where we have an infinite cycle of immediate events and since the associated transitions are not discounted (and take 0 time), we would reach a situation where in null time the expected reward would diverge to infinity. Assumption 4.4.1 works to bound the expected reward for timed transitions and we need an assumption that does the same for untimed ones.

Let us introduce some definitions:

Definition 4.4.1. *A string $s \in \mathcal{L}(G)$ is said to be* immediate *if it is not null and only contains immediate events:*

$$s \in E_U^* \setminus \{\varepsilon\}$$

and

Definition 4.4.2. *A GSTA is said to have* immediate cycles *if for some state $x \in X$ it is possible to revisit it through an immediate string. Considering the extension of the transition function to strings:*

$$\exists_{s \in E_U^* \setminus \{\varepsilon\}} p(x|x,s) > 0$$

Since only states in $x \in X_{rs}$ have immediate events active, this will never happen for any state $x \notin X_{rs}$

Now we will make the following assumption:

Assumption 4.4.2. *The GSTA does not contain immediate cycles.*

$$\forall_{x \in X} \forall_{s \in E_U^* \setminus \{\varepsilon\}} p(x|x,s) = 0$$

With this assumption, the maximum number of immediate transitions that can occur before the system changes to a state where only events in E_T are active is equal to $|X_{rs}|$. In practical terms, since immediate events are often associated with starting or stoping an agent action, what this does is not allowing for indefinite of such starts and stops, forcing the system to wait for something to happen before choosing to start a given action again (or stop it). Still, it does allow for the decomposition of complex actions since it is still possible to chain different immediate events in different *random switch* states.

Under this assumption we can prove the following result:

Lemma 4.4.3. *Suppose we have a supervised C-GSTA that follows Assumption 4.4.2. Then for every state $x \in X_{rs}$ and for the two disjoint sets $V(x) \subset X_{rs}$ and $NV(x) \subset X_{rs}$, with $V(x) \cup NV(x) = X_{rs}$, defined as*[7]*.*

$$V(x) = \left\{ y \in X_{rs} : \forall_{s \in E_U^* \setminus \{\varepsilon\}} \; p(y|x, s) = 0 \right\}$$

$$NV(x) = \left\{ y \in X_{rs} : \exists_{s \in E_U^* \setminus \{\varepsilon\}} \; p(y|x, s) > 0 \right\}$$

we have that:

1. $y \in NV(x) \Rightarrow x \in V(y)$

2. $\forall_{y \in NV(x)} NV(y) \subset NV(x)$

Particularly:

$$\exists_{x \in X_{rs}} \; NV(x) = \emptyset$$

$$\forall_{x \in X_{rs}} \; x \in V(x)$$

Note that $V(x)$ represents the set of random switch states that are not accessible from x through an immediate string, while $NV(x)$ represents the set of random switch states that have a non-null probability of being reached from x through an immediate string.

Proof. Let us start by assuming that, for some $x, y \in X_{rs}$ we have that $y \in NV(x)$. Then there is an $s \in E_U^* \setminus \{\varepsilon\}$ such that $p(y|x, s) > 0$. Put another way, there is a potential path from x to y that only includes immediate events. Because of Assumption 4.4.2, the reverse cannot be true, i.e, there must not be a path consisting of only immediate events from y to x. Put another way:

$$\forall_{s \in E_U^* \setminus \{\varepsilon\}} p(x|y, s) = 0$$

which means that $x \in V(y)$.

As for the second result, if we assume it does not hold, i.e., that there is an $y \in NV(x)$ such that $NV(y) \not\subset NV(x)$, then we can say that:

$$\exists_{z \in NV(y)} z \notin NV(x)$$

and so, for some $s' \in E_U^* \setminus \{\varepsilon\}$:

$$p(z|y, s') > 0$$

while for all possible strings in $s \in E_U^* \setminus \{\varepsilon\}$

$$p(z|x, s) = 0$$

[7] For this definition we have to extend the transition function so that, if $e \notin \Gamma_S(x)$ then $p(y|e, x) = 0$ for all $y \in X$.

Since we had that $y \in NV(x)$, then $p(y|x,s'')$ for some $s'' \in E_U^* \backslash \{\varepsilon\}$. The concatenation of two strings made of immediate events is still a string made of immediate events. So, for $s = s's''$ we have that:

$$p(z|x,s) \geq p(z|y,s')\, p(y|x,s'') > 0$$

contradicting the fact that $z \notin NV(x)$.

If we make $y = x$ and apply the first property we have:

$$x \in NV(x) \Rightarrow x \in V(x)$$

which is impossible since $NV(x) \cap V(x) = \emptyset$. So we must have that $x \in V(x)$.

Finally, by applying the second property of nested NV sets successively, we end up reaching a point where, for some $x \in X_{rs}$ we have that $NV(x) = \emptyset$.

□

In other words, what the lemma says is that, under Assumption 4.4.2, if a state y is reachable from x through an immediate string:

- It is not possible to reach x through an immediate string starting from y.
- The set of states reachable through immediate strings decreases monotonically. Particularly, there is at least one random switch state from which is not possible to reach any other random switch through an immediate string.

Figure 4.5 shows an example of how the random switch states, if under Assumption 4.4.2, together with the immediate events form an acyclic directed sub-graph of the automaton.

We can use these properties induced by the non existence of cycles of random switches to show that, with some periodicity, operator $\mathbf{H}$ does contract, just not on every iteration, which is shown in the following result:

Theorem 4.4.4. *Suppose we have a supervised C-GSTA that follows Assumptions 4.4.2 and 4.4.1. Then the operator:*

$$(\mathbf{H}q)(x,a) = \sum_{e \in \Gamma_S(x)} p(e|x,S,a) \left[r(x,e) + \gamma_e \sum_{y \in X} p(y|e,x) \max_{b \in A_y} q(y,b) \right]$$

is a local power contraction mapping.

Proof. We have:

$$\|\mathbf{H}q_1 - \mathbf{H}q_2\|_\infty =$$

$$= \max_{x,a} \left| \sum_{e \in \Gamma_S(x)} p(e|x,S,a) \gamma_e \sum_{y \in X} p(y|e,x) \left(\max_{b_1 \in A_y} q_1(y,b_1) - \max_{b_2 \in A_y} q_2(y,b_2) \right) \right|$$

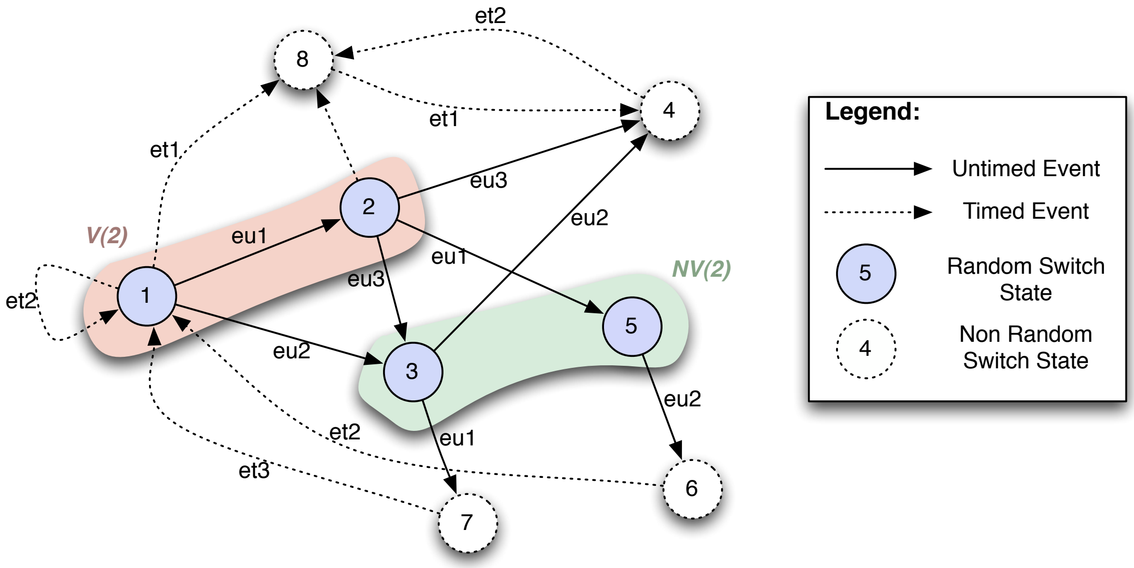


Figure 4.5: Random switch states forming an acyclic sub-graph of the STA.

We will start by noting that, for every state not in X_{rs} (or if the idling action is chosen), we have that the only events active are timed ones, or $\Gamma_S(x) \cap E_U = \emptyset$. So, for $x \notin X_{rs}$ or $a = \varepsilon_a$:

$$\begin{aligned}
&|\mathbf{H}q_1(x,a) - \mathbf{H}q_2(x,a)| = \\
&\qquad = \left| \sum_{e \in \Gamma_S(x)} p(e|x,S,a)\gamma_e \sum_{y \in X} p(y|e,x) \left(\max_{b_1 \in A_y} q_1(y,b_1) - \max_{b_2 \in A_y} q_2(y,b_2) \right) \right| \leq \\
&\qquad \leq \left| \sum_{e \in \Gamma_S(x)} p(e|x,S,a)\gamma_e \sum_{y \in X \setminus V(x)} p(y|e,x) \max_{z,b} |q_1(z,b) - q_2(z,b)| \right| \leq \\
&\qquad \leq \gamma_{max} \max_{z \in X} |q_1(z,b) - q_2(z,b)| = \\
&\qquad = \gamma_{max} \|q1 - q2\|
\end{aligned}$$

where $\gamma_{max} = \max_{e \in E_T} \gamma_e$, since $p(e|x,S,a) = 0$ for $e \in E_U$ and $x \notin X_{rs}$.

If we now apply Theorem 4.4.2 several times, we can also say that, for every $n > 0$:

$$|\mathbf{H}^n q_1(x,a) - \mathbf{H}^n q_2(x,a)| \leq \gamma_{max} \|q1 - q2\|$$

On the other hand, for every possible immediate event, if a state x is a random switch, $p(y|x, e) = 0$ if $y \in V(x)$ and then we can write that:

$$\begin{aligned}
&|\mathbf{H}q_1(x, a) - \mathbf{H}q_2(x, a)| = \\
&\qquad = \left| \sum_{e \in \Gamma_S(x)} p(e|x, S, a)\gamma_e \sum_{y \in X \setminus V(x)} p(y|e, x) \left(\max_{b_1 \in A_y} q_1(y, b_1) - \max_{b_2 \in A_y} q_2(y, b_2) \right) \right| \leq \\
&\qquad \leq \left| \sum_{e \in \Gamma_S(x)} p(e|x, S, a)\gamma_e \sum_{y \in X \setminus V(x)} p(y|e, x) \max_{z \in X \setminus V(x), b} |q_1(z, b) - q_2(z, b)| \right|
\end{aligned}$$

for every $x \in X_{rs}$. Since $\gamma_e \leq 1$ for all events:

$$\begin{aligned}
|\mathbf{H}q_1(x, a) - \mathbf{H}q_2(x, a)| &\leq \\
&\leq \max_{z \in X \setminus V(x), b} |q_1(z, b) - q_2(z, b)|
\end{aligned}$$

So we can say that, for states corresponding to random switches:

$$|\mathbf{H}^n q_1(x, a) - \mathbf{H}^n q_2(x, a)| \leq \max_{z \in X \setminus V(x), b} \left|\mathbf{H}^{n-1} q_1(z, b) - \mathbf{H}^{n-1} q_2(z, b)\right|$$

Since on the maximization we only need to consider states not in $V(x)$ for each $x \in X_{rs}$, we can apply the same reasoning to $|\mathbf{H}^{n-1} q_1(z, b) - \mathbf{H}^{n-1} q_2(z, b)|$. Eventually, we will reach a situation where $NV(x) = \emptyset$, as guaranteed by Lemma 4.4.3, or put another way $V(x) = X_{rs}$. That happens for some $n < |X_{rs}|$.

$$|\mathbf{H}^n q_1(x, a) - \mathbf{H}^n q_2(x, a)| \leq \max_{z_n \in X \setminus V(x), b} \dots \max_{z_0 \in X \setminus X_{rs}, b} |\mathbf{H}q_1(z_0, b) - \mathbf{H}q_2(z_0, b)|$$

And since now the maximization is done only over states that do not correspond to random switches, from Equation 4.4.4:

$$|\mathbf{H}q_1(z_0, b) - \mathbf{H}q_2(z_0, b)| \leq \gamma_{max} \|q1 - q2\|$$

since $z_0 \notin X_{rs}$. Finally, we have:

$$\begin{aligned}
|\mathbf{H}^n q_1(x, a) - \mathbf{H}^n q_2(x, a)| &\leq \\
&\leq \max_{z_n \in X \setminus V(x), b} \dots \max_{z_0 \in X \setminus X_{rs}, b} \gamma_{max} \|q1 - q2\| \\
&\leq \gamma_{max} \|q1 - q2\|
\end{aligned}$$

In general, at least for $N = |X_{rs}|$ every $x \in X_{rs}$ will the satisfy relation. So for every possible state and action we can say that:

$$|\mathbf{H}^N q_1(x, a) - \mathbf{H}^N q_2(x, a)| \leq \gamma_{max} \|q1 - q2\|$$

and for this reason:

$$\|\mathbf{H}^N q_1 - \mathbf{H}^N q_2\| \leq \gamma_{max} \|q1 - q2\|$$

which proves the operator **H** is a local power contraction mapping (see Definition A.2.3).

□

So now we can use Theorem A.2.2 to guarantee the existence of a fixed point of **H**, i.e., a point q^* such that $\mathbf{H}q^* = q^*$.

4.4.3 Q-learning Rule

Since the our supervised system is equivalent to an SMDP, we use the same update rule for the reinforcement learning based controller:

$$Q_{k+1}^S(x_k, a_k) = (1 - \alpha_k) Q_k^S(x_k, a_k) + \alpha_k \tilde{Q}_{k+1}^S(x_k, a_k) \tag{4.26}$$

with

$$\tilde{Q}_{k+1}^S(x_k, a_k) = r_k + e^{-\beta\tau_k} \max_{b \in A_{x_{k+1}}} Q_k^S(x_{k+1}, b)$$

with a modification to account for the fact that the lump reward is received in the end of the transition and not right after the decision is made. For this reason, we now have:

$$r_k = e^{-\beta\tau_k}\kappa_k + \frac{1 - e^{-\beta\tau_k}}{\beta}\mu_k$$

To prove the convergence of this update rule we use a derivation similar to the one in Theorem 2.2.3.

Theorem 4.4.5. *Given a finite C-GSTA, the Q-learning sequence $\{Q_k\}$, given by the update rule:*

$$Q_{k+1}(x_k, a_k) = (1 - \alpha_k(x_k, a_k)) Q_k(x_k, a_k) + \alpha_k(x_k, a_k) \left(r_k + e^{-\beta\tau_k} \max_{b \in A_{x_{k+1}}} Q_k(x_{k+1}, b) \right)$$

with

$$r_k = e^{-\beta\tau_k}\kappa_k + \frac{1 - e^{-\beta\tau_k}}{\beta}\mu_k$$

converges with probability 1 to the optimal Q-function if:

- *Assumption 4.4.1 holds.*
- *Assumption 4.4.2 holds.*
- $0 \leq \alpha_k(x_k, a_k) \leq 1$, $\sum_k \alpha_k(x_k, a_k) = \infty$ *and* $\sum_k \alpha_k^2(x_k, a_k) < \infty$*;*
- k_k *and* c_k *are bounded.*

Proof. The convergence of the Q-learning update rule can be proved by making use of Theorem 2.2.3 and the fact that the Bellman operator $\mathbf{H}$ has a unique fixed point, as shown by Theorem 4.4.4.

□

4.5 A Practical Case

We applied our approach to a simulated domain where a robot navigates in the corridors of a building, with the goal of interacting with people he meets along the way and learning to lead them to 2 specific locations, according to their preference. Figure 4.6 represents the environment in which the robot is navigating, as well as the goal states and initial starting point of the robot. After reaching a goal state, the robot keeps doing its task from where it is, possibly wandering around the corridors waiting to find more people to help out.

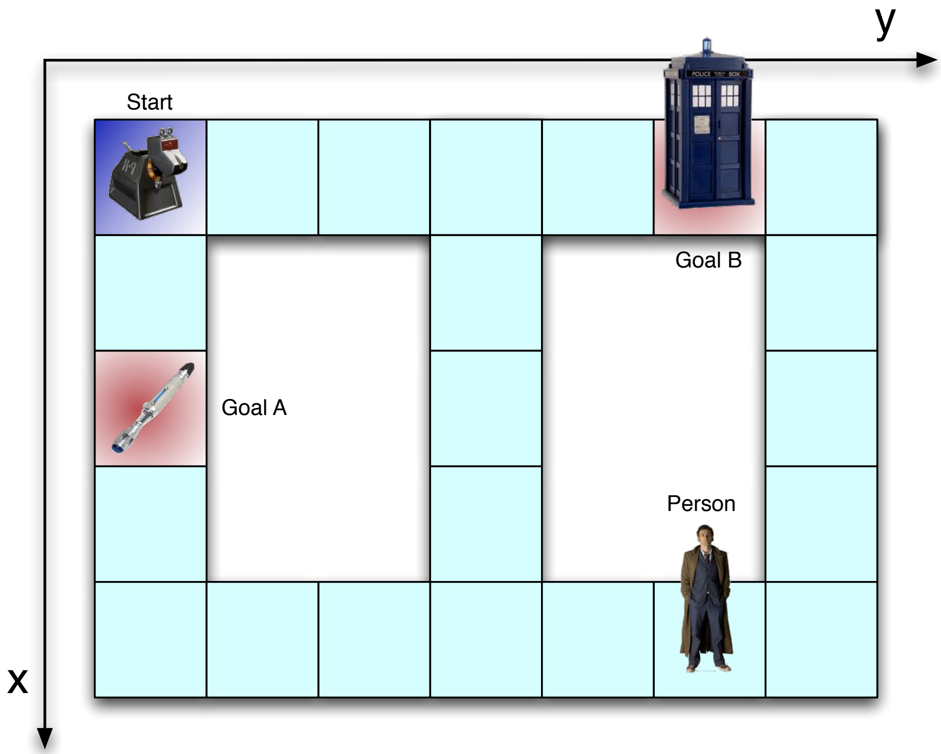


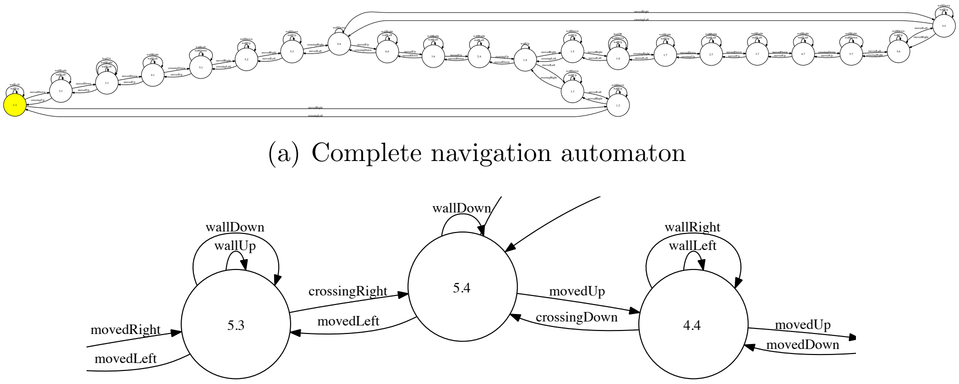
Figure 4.6: Map of the environment where the robot lives.

4.5.1 Environment Model

While navigating in the environment, the robot recognizes a set of navigation events:

- {`movedUp, movedDown, movedLeft, movedRight`} represent a situation where the robot changed cell while moving in a certain direction.
- {`crossingUp , crossingDown, crossingLeft, crossingRight`} are similar but only occur when the robot enters a corner or intersection.
- {`wallUp , wallDown, wallLeft, wallRight`} indicate the robot tried to navigate in a certain direction but detected a wall when doing so.

Figure 4.8(a) shows an automaton model for navigation representing the map of Figure 4.6, with the navigation events, and Figure 4.8(b) shows a detail of that automaton.



(a) Complete navigation automaton

(b) Navigation automaton detail

Figure 4.7: Representation of the navigation map as an automaton. The states represent $x.y$ coordinates of the map cells from Figure 4.6.

Additionally, we considered automata for the robot actions, representing the possible effects they can have on the environment. The events that start a robot action were:

- {`startUp, startDown, startLeft, startRight`} start a navigation action in a certain direction.
- {`startCall`} starts an action that tries to approach a person to get her attention.
- {`startAsk`} starts an action that asks a person for the location they intend to go to.

Besides these events corresponding to the start of an action, we modeled an additional event, `stopAction`, that stops all actions currently running. We could also consider individual stop actions for each of the possible robot actions but, in this problem, we modeled the supervisor to allow only one behavior to be run at a given time, making more stopping actions redundant. These events to start or stop a robot action are the only ones considered to be controllable. Figure 4.8 shows the general scheme for modeling the effects of a robot action with an automaton.

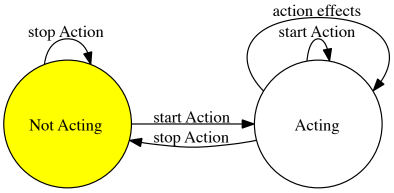


Figure 4.8: Environment automaton than models a general action's effects.

Finally, we considered another automaton which models the interaction between the robot and a person he meets along the way. Additional uncontrollable events generated by the environment include:

- {`metPerson, gotAttention, choseA, choseB, declinedHelp`}, that concern the actual interaction between the robot and the person.
- {`lead2A, lead2B`} are fired when the robot leads the person to the place they previously chose to go.
- {`bored`} represents the moment a person gives up on the robot's help, from having spent to much time in the same map cell.

These events, as well as the ones mentioned previously that describe navigation of the robot in the environment, are all considered to be uncontrollable.

In Figure 4.9 the interaction automaton is shown.

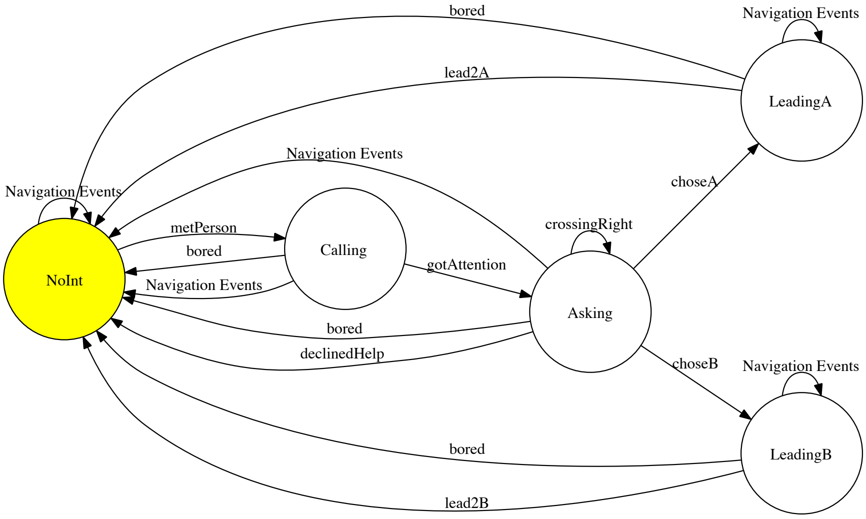


Figure 4.9: Environment automaton that models the interaction between the robot and a person.

As mentioned in Section 4.1, the controllable events are considered to fire immediately and the uncontrollable ones to have an associated time distribution. In this example, the uncertainty in the robot's actions stems from the underlying uncertainty associated with the firing of different events.

In Table 4.1, we present a summary of the distributions associated with each one of the timed events. Note that some are not exponential or the system would actually be fully Markovian (Cassandras and Lafortune, 2007).

The automata presented previously were used to model and simulate the environment and can be composed to analyze the supervised and controlled system but, from a decision standpoint, the supervisor and controller only have direct access to the events the system generates.

Events	Distribution Class	Expected Value
`movedUp ,movedDown, movedLeft, movedRight crossingUp , crossingDown, crossingLeft, crossingRight`	Raleygh	2.50
`wallUp , wallDown, wallLeft, wallRight`	Rayleigh	1.25
`choseA, choseB, declinedHelp`	Exponential	0.50
`lead2A, lead2B`	Exponential	0.10
`bored`	Rayleigh	18.80
`metPerson`	Exponential	10.00
`gotAttention`	Exponential	1.00

Table 4.1: Distribution parameters for the uncontrollable events.

4.5.2 Supervisor Model

We used 3 different supervisors in parallel to steer the behavior of the system and allow the learner to more efficiently converge to the real Q-function.

The first simply forces the action choice to follow a (`Start` → `Wait` → `Stop`) pattern. We applied this supervisor to the system in all tests, which guarantees that Assumption 2.2.1 is always met. We'll denote this supervisor by $S_{1Action}$.

The second supervisor we considered concerns the navigation actions. It can be thought as a map of the building which does not allow the robot to try actions that will certainly lead him against a wall. A detail of the automaton that implements can be seen in Figure 4.10. Note that it is similar to the one that represented the map in the system model but it includes the controllable events that are or are not allowed in each state and does not include the wall-related events. We'll denote this supervisor by

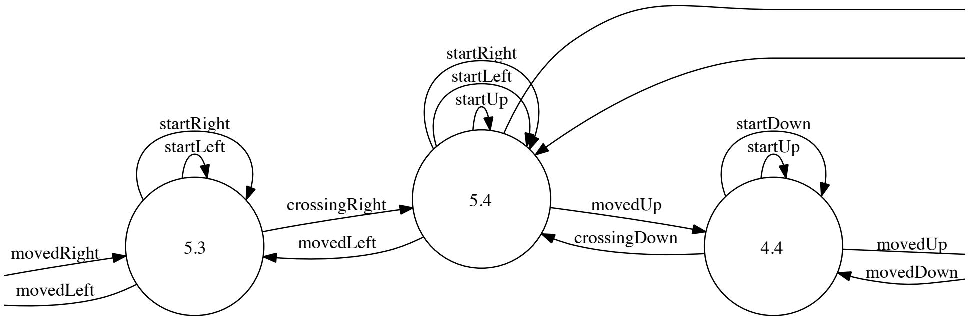


Figure 4.10: Supervisor that regulates the navigation of the robot to avoid going in the direction of walls.

S_{Map}.

The third supervisor concerns the interaction procedure between the robot and a person and, essentially, forces a small plan on the robot every time it meets a person in any given map cell. Figure 4.11 represents it. The plan goes like this:

1. When the robot meets a person, the supervisor forces it to get the person's attention, through action `startCall`.
2. When the robot has the attention of a person, the supervisor forces it to ask for the goal room, through action startAsk.
3. Finally, when a person chooses a room the supervisor will allow for navigation actions to be made until the room is reached.

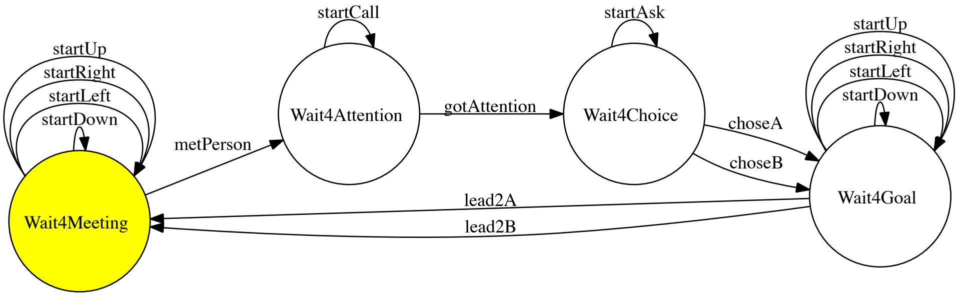


Figure 4.11: Supervisor that regulates the interaction between the robot and a person.

We'll denote this supervisor by S_{Int}.

We did 4 tests under full observability (the observer automata have an identical structure to the ones that model the system): unsupervised, interaction supervised, map supervised and both. The supervisors were combined using Modular Supervisory Control as described in Section 2.1.3 and Figure 4.12 shows the block diagrams for all the 4 tests[8].

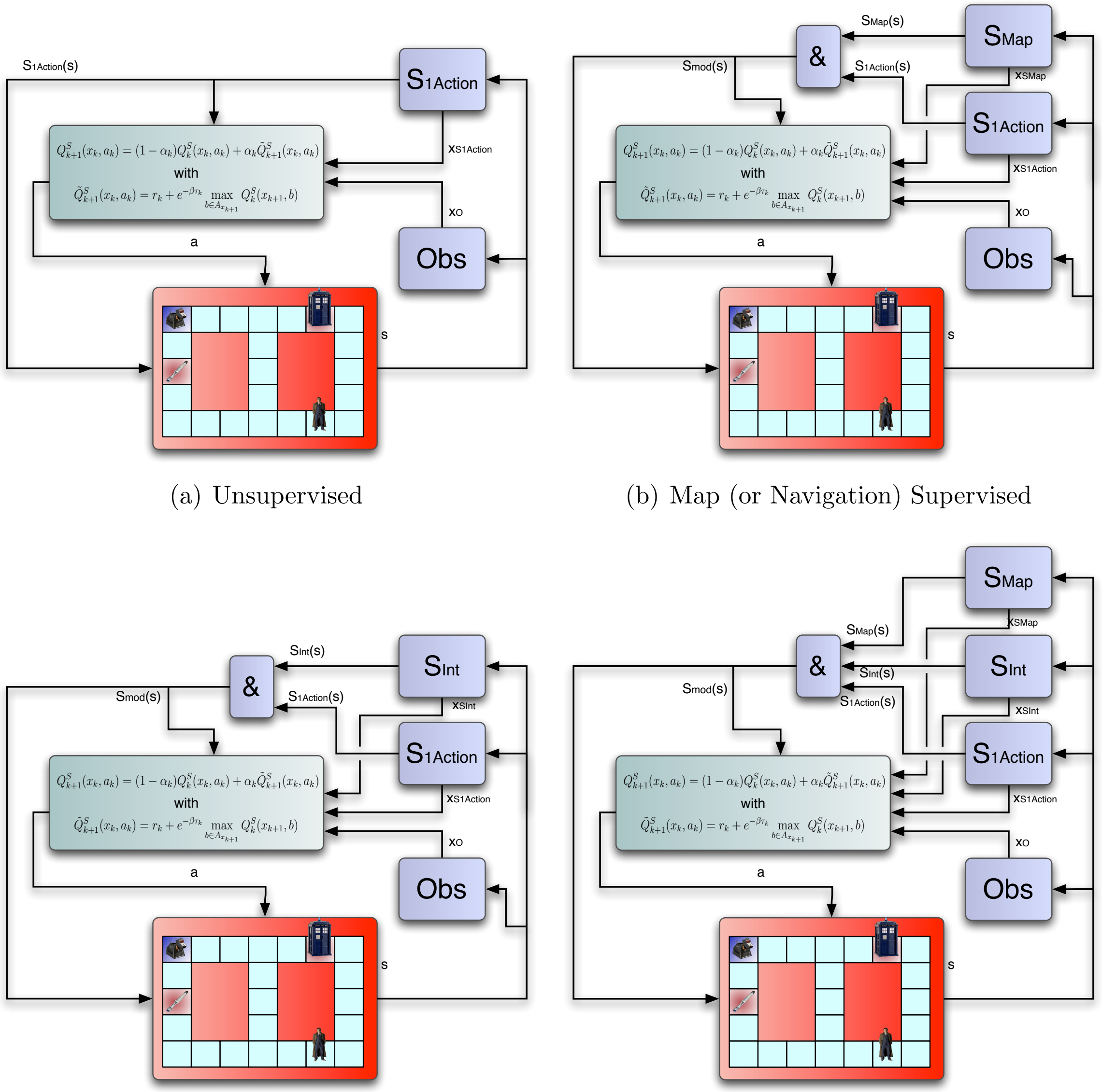
4.5.3 Controller Model

We used the modified Q-learning algorithm for SMDPs described in Theorem 4.4.5, with the available actions at each state being given by the supervisor and the state being the cartesian product of the observer state and supervisor state. As learning parameters we used $\beta = 0.1$ (equivalent to a discrete time MDP discount factor of $\gamma = 0.9$ for a time interval of 1 unit), $\alpha = 0.2$ (not too large since some events are not dependent on the robot making the tasks non deterministic) and $\epsilon = 0.1$ with a simple ϵ-greedy exploration strategy (Sutton and Barto, 1998). Table 4.2 shows what are the individual equivalent discount factors for each of the uncontrollable events, when we use $\beta = 0.1$ and assume the inter-event times are distributed according to Table 4.1 [9] [10].

[8] As mentioned, the supervisor that forces only one action to run at each time was used for all tests in order to ensure Assumption 4.4.2 was always satisfied.

[9] λ is the parameter (rate) of the exponential distribution and $E[X] = \frac{1}{\lambda}$ if $X \sim \mathbf{Exponential}(\lambda)$.

[10] σ is the parameter of the Raleygh distribution and $E[X] = \sigma\sqrt{\frac{\pi}{2}}$ if $X \sim \mathbf{Raleygh}(\sigma)$.


S1Action(s)
S1Action
xS1Action
xO
Obs
a
s
$Q^S_{k+1}(x_k,a_k)=(1-\alpha_k)Q^S_k(x_k,a_k)+\alpha_k\tilde{Q}^S_{k+1}(x_k,a_k)$
with
$\tilde{Q}^S_{k+1}(x_k,a_k)=r_k+e^{-\beta\tau_k}\max_{b\in A_{x_{k+1}}}Q^S_k(x_{k+1},b)$
SMap(s)
SMap
xSMap
&
Smod(s)
SInt(s)
SInt
xSInt

(a) Unsupervised

(b) Map (or Navigation) Supervised

(c) Interaction Supervised

(d) Fully Supervised

Figure 4.12: Block diagrams for the 4 different tests.

We assumed every test to run with $S_{1Action}$ by default and, for that reason, when we say, e.g., *Unsupervised* we are referring to the absence of other two supervisors. The same reasoning can be applied for the other test labels.

Events	γ_e **Calculation**	γ_e **Value**

`movedUp ,movedDown, movedLeft, movedRight`
`crossingUp , crossingDown, crossingLeft,`
`crossingRight`
	$\int_0^\infty \frac{t}{\sigma^2} e^{-\frac{t^2}{2\sigma^2}-\beta t} dt$	0.78

`wallUp , wallDown, wallLeft, wallRight`	$\int_0^\infty \frac{t}{\sigma^2} e^{-\frac{t^2}{2\sigma^2}-\beta t} dt$	0.88
`choseA, choseB, declinedHelp`	$\int_0^\infty e^{-(\beta+\lambda)t} dt = \frac{\lambda}{\beta+\lambda}$	0.95
`lead2A, lead2B`	$\int_0^\infty e^{-(\beta+\lambda)t} dt = \frac{\lambda}{\beta+\lambda}$	0.99
`bored`	$\int_0^\infty \frac{t}{\sigma^2} e^{-\frac{t^2}{2\sigma^2}-\beta t} dt$	0.23
`metPerson`	$\int_0^\infty e^{-(\beta+\lambda)t} dt = \frac{\lambda}{\beta+\lambda}$	0.5
`gotAttention`	$\int_0^\infty e^{-(\beta+\lambda)t} dt = \frac{\lambda}{\beta+\lambda}$	0.91

Table 4.2: Equivalent event-based discount factors for the uncontrollable events.

For rewards, we gave a reward of 100 for completing the task of leading a person to the appropriate room successfully, and thus receiving events { `lead2A` , `lead2B` }, a reward of -50 for abandoning a task after beginning interaction, a reward of -50 every time the person got tired of waiting and sent the `bored` event and a reward of -10 for bumping into a wall. The goal was to have the robot learn how interact with a person it met in the building and learn how to lead the person to one of two possible rooms, chosen by the person through the interaction procedure.

To speed up the learning process, particularly because the supervisors occasionally force the system to choose only one action or to wait for an unobservable event to occur (an idling action), we only applied the learning algorithm in the states for which $\#actions \geq 2$. Nevertheless, we kept updating the reward to account for changes in its value between states where an actual decision was needed.

4.5.4 Observer Model

Under full observability, the observer models were identical in structure to the environment automata, i.e., the controller had access to the exact state the environment was in, albeit not to the transition probabilities and time distributions of the associated events.

4.6 Results

The evolution of the total accumulated rewards over time can be seen in Figure 4.13; all tests ran for T=50000. We can see how the fully supervised system converges to a steady reward rate much faster than all the other ones, since it does not need to learn the correct sequence of interaction actions or the location of the walls. Among the unsupervised (or partially unsupervised) cases, the ones without the interaction supervisor suffer the most because failure to complete the full interaction action sequence, until the person agrees on help and communicates the place he wants to go to, means the robot has to wait for another person and start the process all over.

Eventually, all systems start to approach a similar reward rate after having learned the complete task and the location of the walls, with the difference being explained by exploration factors, which again affect the cases without an interaction supervisor the most.

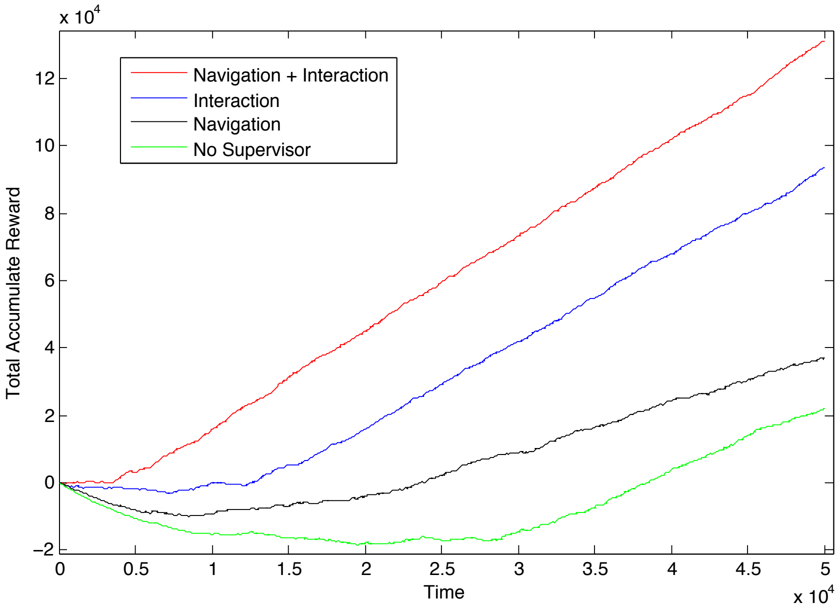


Figure 4.13: Total accumulated rewards over time under full observability for different supervisory schemes, for a test with T=50000.

In Table 4.3 it is possible to see statistics of the interactions for each of the cases. Over the course of T=50000 the fully supervised system managed to reach 60.6% successful tasks since it converged faster, with the failure rate being only of 6.1% and the remaining cases corresponding to when help was simply declined by the person. The systems with supervised navigation performed slightly better than their counterparts, both with or without supervised interaction. The systems with supervised interaction had no task abandons, by supervisor design, and those without supervised interaction abandoned the task before completion roughly around 14% of the times.

In Figure 4.14 we show a detail of the initial instants of the experiment's path to highlight how inter-event time is not discrete and is, instead, regulated by the associated cdfs.

Supervisors Used	**# interactions**	**% leads to goal**	**% person gave up**	**% robot gave up**	**% help declined**
None	2353	34.8%	26.1%	14.7%	28.1%
Interaction	2362	57.7%	7.8%	0%	34.5%
Navigation	2473	36.8%	24.9%	14.0%	24.3%
Both	2655	60.6%	6.1%	0%	33.2%

Table 4.3: Task statistics under full observability for a test with T=50000.

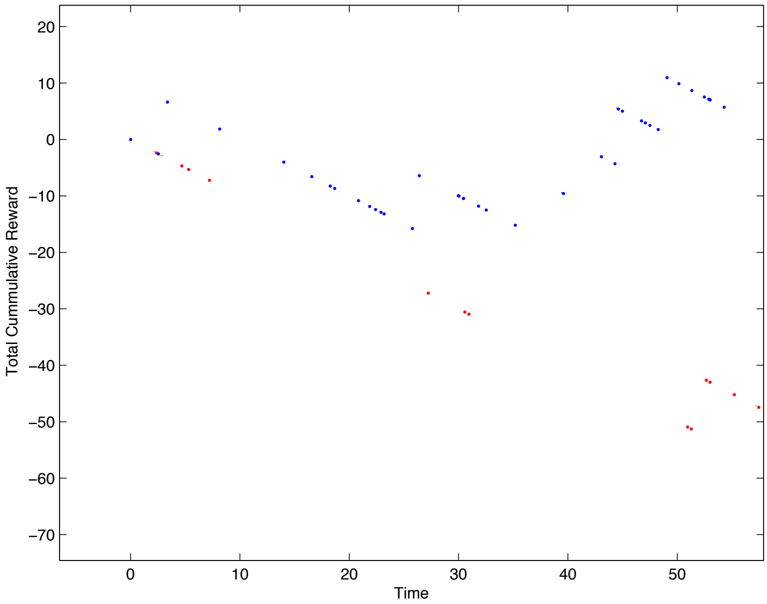


Figure 4.14: Detail of the cummulative rewards over time showing how time between events is not discrete.

4.7 Summary

In this chapter we established the building blocks of our agent control framework. We assumed the systems to be controlled by the agent generated Generalized Semi-Markov Processes and could be modeled by STA.

We classified the events of the STA according to their controllability and the time it takes for them to occur. In Table 4.4 we summarize our interpretation of what each kind of events represent in the environment, which we maintain throughout the thesis. Furthermore, in a single agent scenario it is quite common that two kinds of events, immediate controllable events and timed uncontrollable events, suffice to model a large number of applications. In fact, these two kinds encompass the most fundamental aspects of the behavior of an agent acting in an environment: the start/stop of an agents' actions and the effects of agents' actions.

	Timed	**Untimed**
Controllable	All timed events were assumed to be uncontrollable.	• Start/Stop commands of the agent.
Uncontrollable	• Effects of agent's actions. • Effects of external processes.	• Start/Stop commands of other agents. • Immediate effects of external processes. • Observations.

Table 4.4: Semantics of the different kinds of events.

In our modeling we assumed the correspondence of events to processes in the environment was given by this table but, nevertheless, most of the results are general enough to allow other interpretations.

Additionally, we presented a definition of full observability w.r.t. the state of the system and derived the necessary and sufficient conditions for the existence of full observability, considering an observer only has access to the strings generated by the system and not its state directly. One of the effects of these conditions is that the observer can be represented by a DFA. The supervisor was also assumed to be represented by a DFA, and for that reason the language generated by the supervised system is regular.

We derived the Bellman optimality equations for the reinforcement learning based controller and shown that under some assumptions a Q-learning rule could be proven to converge to the optimal value function. The choice of actions for the controller is based on the controllability of immediate events and, if no immediate uncontrollable events exist, the emission probabilities of the controllable events will be directly related to the value of the policy for the corresponding action. The choice of which controllable event to fire can be deterministic or probabilistic according to the controller policy.

Finally, we presented a case study of a robot navigating in a building environment, whose goal was to lead visitors to one of two rooms, and shown how the introduction of a supervisor improved the learning task.

Chapter 5

Partial Observability: Deterministic Observer

Full observability of a Stochastic Time Automata, as defined in the previous chapter, requires an observer to be able to univocally identify the state of the system by looking at the string of events that it outputs. For this to happen, the transition function of the STA (defined in terms of states and events) must be deterministic and all events must be observed. The questions that naturally arise are:

- *What happens if one of the observability assumptions is broken?*
- *What can we do under partial observability?*

Answering those questions is closely tied to the observer we use to obtain state estimates, the amount of knowledge about the system embedded in said observer and the type of representation the observer uses. When assuming full observability, we saw how the observer can be represented by a function which, essentially, mimics the system's transition function, which allows us to use a FSA as observer with the transition function being isomorphic with the transition function of the system.

The main difference between this observer automaton and the system itself is the fact that, since the observer is built as a recognizer for the language generated by the system, it has no information about the parameters that determine the firing of events: random switch probabilities for events that fire immediately and clock structure for timed events. The observer retains the logic structure of the STA without maintaining information about its stochastic part. This observer is deterministic by its own nature, which as mentioned is a consequence of full observability.

Under partial observability, it will generally not be possible to construct an observer function which is isomorphic to the transition function of the STA. Nevertheless, it is often true that the uncertainty in the state of the system is not total and an observer can still identify a subset of the state space where the system will certainly be. This observer will, in fact, still be deterministic but instead of identifying states it will identify subsets of the state space.

We will address the questions posed previously under the assumption that we still have a deterministic observer that can be represented by a FSA. The only information the observer is assumed to have about the system is the logic transition structure.

5.1 Observer Model

5.1.1 Unobservable Events

We start by assuming the existence of unobservable events with $E_{uo} \neq \emptyset$, dropping one of the assumptions of Theorem 4.2.1. Nevertheless, the STA transition function is still assumed to be deterministic, with the uncertainty stemming from the event firing distributions. An observer is still a function:

$$O : P(\mathcal{L}(G)) \to \hat{X}$$

and the difference to the full observability case is that it is not possible to find a function O that induces a bijection between X and $\hat{X}$.

In fact, since the projection is not the identity as was previously the case, due to the existence of unobservable events, it is possible to have strings $s_1 \neq s_2$ such that $P(s_1) = P(s_2)$, which means they will be mapped by O to the same state estimate. In addition, if we have that $x_1 \neq x_2$, with $x_1 = f(x_0, s_1)$ and $x_2 = f(x_0, s_2)$, then it would not be possible to construct a bijective function such that its inverse reconstructs the system state, with $(b^{-1} \circ O)(P(s)) = x$.

Nevertheless, it is still possible to construct an observer and a backward projection function that, for each element of $\hat{X}$ identifies the subset of X where the system might be. We define a backward state projection induced by an observer:

Definition 5.1.1. *An STA, $\boldsymbol{G}$, with deterministic transition function $f : X \times E^* \to X$ (extended to strings) , a projection $P : \mathcal{L}(G) \to E_O^*$ and an observer $O : P(\mathcal{L}(G)) \to \hat{X}$ induce a* backward state projection*:*

$$b^+ : \hat{X} \to 2^X$$

with $b^+(\hat{x}) = \{z \in X : \exists_{s \in \mathcal{L}(G)} (z = f(x_0, s)) \wedge (O(P(s)) = \hat{x})\}$

Under full observability we have that every element of the backward projection will be a singleton, with $b^+(\hat{x}) = \{b^{-1}(\hat{x})\}$.

5.1.2 Building the Observer Automaton

This kind of observer can be given by a finite state automaton similarly to what was done under full observability. The difference is that the observer automaton will not have the same transition structure as the system and will not be isomorphic to the underlying DFA[1] associated with the transitions of **G** . An algorithm to construct such an automaton is presented in (Cassandras and Lafortune, 2007) and the first step is to transform the transition DFA in the non deterministic FSA that is seen after projection P is applied.

We have that $G_{nd} = (X, E_o \cup \{\varepsilon\}, f_{nd}, \Gamma_{nd}, x_0^{nd})$ with:

- X is the same state space as the original automaton.
- $E_o \cup \{\varepsilon\}$ is the event set which only considers observable events and the special event labeled as the empty string.
- $f_{nd} : X \times E_o \cup \{\varepsilon\} \rightarrow 2^X$ is the non deterministic transition function with:

$$f_{nd}(x,e) = \begin{cases} \{f(x,e)\} & e \in E_o \\ \{y \in X : y = f(x,e) \wedge e \in E_{uo}\} & e = \varepsilon \end{cases}$$

- $\Gamma : X \rightarrow 2^E$ is the set of enabled events for each state but with all the unobservable events replaced with ε.
- x_0^{nd} is the initial state with $x_0^{nd} = \{x_0\}$

The procedure to construct a DFA observer **Obs** $= (\hat{X}, E_o, f_O, \Gamma_O, \hat{x}_0)$ from the NFA automaton that we just described, as presented in (Cassandras and Lafortune, 2007), can be seen in the following algorithm:

Algorithm 5.1.1 Procedure to Build an Observer from a Nondeterministic FSA.

```
Start with $\hat{X} = 2^X \setminus \emptyset$
for all $x \in X$ do
  Define [2]:
  $UR(x) = f_{nd}(x, \varepsilon)$
end for
Define, for a set $B$:
$UR(B) = \bigcup_{x \in B} UR(x)$
Define $\hat{x}_0 = UR(x_0)$
for all $S \in X$ and $e \in E_o$ do
  Define:
  $f_O(S,e) = UR(\{x \in X : \exists_{x_e \in S} x \in f_{nd}(x_e, e)\})$
end for
{The resulting state space of the observer is a subset of $2^X$}
```

With this definition, the observer function will essentially be given by the transition function of the observer automaton extended to strings, with $O(s) = f_O(\hat{x}_0, s)$, and a backwards projection is given by the identity function, with $b^+(\hat{x}) = \hat{x}$, since the states of the observer as constructed by the algorithm are essentially subsets of the state space of

[1] Note that without the stochastic structures that define the firing of events, the STA is a Finite State Automaton.

the original automaton already. Still, other sets could be used as states of the observer, as long as would be possible to construct a function b^+ that preserves the correspondence between states of the observer and subsets of X. Put another way, if for some observable event $e \in E_o$ and observer states $\hat{x}_1, \hat{x}_2 \in \hat{X}$ we have that $\hat{x}_2 = f_O(\hat{x}_1, e)$ then:

$$b^+(\hat{x}_2) = UR(\{x \in X : \exists_{x_e \in b^+(\hat{x}_1)} x \in f_{nd}(x_e, e)\})$$

The obvious drawback to using an observer built with the procedure described in (Cassandras and Lafortune, 2007) is the fact that its usefulness depends on the transition structure of the system automaton. Worst case scenario, after a transition period the observer will be stuck in a state $\hat{x}$ with $b^+(\hat{x}) = X$ and $f_O(\hat{x}, e) = \hat{x}$ for all $e \in E_o$. What this means is that the observer reached an observer state where it is not possible to extract any information about the state in which the system is in.

Nevertheless, it is common that systems have a sparsely defined transition function and, for that reason, there is still useful information in the state of the observer, which can be used by the controller as support for decision. A typical and ideal example is when a given observable event e_R resets the state of the original automaton, with:

$$\exists_{x_R \in X} \forall_{x \in X} f(x, e_R) = x_R$$

In this case, whatever the observer state $\hat{x}$ is, we have that:

$$\begin{aligned} b^+(f_O(\hat{x}, e_R)) &= UR(\{x \in X : \exists_{x_e \in b^+(\hat{x})} x \in f_{nd}(x_e, e_R)\}) \\ &= UR(\{x \in X : x = x_R\}) \\ &= UR(x_R) \\ &= f_{nd}(x_R, \varepsilon) \end{aligned}$$

The occurrence of event e_R grounded the observer to a state which corresponds x_R and all the states that can be reached from x_R through unobservable transitions, in all effects reseting the state of the observer to one with potentially more accurate information, particularly if as worst case scenario $b^+(\hat{x}) = X$[3].

5.1.3 Probabilistic Transition Function

If, in addition to considering unobservable events, we also drop the assumption of a deterministic transition function, it is important to determine how much information we can obtain from the system by using a deterministic observer. So, in this case, the transitions of the automaton are represented by a function:

$$p : X \times E \times X \to [0, 1]$$

[3] In MDPs, the information about the current state that the agent receives at each time step can be though as one of those events. In a way, an MDP in a classical Reinforcement Learning setting where the system outputs a state value can be interpreted, in light of a DES formalism, as a system that has an event set E such that $|E| = |X|$ and there is a bijection $e : X \times E$ between states and events such that the system always outputs $e(x)$ when it reaches state x.

representing the probability of jumping from one state to another, given the occurrence of an event. Since p defines a probability mass function, it is clear that $\sum_{y \in X} p(y|x, e) = 1$ for every state x and event e[4].

In this case, even if all events are observable, it is still not possible for an observer function O to distinguish between two different states of the system, since they are possible to be reached using the same trace s, as seen in the proof of Theorem 4.2.1. Nevertheless, it is still possible to obtain a deterministic observer for the STA by looking at the points where p is defined and not zero, the support of the function. We can construct a NFA from the PA, obtaining a nondeterministic transition function, $f_{nd} : X \times E \to 2^X$, from the probabilistic one, with:

$$f_{nd}(x, e) = \{y \in X : (p(y|x, e) \text{ is defined}) \wedge (p(y|x, e) > 0)\}$$

After that, we simply replace all the unobservable events with ε and obtain a nondeterministic FSA that can be used to construct the observer using the algorithm described previously. The practical aspect behind building such an observer for an STA with an underlying probabilistic automaton for transition structure is related to the fact that the probabilities $p(x'|x, e)$ are often not known by the agent but there is still knowledge about the states to which the system might jump to by effect of a given event.

Again, this procedure is susceptible to an decrease of information in the state of the observer and, worst case scenario, to the observer being locked in a state $\hat{x}$ with $b^+(\hat{x}) = X$. In this case, since the nondeterminism of the system increases the chances of the observer not being able to extract any useful information from the strings of events it observes, it is even more important that the function $p(x'|x, e)$ has support over a small set; if we represent it in matrix form P_e, for some ordering of X, then for each event e the matrix will be ideally as sparse as possible. In the case a reseting event e_R exists, leading the system from any state to state x_R, then P_{e_R} will be zero everywhere except for a column of ones corresponding to x_R.

5.2 Properties of the Observer States

To be able to apply some reinforcement learning algorithm to the signal provided by the observer state, we need to verify wether the parameters associated with such a state are Markovian and stationary. Since the observer is, by definition, deterministic, the transition probabilities are intrinsically defined by f_O.

However, there is no guarantee that even events that in the system occur immediately, will manifest themselves immediately from the point of view of the observer. As an example, take a string $s = e_1e_2e_3$ with $e_1, e_3 \in E_O$ and $e_3 \in E_U$. Also, let us assume that τ time passed between the occurrence of event $e1$ and the occurrence of event e_2. From the point of view of the system the generated events and time between them evolves in the following way:

$$e_1 \xrightarrow{\tau} e_2 \xrightarrow{0} e_3$$

[4] In reality, p might not be defined for every *(state, event)* pair but only for those in which $e \in \Gamma(x)$.

On the other hand, after applying projection P, and from the point of view of the observer, we have:

$$e_1 \xrightarrow{\tau} e_3$$

In practical terms, the observer view event e_3 has having a time distribution associated, with $\hat{F}_{e_3}(t) = F_{e_2}(t)$, where $\hat{F}_e(t)$ is a distribution associated with the random variable that represents the time between the occurrence of any event and event e, as seen by the observer. Naturally, if e_3 always occurs after an observable event, then the perceived distribution will be the Heaviside function, as is the case for immediate events.

So it becomes obvious that we need to try to determine these and associated parameters, or a way to calculate them, for the observer states, in order to be able to apply the optimality equations derived in Chapter 4. Let us start by taking a look inside one of the observer states. By definition, a state of the observer $\hat{x}$ corresponds to a subset of the the system automaton, by the way of function b^+. So, we can always say that $b^+(\hat{X}) \subset 2^X$ but for simplicity of notation, and since the map b^+ is by definition injective, we will simply write $\hat{X} \in\subset 2^X$, although strictly this is not necessarily the case.

5.2.1 Semi-Markov Chains

To help determining the probability measures associated with the occurrence of events, we will make use of some results on *Semi-Markov Chains* (SMC), as explained in (Lopez *et al.*, 2001) and in greater detail in (Kulkarni, 1995). A SMC can be described as a tuple $M = (X, P, Q)$ where:

- X is a finite set of states.
- $p : X \times X \to [0, 1]$ is a transition probability matrix such that $\sum_{x' \in X} p(x'|x) = 1$ for every $x \in X$.
- $Q : X \times X \times (\mathbb{R}_0^+ \to [0, 1])$ is a matrix of continuous probability distributions, with $Q(x', x, t)$ representing the probability of moving from state x to x' in less than t time, i.e., $Q(x, x', t) = P[Y < t | X = x, X' = x']$. Y is a random variable representing the holding time between state x and state x'.

The transition structure of an SMC is identical to that of a Discrete Time Markov Chain (DTMC), and (X, P) is often referred to as the *embedded* DTMC of M.

The state holding time is of some state x can be given by a distribution F such that:

$$F(t|x) = P[Y < t | X = x] = \sum_{x' \in X} p(x'|x)\, Q(x, x', t) \tag{5.1}$$

essentially taking the expected value of Q over all the possible successor states.

It is common to impose restrictions on the time it takes for a system to change state, which are essentially the same as Assumptions 2.2.1 and 4.4.1. We say that:

Assumption 5.2.1. *There exist $\epsilon > 0$ and $\delta > 0$ such that*

$$F(\delta|x) < 1 - \epsilon$$

$\forall_{x \in X}$.

Additionally, we assume the expected value of the holding time at each state is finite, $E[F(t|s)] < \infty$.

A sequence of states:

$$x_0 \xrightarrow{t_0} x_1 \xrightarrow{t_1} x_2 \xrightarrow{t_2} \cdots$$

is called a *path* in M if all the consecutive probabilities are non null, i.e, $p(x_{i+1}|x_i) > 0$. The set of all paths in the SMC can be denoted by $Path^M$ with $Path^M(x)$ referring to all the paths that start in state x. For a given path σ, we can denote the state occupied at time t by $\sigma@t$.

Some important measures on SMCs can be defined.

$$\begin{aligned} \pi(x, x', t) &= P\{\sigma \in Path(x) \ \mid \ \sigma@t = x'\} \\ &= P[X' = x'|T = t, X = x] \end{aligned} \tag{5.2}$$

denotes the probability of the state being $x' \in X$ at time t if the chain started in state x. This has a similar meaning to $\pi(t)$ in a Continuous Time Markov Chain (CTMC) except that the initial state is specified.

Another important measure is related to the first time a state x' is reached after starting in state x and within t time units:

$$\begin{aligned} H(x', t|x) &= P\{\sigma \in Path(x) \ \mid \ \exists_{t' \in]0,t]} \ \sigma@t' = x'\} \\ &= P[X(t) = x', Y < t|X(0) = x] \end{aligned} \tag{5.3}$$

This measure satisfies the following system of equations:

$$H(x', t|x) = p(x'|x)\, Q(x, x', t) + \sum_{x'' \neq x'} \int_0^\infty p(x'|x) \frac{dQ(x, x'', \tau)}{d\tau}\, H(x'', t - \tau|x') d\tau \tag{5.4}$$

which can be proved to have a unique solution if the state holding times are positive with nonzero probability, as guaranteed by Assumption 5.2.1.

5.2.2 Observer States as Semi-Markov Chains

Let us now consider, for each state $\hat{x}$ of the observer, a SMC $M_{\hat{x}} = (\hat{x}, p, Q)$[5]. If we focus on the dynamics of the SMC without the possibility of escaping it, i.e, of the occurrence

[5] Although we generally denote sets in upper case, here we maintain the notation $\hat{x}$ to emphasize that the state space of the SMC is one state of the observer. Strictly speaking, $\hat{x}$ is not necessarily a set of states but can be associated with a set of the system states through the function b^+, as previously defined, and so the abuse of notation is reasonable for the sake of simplicity of expressions.

of observable events that, by definition, will change the state of the observer:

$$\begin{aligned} p(x'|x) &= P[X' = x'|X = x] \\ &= \frac{1}{\eta(x)} \sum_{e \in \Gamma(x) \cap E_{UO}} P[X' = x|X = x, E = e] \; P[E = e|X = x] \\ &= \frac{1}{\eta(x)} \sum_{e \in \Gamma(x) \cap E_{UO}} p(x'|x, e) \, p(e|x) \end{aligned} \tag{5.5}$$

with $p(x'|x, e)$ and $p(e|x)$ being given by the STA parameters as described in Chapter 4. The normalizing factor $\eta(x)$ is needed because the probabilities of events occurring in each state are defined for all the events and not just the unobservable ones. We have:

$$\eta(x) = \sum_{e \in \Gamma(x) \cap E_{UO}} p(e|x)$$

If we make another assumption:

Assumption 5.2.2. *The control and supervisor actions only change after the occurrence of observable events.*

It is reasonable to assume so since from the point of view of the supervisor and observer there are no changes in the system. We can now redefine the SMC to include the influence of the supervisor and observer actions, with $M_{\hat{x}}^{aS} = (\hat{x}, p_{aS}, Q_{aS})$, which leads to:

$$\begin{aligned} p_{aS}(x'|x) &= P[X' = x'|X = x, A = a, S] \\ &= \frac{1}{\eta(x)} \sum_{e \in \Gamma_S(x) \cap E_{UO}} P[X' = x|X = x, E = e] \; P[E = e|X = x, A = a, S] \\ &= \frac{1}{\eta(x)} \sum_{e \in \Gamma_S(x) \cap E_{UO}} p(x'|x, e) \, p(e|x, S, a) \end{aligned} \tag{5.6}$$

with $p(e|x, S, a)$ being given directly by Equations (4.3) and (4.4). Additionally:

$$\eta(x) = \sum_{e \in \Gamma_S(x) \cap E_{UO}} p(e|x, S, a)$$

On the other hand, the function Q_{aS} is given by:

$$\begin{aligned} Q_{aS}(x', x, t) &= P[Y < t|X' = x', X = x, A = a, S] \\ &= \frac{P[Y < t, X' = x'|X = x, A = a, S]}{P[X' = x'|X = x, A = a, S]} \\ &= \frac{\frac{1}{\eta} \sum_{e \in \Gamma_S(x) \cap E_{UO}} P[Y < t, X' = x'|X = x, E = e] \; P[E = e|X = x, A = a, S]}{\frac{1}{\eta} \sum_{e \in \Gamma_S(x) \cap E_{UO}} P[X' = x'|X = x, E = e] \; P[E = e|X = x, A = a, S]} \\ &= \frac{\sum_{e \in \Gamma_S(x) \cap E_{UO}} F_e(t) \, p(x'|x, e) \, p(e|x, S, a)}{\sum_{e \in \Gamma_S(x) \cap E_{UO}} p(x'|x, e) \, p(e|x, S, a)} \end{aligned} \tag{5.7}$$

with $H(t, x'|x, e)$ being the same as in Equation (4.6).

Now let us extend the SMC with some states corresponding to the occurrence of observable events. Let us define $X_E = \{x_e\}_{e \in E_O}$ that essentially creates one additional state for each $e \in E_O$. The purpose of these states is to study the conditions under which the original SMC corresponding to a given observer state $\hat{x}$, as defined previously, is abandoned and the idea is to make the chain jump to an absorbing state in X_E every time an observable event occurs.

So now we have $\tilde{M}_{\hat{x}}^{aS} = (\hat{x} \cup X_E, \tilde{p}_{aS}, \tilde{Q}_{aS})$, and the chain parameters are defined as follows.

- For $x, x' \in \hat{x}$[6], the transition probabilities are defined as Equation (5.6) but considering all the active events in x. So, we have:

$$\tilde{p}_{aS}(x'|x) = \sum_{e \in \Gamma_S(x)} p(x'|x, e)\, p(e|x, S, a) \tag{5.8}$$

  Additionally, we assume that $p(x'|x, e) = 0$ if e is an observable event.

- For $x \in \hat{x}$ and $x' \in X_E$ the system will change from any state x to x' every time there is an occurrence of the corresponding observable event $e(x') \in E_O$. So we have that:

$$\tilde{p}_{aS}(x'|x) = p(e(x')|x, S, a) \tag{5.9}$$

  which stems from the fact that

$$p(x'|x, e) = \begin{cases} 1 & e = e(x') \\ 0 & otherwise \end{cases}$$

- Finally, since we are only interested in studying the first time the markov chain reaches a state in X_E, corresponding to the occurrence of an observable event and abandoning the original SMC $M_{\hat{x}}^{aS}$, we defined that each state in X_E is an absorbing state, i.e., if $x \in X_E$ then $\tilde{p}_{aS}(x|x) = 1$.

As for the Q matrix, we have:

$$\tilde{Q}_{aS}(x', x, t) = \frac{\sum_{e \in \Gamma_S(x)} F_e(t)\, p(x'|x, e)\, p(e|x, S, a)}{\sum_{e \in \Gamma_S(x)} p(x'|x, e)\, p(e|x, S, a)}$$

Particularly this expression also defines $\tilde{Q}_{aS}(t|x, x')$ when $x \in X_E$ but its value is irrelevant for our study here.

[6] Note that $\hat{x} \cap X_E = \emptyset$.

Now, if we go back to Equation (5.4) and apply it for some initial state $x_0 \in \hat{x}$ and final state $x_e \in X_E$ we have:

$$\begin{aligned} &H_{aS}(x_e, t|x_0) = \\ &\quad \tilde{p}_{aS}(x_e|x_0)\, \tilde{Q}_{aS}(x_0, x_e, t) + \sum_{x'' \neq x_e} \int_0^\infty \tilde{p}_{aS}(x_e|x_0) \frac{d\tilde{Q}_{aS}(x_0, x'', \tau)}{d\tau}\, H_{aS}(x'', t-\tau|x_e) d\tau \end{aligned} \tag{5.10}$$

and we know there is a solution to this system of equations (Kulkarni, 1995). Since $H_{aS}(x_e, t|x_0)$ represents the probability of reaching state x_e for the first time, and since we defined x_e as being reached every time the event e occurred, we can write that:

$$H_{aS}(x_e, t|x_0) = P[Y < t, X_t = x_e | X_0 = x_0, a, S]$$

By taking the marginal over t we can obtain the probability of ever reaching state x_e if we start in state x_0.

$$p_\infty(x_e|x_0, a, S) = \int_0^\infty H_{aS}(x_e, dt|x_0)$$

This probability is not necessarily 1 if nothing else because we assumed every state in X_E to be an absorbing state, so there might be a chance that some state is reached before x_e, i.e., there might a chance another observable event occurs before e. Since reaching x_e does correspond to the occurrence of event e in $\hat{x}$, we can write that:

$$p(e|\hat{x} \leftarrow x_0, a, S) = \int_0^\infty H_{aS}(x_e, dt|x_0) \tag{5.11}$$

On the other hand:

$$\begin{aligned} P[Y < t, X_t = x_e | X_0 = x_0, a, S] &= P[Y < t, E = e | X_0 = x_0, a, S] \\ &= P[Y < t | E = e, X_0 = x_0, a, S]\; P[E = e | X_0 = x_0, a, S] \end{aligned}$$

And so we can write that:

$$H_{aS}(x_e, t|x_0) = F_e(t|\hat{x} \leftarrow x_0)\, p(e|\hat{x} \leftarrow x_0, a, S)$$

or

$$F_e(t|\hat{x} \leftarrow x_0, a, S) = \frac{H_{aS}(x_e, t|x_0)}{\int_0^\infty H_{aS}(x_e, dt|x_0)} \tag{5.12}$$

We are now in the condition to prove the following result:

Theorem 5.2.1. *Suppose we have a C-GSTA with a set of observable events E_O. Let* **Obs** $= (\hat{X}, E_o, f_O, \Gamma_O, \hat{x}_0)$ *be the observer automaton constructed using the algorithm described previously. The states of the observer are a Markovian signal that can be used for learning if each state $\hat{x}$ is reached always under the same initial condition $x_0(\hat{x})$.*

Proof. Going back to Equations (5.12) and (5.11), if the initial state of $\hat{x}$ is always $x_0(\hat{x})$, then we can write:

$$p(e|\hat{x} \leftarrow x_0(\hat{x}), a, S) = \int_0^\infty H_{aS}(x_e, dt|x_0(\hat{x}))$$

$$F_e(t|\hat{x} \leftarrow x_0(\hat{x}), a, S) = \frac{H_{aS}(x_e, t|x_0(\hat{x}))}{\int_0^\infty H_{aS}(x_e, dt|x_0(\hat{x}))}$$

But since x_0 is fixed for every $\hat{x}$, the quantities $p(e|\hat{x}, a, S)$ and $F_e(t|\hat{x}, a, S)$ will only be dependent on the observer state and supervisor and controller actions. We can also extend the state representation to include both the state of the observer and the supervisor, and in this case the supervisor function is also dependent on the state.

So essentially, we have the same parameters defined in Chapter 4 with no dependence on the past history of the observer but only its current state – the event path which lead to a given state of the observer $\hat{x}$ is irrelevant since we are assuming the state will always be reached in the same way, i.e. when the observer enters $\hat{x}$ the system will always enter the same $x_0 \in \hat{x}$. For this reason, the state of the observer is a Markovian signal and the optimality equations from Chapter 4 also apply.

Additionally, if we maintain the Assumptions 4.4.1 and 4.4.2, we can still guarantee the existence of a fixed point of the Bellman operator, applied to the observer state, and the same Q-learning update rule can be used.

□

It is important to note that, although we shown how to obtain the expressions for $p(e|\hat{x}, a, S)$ and $F_e(t|\hat{x}, a, S)$, if a learning algorithm is used these quantities will be embedded in the $Q-values$ and there will be no need to calculate them analytically, which might prove difficult specially solving the system of equations given by Equation (5.4) or (5.10). This is one of the advantages of using a stochastic approximation strategy like Q-learning, and its well patent in this particular application.

Also, the restriction of making every sub-SMC associated with a given state of the observer might be lifted if the initial condition is not always the same but the controller has information about what such initial condition is. In this case, the state of the observer used by the learner must be given by $(\hat{x}, x_0)$, with $x_0 \in \hat{x}$. Put another way, by extending the state of the learner to the initial condition of each observer state, it is possible to differentiate from the several emission probabilities and cumulative distribution functions (cdfs) $p(e|\hat{x} \leftarrow x_0(\hat{x}), a, S)$ and $F_e(t|\hat{x} \leftarrow x_0(\hat{x}), a, S)$ respectively. Since each state of the observer is associated with a finite number of system states, there will also be a finite number of initial conditions. An upper bound for the number of states of the learner is $|X^S||\hat{X}||X|$ but since each $\hat{x}$ does not contain all the possible states in X, the number of possible initial conditions will be slightly lower, and the overall number of states that the learner will experience will be lower as well. We are not considering the situations where the initial state of each sub-SMC might be a distribution over $x \in \hat{x}$.

5.3 Controller Model

Under the conditions expressed in Theorem 5.2.1, it is possible to write Markovian transition and event emission parameters for each observer state so that it can be used in a Bellman equation and, ultimately, by the reinforcement learning based controller described in Chapter 4. Nevertheless, there are three differences that need to be considered when adapting the algorithm to a setup with this kind of observer:

- The cdfs associated with the inter-event time are not only associated with events but with the observer state and controller/supervisor actions being considered. In fact, as explained previously, even though the occurrence of a given event is still governed, at the system level, by a fixed cdf, the crucial quantity when applying the optimality equations to the observer is the *perceived* cdf, from the point of view of the observer, i.e., $F_e(t|\hat{x}, a, S)$.
- If the rewards are dependent on the system state and not the observer state, it might happen that the controller receives lump rewards between event occurrences, or put another way, in an instant where it is not expecting to receive any reward. Additionally, the rate reward term, which depends linearly on time, might change due to an observable event which, from the point of view of the observer, is as if the reward changed with no apparent reason since it does not have access to state changes produced by unobservable events.
- The action set cannot depend directly on the system state but needs to be a function of the observer and supervisor states.

The first difference of the case being explored in this chapter is easily addressed by modifying the optimality equations and associated quantities. As for the reward problem, we introduce the following assumption:

Assumption 5.3.1. *The rewards are assumed to be obtained by a process posterior to the observation and, for this reason, are subject to the same constraints in terms of unobservable events as the observer. Put another way, they are a function of the observer state.*

In fact, we had already considered such an assumption in Chapter 4 but since there was a bijective correspondence between states of the observer and states of the system there was no visible effect of assuming the rewards were dependent on the state of the observer and not the state of the system.

Finally, we will re-define the action set as:

$$A_{\hat{x}} = \begin{cases} \Gamma_S(\hat{x}) \cap E_A & \Gamma_S(\hat{x}) \setminus E_A = \emptyset \\ (\Gamma_S(\hat{x}) \cap E_A) \cup \{\varepsilon_a\} & \textit{otherwise} \end{cases}$$

with $\Gamma_S(\hat{x}) = \Gamma_O(\hat{x}) \cap S(x^S)$, where we explicitly refer to x^S in the supervisor function. Additionally, $\Gamma_O(\hat{x})$ only depends on the observer state but, for simplicity of notation in the Bellman equations, we maintain the extended state assumption.

Now, we can re-write the bellman optimality equations in the following way:[7]

$$V^{S,*}(\hat{x}) = \max_{a \in A_{\hat{x}}} \left[\sum_{e \in \Gamma_S(\hat{x})} p(e|\hat{x}, S, a) \left[r(\hat{x}, e) + \gamma_e(\hat{x}, S, a) \sum_{y \in \hat{X}} p(y|e, \hat{x}) V^{S,*}(y) \right] \right] \tag{5.13}$$

where the event dependent discount factors now also depend on the action and state, with:

$$\gamma_e(\hat{x}, S, a) = \int_0^\infty e^{-\beta\tau} F_e(d\tau|\hat{x}, S, a)$$

Like mentioned previously, this differences are mostly relevant to writing the equations and showing how the observer state still can be used for learning, leading to the optimal control solution given the information that the observer has available. However, if the controller is implemented by a Q-learning algorithm like in Chapter 4 we can use the same update rule from Equation (4.26), given by:

$$Q^S_{k+1}(\hat{x}_k, a_k) = (1 - \alpha_k) Q^S_k(\hat{x}_k, a_k) + \alpha_k \tilde{Q}^S_{k+1}(\hat{x}_k, a_k)$$

with

$$\tilde{Q}^S_{k+1}(\hat{x}_k, a_k) = r_k + e^{-\beta\tau_k} \max_{b \in A_{\hat{x}_{k+1}}} Q^S_k(\hat{x}_{k+1}, b)$$

5.4 A Practical Case

We used the same domain as in the full observability chapter but considered the supervisor, observer and controller were not able to observer the events of type `moveDirection`, only being able to determine when the robot reached a corner/intersection or when it bumped into a wall. All the interaction events were considered to be observable, and the observer automata representing robot-person interaction did not change.

We did not use the navigation supervisor in the partial observability tests because it requires a perfect knowledge of the map position of the robot. Since the states of the map observer are subsets of the map positions, it would still be possible to do some supervision, if for some state $x_{obs} \in X_{obs}$ of the observer, all corresponding states[8] of the original supervisor shared a non-active event:

$$\exists_{e \in E_C} \forall_{x \in x_{obs}} e \notin \Gamma(x)$$

but the amount of those states would be minimal and there would not be a big gain in using a navigation supervisor.

In Figure 5.1 a detail of the of the navigation observer under partial observability is shown.

We did 2 additional tests comparing the performance of the supervised system partial observability, both with no supervisor and with the interaction supervisor[9].

[7] We are still considering the state of the observer augmented by the state of the supervisor, as was done in Chapter 4. So, in reality $\hat{x}$ actually refers to the pair $(\hat{x}, x^S)$.

[8] These states do not reflect the complete state of the environment, observer or supervisor, but rather only of the corresponding map automata at each level.

[9] The single-action supervisor was still used in all situations.

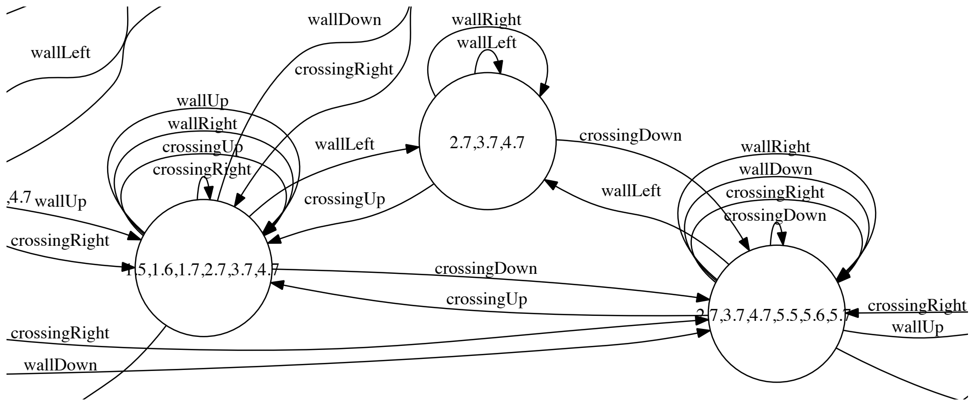


Figure 5.1: Detail of the map observer under partial observability.

5.5 Results

For the partial observability case, we obtained an evolution of total accumulated rewards over a period of T=50000 that can be seen in Figure 5.2. At first sight, it seems surprising that under partial observability the system seems to converge faster than under full observability. This can, in fact, be explained by the nature of the environment: since the robot lives in the corridors of a building, there is often no reason to turn back while navigating a corridor, and the fact that under partial observability there is no way to observe some of the intermediate state transitions, makes the robot always travel full corridors. Nevertheless, for the particular states where it might be better to turn back, usually after unexpected events like meeting a person in the middle of the corridor or finishing a task, the fully observable system is able to distinguish between states and, thus, learn exactly when to turn back. We can see that, for the time period considered, the fully observable system under interaction supervision in fact does converge eventually to a higher reward rate than the partially observable one.

In Table 5.1 the statistics under partial observability are shown. We can see that with the Interaction supervisor, the percentage of successful tasks was slightly bigger under full observability than under partial observability (60.6% vs. 59.2%), that difference being likely to increase if more time was considered. The % of times the robot abandoned the task or the person got tired of waiting are all very similar to the fully observable case, the first one because it mainly depends on the exploration parameter and the amount of actions available and the second one depends on the probability of doing actions outside the correct action sequence, and that is also independent of observing the map state.

In this case we did not impose the condition of using the same initial condition for each state of the observer. Nevertheless, we can see the system behaved well, although this is dependent on the particular application at hand.

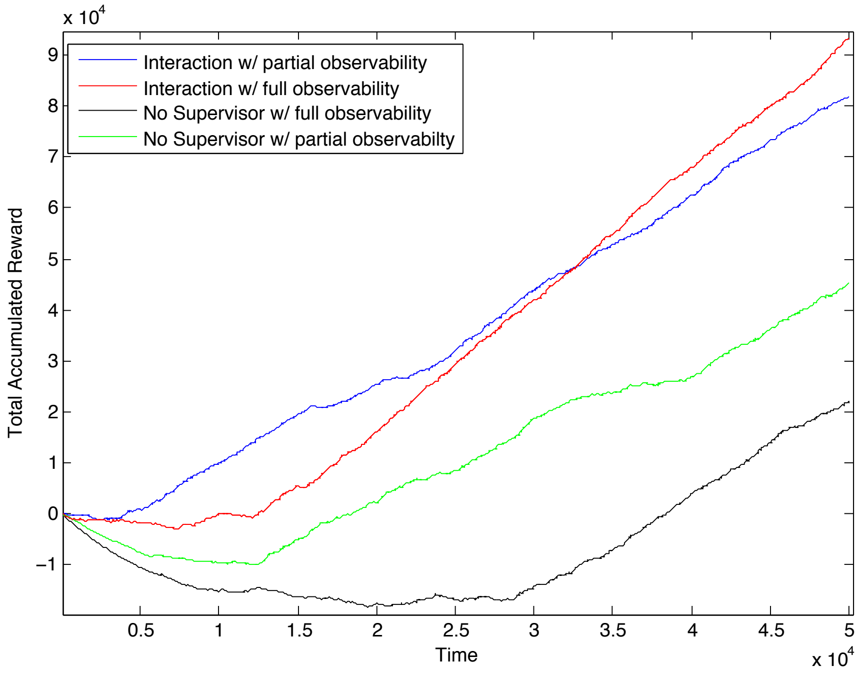


Figure 5.2: Total accumulated rewards over time under partial and full observability for different supervisory schemes, for a test with T=50000.

Supervisors Used	**# interactions**	**% leads to goal**	**% person gave up**	**% robot gave up**	**% help declined**
Interaction	2182	59.2%	8.2%	0%	32.6%
None	2400	41.0%	21.4%	13.1%	24.5%

Table 5.1: Task statistics under partial observability for a test with T=50000.

5.6 Summary

In this chapter we studied the effects on our framework of dropping the observability assumptions from Chapter 4. Under partial observability, it is still possible to construct a deterministic observer which captures the logical aspects of the system. In general, this observer is less informative than the one under full observability but it still provides enough information to support decision making, particularly since it only requires knowledge about the logic structure of the transitions of the underlying system to be built, and is not dependent on the knowledge about its parameters. The observer works by identifying states in a disjunctive way, i.e., each state of the observer identifies a subset of the state of the system, which is essentially as saying, e.g., ”The system is either in state x, y or z”.

With this model of the observer, each observer state will not only be associated with a subset of the system but also with the dynamics within that subset. For this reason, each state of the observer can be identified with a Semi-Markov sub-Chain, similar to the ones defined in (Lopez *et al.*, 2001), of the STA. We proved that, if the initial conditions of this subchain are always the same, then the evolution of the states of the observer is also a Semi-Markov Process, as under full observability in Chapter 4, and the optimality equations and learning rule used in Chapter 4 can still be applied to the observer. Nevertheless, there are some minor differences in the optimality equations that need to be taken into account which we discussed, although they do not affect the update rule.

Finally, we picked up the case study from Chapter 4, re-did the experiments under partial observability and with the construction of a deterministic observer as described in this chapter, and compared the results to the ones under full observability. We saw that incomplete knowledge about the state of the system, in this particular case, actually led to better short term results since it eliminated some exploring options, but the long term behavior was, as expected, not as good.

Chapter 6

Partial Observability: Probabilistic Observer

In Chapter 5 we looked at the question of partial observability by assuming we could construct an observer that retained the deterministic characteristics of the system, in the presence of uncertainty in observation both from the existence of unobservable events and the lack of knowledge about the effects of the events. In fact such approach is often enough for a vast number of problems, particularly if they exhibit sparse transition functions and identifiable states. Additionally, we saw how the observer models used required little information about the parameters of the underlying system.

But what if the observer has access to additional information? It should be possible to construct policies that are more adequate to deal with partial observability by including that additional information in the observer model. Since the parameters of the system are probabilistic quantities, it seems natural to adopt as observers models that are able to encode such probability information.

In this chapter we address the question of considering probabilistic observer models and how that affects the supervisor and the optimality equations for the controller.

6.1 Observer Model

We start by noting that the transition structure of the STA that models the system is given by a *probabilistic automaton*, if the full observability assumptions are not considered. A probabilistic automaton (PA) can be defined as (X, E, Γ, p, p_0) where:

- X represents a state space.
- E represents an event set.
- $\Gamma : X \to 2^E$ is the set of enabled events in state x
- $p : X \times E \times X \to [0, 1]$ is a transition function defined as:

$$p(x'|x, e) = P\left[X_{k+1} = x'|X_k = x, E_k = e\right]$$

  The function is possibly partial, not defined for events $e \notin \Gamma(x)$. The indices $k, k+1, \ldots$ are associated with the moments in time where some event occurs.

- $p_0(x)$ is a probability distribution over X that represents the knowledge about the initial state of the system, $P\left[X_0 = x\right]$.

The name probabilistic automaton is sometimes used to represented one of the most popular kinds of these systems called *Rabin Automata* (Rabin, 1963).

This automaton can be used as a recognizer for the language generated by the system taking the event strings as inputs. The state of the observer automaton at each time step is characterized by state probabilities, much like in *Discrete Time Markov Chains*, such that:

$$v_j(k) = P[X_k = x_j]$$

for some ordering of state space X. Since the we can always build a bijective index function according to the ordering of X, it is sometimes common to write $P[X_k = j]$ even though in reality the state space is not a subset of $\mathbb{N}$.

In fact, if we consider matrices P_e according to the referred ordering of X such that:

$$P_e(i,j) = p(x_j|x_i, e)$$

and

$$P_e(i,j) = 0 \quad \text{if} \quad j \neq i, e \notin \Gamma(x_i)$$

the distributed state of the probabilistic automaton, given an input string, e.g., $e_1e_2e_3$, can be calculated as:

$$v = p_0 \, P_{e_1} \, P_{e_2} \, P_{e_3}$$

with the difference to DTMCs being that the probability matrices depend on the input string.

This kind of automaton is specially adequate when we want the transitions to depend on input events. The idea is that the automaton state will change according to whatever dynamics are associated with the firing of events but this mechanism does not need to be known by the automaton itself. However, since we intend to use it as an observer for the state of the system, there is information not encoded in the automaton definition and that could be used to more precisely provide a probabilistic estimation of the system state.

In fact, since the automaton does not know the exact state of the system, the following Bayesian expression is more accurate in expressing the update of the probability vector v:

$$\begin{aligned} v_{k+1}(x') &= P[X_{k+1} = x'|E_k = e, V_k = v_k] \\ &= \sum_{x \in X} P\left[X_{k+1} = x'|E_k = e, X_k = x, V_k = v_k\right]\; P\left[X_k = x|E_k = e, V_k = v_k\right] \\ &= \sum_{x \in X} P\left[X_{k+1} = x'|E_k = e, X_k = x\right]\; \frac{P\left[E_k = e|X_k = x, V_k = v_k\right]\; P\left[X_k = x|V_k = v_k\right]}{P\left[E_k = e|V_k = v_k\right]} \\ &= \frac{1}{\eta} \sum_{x \in X} p(x'|x, e)\; P\left[E_k = e|X_k = x\right]\; v_k(x) \end{aligned} \tag{6.1}$$

where η is a normalization constant.

The expression bears a strong resemblance to the POMDP update rule of Equation (2.21), with the difference residing in the fact that not only events work in a similar way to both actions and observations but the emission of events, corresponding to $P\left[E_k = e|X_k = x\right]$, depends on the previous state and not the next one, as opposed to observations in POMDPs . We can see how events have a double aspect of changing the state of the system, like actions, and providing information about the state of the system, like observations. Conversely, we could think of actions and observations as kinds of events.

In any case, the probability vector essentially represents the same concept as the notion of *belief* and, for this reason, we will start using b to denote it and b_0 to denote the initial probability vector of the system.

6.1.1 Probabilistic Discrete Event System

To better model all the quantities that the observer needs to have access to we will use, instead of a PA, a model that in (Lawford and Wonham, 1993; Pantelic *et al.*, 2009) is known as a *Probabilistic Discrete Event System* (PDES), which can be described as a tuple $(X, E, \Gamma, p, b_0, p_E)$ with:

- X, E, Γ, p, b_0 having the same meaning as for Probabilistic Automata.
- $p_E : X \times E \to [0, 1]$ with $p_E(e|x) = P[E = e|X = x]$ representing the probability of event e firing in state x.

This kind of model is similar to GSTAs but time is not considered explicitly and all the event emission probabilities are given directly by probability distributions over events. In a way, it is as if all the states of a GSTA corresponded to random switches but with no consideration about the time transitions take. Since we intend to use the PDES as an observer to the system, it is important to note that it does not generate a string of events, although it has information about the probabilities of generating events, but rather it receives a string of events and outputs a probability vector, or belief.

The belief update for this system is, then, done using the following rule based on Equation (6.1)

$$b^e(x') = \frac{1}{\eta} \sum_{x \in X} p(x'|x,e)\, p_E(e|x)\, b(x) \tag{6.2}$$

with $\eta = p(e|b) = \sum_{x \in X} p_E(e|x)\, b(x)$. The update is similar to Equation (2.21).

So far we have not considered the time between states x and x' to be an observational input. If we do consider that besides the events the observer also has access to the time the transition took, then the update expression must be modified to:

$$b^{e\tau}(x') = \frac{1}{\eta} \sum_{x \in X} p(x'|x,e)\, p_E(e|x) f(\tau|x,e)\, b(x) \tag{6.3}$$

with $\eta = \left(\sum_{x \in X} p_E(e|x)\, b(x)\right) \left(\sum_{x \in X} f(\tau|x,e)\, b(x)\right)$ and $f(t|x,e) = \frac{dF(t|x,e)}{dt}$. However, since we assumed that the time distribution associated with each event only depended on the event itself, then:

$$f(\tau|x,e) = f_e(\tau) = \frac{dF_e(t)}{dt}|_{t=tau}$$

and

$$\sum_{x \in X} f(\tau|x,e)\, b(x) = f_e(\tau)$$

which reduces the update to Equation (6.2). Essentially, if the time cdf associated with each event only depends on the event itself, knowing the sampled time the event took to occur, τ, does not provide any additional information about the state of the system, considering the event itself is known.

6.1.2 The Observer as a Probabilistic Discrete Event System

The observer is a function:

$$O : P(\mathcal{L}(G)) \to \Delta(\hat{X})$$

with $\Delta(\hat{X})$ representing the simplex of beliefs over $\hat{X}$, i.e., the space of probability distributions on $\hat{X}$ – for simplicity of notation we will refer to it as Δ, similarly to what was done for POMDPs. Note the observer progression we analyzed so far:

Deterministic Full Observer: in Chapter 4 the observer had as output a value that could be bijectively identified with the state of the original system.

Deterministic Partial Observer: in Chapter 5 the observer had as output a rough idea of what the state of the system could be, with observer states corresponding to subsets of the state space of the system.

Probabilistic Partial Observer: in this chapter the observer does not have a deterministic state but a probabilistic one. Essentially, the idea is that here the state space of the observer is isomorphic to the state space of the system, as in Chapter 4, but the observer does not pick a state directly and instead maintains a distributed representation of state.

The observer can be implemented by a PDES defined as:

$$\mathbf{Obs} = (\hat{X}, E_o, \Gamma_O, \hat{p}, b_0, p_E)$$

with $\hat{X} = b(X)$ for some bijection between both state spaces, E_o is the set of observable events, $\Gamma_O(\hat{x}) = \Gamma(b^{-1}(\hat{x}))$ and $\hat{p} : \hat{X} \times E \times \hat{X} \to [0, 1]$ is a function such that, for $\hat{x}' = b(x')$ and $\hat{x} = b(x)$ we have:

$$\hat{p}(\hat{x}'|\hat{x}, e) = p(x'|x, e)$$

For now we assume the effects of events are not deterministic, which breaks one of the observability conditions, but all events are observable. In this case, since X, Γ and p are isomorphic with $\hat{X}$, Γ_O and $\hat{p}$, we will drop the additional notation and refer to the quantities of the observer as the corresponding quantities of the original system, unless explicitly stated. Still to define is the function p_E but we will address that further ahead.

In general, even under full observability as defined in Chapter 4, meaning that it is possible to reconstruct an estimate of the state of the system from the produced string of events, there might be errors in the definition of the observer model. In that case, the system is still fully observable but the observer is not sound, and sets like X and $\hat{X}$ might not be isomorphic. The presence of faulty observers and inaccurate estimations of the system parameters is outside the scope of this work.

Besides the tuple definition, the observer maintains not a representation of the state it is in but a representation of the belief. The belief update is then made using the Bayesian rule presented in Equation (6.2). If we consider the existence of a supervisor and controller of the system, the belief update expression has to be modified to take into account the supervisor and controller actions, that we will assume are also given as an input to the system. In this case, we have:

$$b^{eSa}(x') = \frac{1}{\eta} \sum_{x \in X} p(x'|x, e)\, p_E(e|x, S, a)\, b(x) \tag{6.4}$$

with $\eta = p(e|b, S, a) = \sum_{x \in X} p_E(e|x, S, a)\, b(x)$. If we considered τ to be an observational input, a similar reasoning to what was made previously could be applied, and the modified belief update would reduce to Equation (6.4). If however, as for the observer of Chapter 5, the time cdfs associated with events also depend on the state and action, then the expression of belief update has to be modified to account for the additional observational input provided by τ, similar to what was done for POSMDPs by (Mahadevan, 1998) as described briefly in Chapter 3. In this chapter, we continue assuming independence of $F_e(t)$ from the state and action.

We will again assume, as in Chapter 4, that all actions correspond to a choice of a controllable and immediate event. Additionally, we recall that it is possible to choose an idling action ε_a that represents waiting for some uncontrollable event to happen. With this in mind, the emission probabilities can be obtained by recalling Equations (4.3) and (4.4). The first one

$$p(e|x,S,a) = \begin{cases} p_{rs}(x,e) & e \in \Gamma_S(x,s) \cap (E_U \setminus E_C) \\ 1 - \sum_{i \in (\Gamma_S(x,s) \cap (E_U \setminus E_C))} p_{rs}(x,i) & e = a \wedge e \in \Gamma_S(x,s) \\ 0 & \text{otherwise} \end{cases} \tag{6.5}$$

is used if the state corresponds to a random switch, i.e., there are still active events with associated null time after the effects of the controller and supervisor.

The second one

$$p(e|x,S,\varepsilon_a) = \int_0^\infty F_e(y) \cdot dF_{W_e}(y) \tag{6.6}$$

with $F_{W_e}(t)$ defined by Equation (4.9), is used when the state is not a random switch, or when an idling action was chosen[1].

Essentially, even if the parameters of the system are known, it is clear that the hard part about constructing an observer is integrating the cumulative distribution functions associated with the time between events, since all other parameters are known. For this reason, a common assumption in modeling continuous-time Markov processes is to assume all the inter-event times to be exponentially distributed, i.e., generated by a Poisson process. This not only gives the expressions a closed form, as seen in Chapter 4, but guarantees that the system is fully Markovian. Under a Poisson clock structure we have, for timed events:

$$p(e|x,S,\varepsilon_a) = \frac{\lambda_e}{\Lambda(x,s)}$$

where the rates are the same as defined in Chapter 4.

We refer this approximation because it is a common assumption, particularly in Stochastic Petri Nets (Molloy, 1982), but it is still the goal of this work not to be restrictive in the type of cumulative distribution function (cdf) used for inter-event times.

6.2 Controller Model

The control scheme for the system with a probabilistic observer is slightly different from the general one, in the sense that the observer does not output an exact state but a belief state, as represented in Figure 6.1. The fact that we denote the policy generator

[1] Again, note that we can consider that the only possible choice of action in non-random-switch states is the idling action.

by the general designation of *Controller* is tied to the fact that, by using the belief state as input, the system will be similar to a POMDP and for this reason optimizing it will be equivalent to optimizing a particular kind of infinite MDP, which cannot be accomplished by the basic Q-learning algorithm.

Our goal in this chapter is to obtain equations that can be used to derive POMDP-like dynamic programming algorithms for our discrete event based system, which are generally run offline assuming we know the model parameters, as opposed to Q-learning. When the agent is actually interacting with the environment, it does so through the optimal control policy but during that time the policy is fixed. Contrary to what we did in Chapter 4, we will explicitly consider referring to the complete state, as seen by the controller, as (b, x^S) to distinguish between the different nature of the observer belief, which is continuous, and the supervisor state, which is discrete and finite.

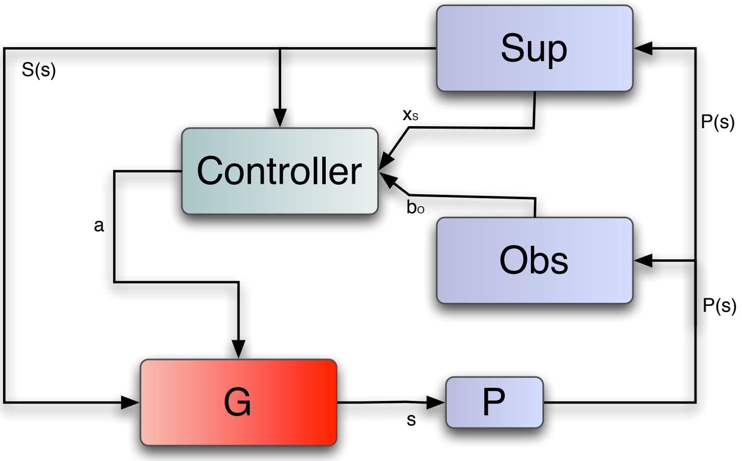


Figure 6.1: Supervised event-based control block diagram for a system with a probabilistic observer.

6.2.1 Optimality Equations

To derive the optimality equations, we follow a similar procedure as what was done for POMDPs. First we start by recalling the value function optimality equation derived in Chapter 4. We have, for a policy π:

$$V^{S,\pi}(x) = \sum_{e \in \Gamma_S(x)} p(e|x,S,\pi) \left[r(x,e) + \gamma_e \sum_{y \in X} p(y|e,x) V^{S,\pi}(y) \right] \tag{6.7}$$

and for the optimal policy:

$$V^{S,*}(x) = \max_{a \in A_x} \left[\sum_{e \in \Gamma_S(x)} p(e|x,S,a) \left[r(x,e) + \gamma_e \sum_{y \in X} p(y|e,x) V^{S,*}(y) \right] \right] \tag{6.8}$$

Since in this chapter we not only do not have an access to the state but it is impossible to provide an exact estimation from the inspection of the output strings, because we dropped one of the conditions of Theorem 4.2.1, we need to rewrite the value function in terms of the quantities that the controllers does have access to: the belief about the state of the system and the state of the supervisor.

From the point of view of the controller and considering a probabilistic observer whose output is a belief vector, it is as if the process is being generated by an automaton $\mathbf{A} = (\Delta \times X^S, E, p, p_E, (b_0, x_0^S))$ from which the controller can inspect the state and the events being produced. The state space is simply a product of the observer belief simplex and supervisor state space, and the event set used is the same the system produces[2]. The transition probabilities are given by:

$$p(b', x^{S'}|b, x^S, e) = p(b'|b, e)\, p(x^{S'}|x^S, e)$$

Since the supervisor is considered to be deterministic we have that:

$$p(b', x^{S'}|b, x^S, e) = p(b'|b, e)\mathbf{1}_{\{f(x^S,e)\}}(x^{S'})$$

Additionally, if we consider the effects of a supervisor action and controller action:

$$\begin{aligned} p(b', x^{S'}|b, x^S, e, S, a) &= p(b'|b, e, S, a)\mathbf{1}_{\{f(x^S,e)\}}(x^{S'}) \\ &= \mathbf{1}_{\{b^{aSe}\}}(b')\mathbf{1}_{\{f(x^S,e)\}}(x^{S'}) \end{aligned} \tag{6.9}$$

Note the transition function of this automaton is exactly deterministic since for each state in $\Delta \times X^S$ and every event e, the successor state is well defined. As was done for FSAs, we could write the transition function as: $f : \Delta \times X^S \times E \to \Delta \times X^S$ with:

$$f(b, x^S, e) = (b^e, f(x^S, e))$$

but this function could not be described tabularly since the state of the automaton is uncountably infinite.

If we want to consider time in the probability expression, we have that:

$$H(t, b', x^S|b, x^S, e, S, a) = F_e(t).p(b', x^{S'}|b, x^S, e, S, a)$$

The emission probabilities do not depend directly on the state of the supervisor, although indirectly they are affected by it since the supervisor function does depend on the state of the supervisor, so we can write that:

$$p_E(e|b, x^S, S, a) = p_E(e|b, S, a) = \sum_{x \in X} p_E(e|x, S, a)b(x)$$

with $p_E(e|x, S, a)$ being the same as defined in Chapter 4.

[2] In reality, it is only the set of observable events but we are not considering that additional source of partial observability yet.

The process to derive optimality equations written in Chapter 4 can be replicated to this automaton and we have:

$$\begin{aligned} V^\pi(b,x^S) &= E\left\{ \sum_{k=0}^{\infty} e^{-\beta\sigma_k} r(b_k, x_k^S, e_k) \middle| b_0 = b, x_0^S = x^S, S, \pi \right\} \\ &= \sum_{e\in\Gamma_S(b,x^S)} p(e|b,x^S,S,\pi) E\left\{ \left[r(b,x^S,e) + e^{-\beta\tau} V^\pi(b_{k+1}, x_{k+1}^S) \middle| \pi \right\} \right] \\ &= \sum_{e\in\Gamma_S(b,x^S)} p(e|b,S,\pi) \tilde{V}_{aSe}^\pi(b,x^S) \end{aligned} \tag{6.10}$$

with

$$\tilde{V}_{aSe}^\pi(b,x^S) = \left[r(b,e) + \sum_{y^S\in X^S} \int_\Delta \left(\int_0^\infty e^{-\beta\tau} H(d\tau, b', y^S | b, x^S, e, S, a) \right) V^\pi(b', y^S) db' \right]$$

The rewards are the same as defined in Chapter 4 and do not depend on the supervisor state, and that is why we simply write $r(b,e)$. Additionally, we have that:

$$r(b,e) = \sum_{x\in X} r(x,e) b(x)$$

or, if we consider the vector r_e for the same ordering of X that defines the belief vectors:

$$r(b,e) = b \cdot r_e$$

We continue the derivation by making use of the fact that, for each event, it is possible to decouple the transition probabilities from the time distribution. We have:

$$\begin{aligned} V^\pi(b,x^S) &= \sum_{a\in A_{b,x^S}} \pi(b,x^S,a) \sum_{e\in\Gamma_S(b,x^S)} p(e|b,S,a) \tilde{V}_{aSe}^\pi(x) \\ &= \sum_{a\in A_{b,x^S}} \pi(b,x^S,a) \sum_{e\in\Gamma_S(b,x^S)} p(e|b,S,a) \left[r(b,e) + \gamma_e \, V^\pi(b^{aSe}, f(x^S,e)) \right] \end{aligned} \tag{6.11}$$

with

$$\tilde{V}_{aSe}^\pi(b,x^S) = \left[r(b,e) + \gamma_e \sum_{y^S\in X^S} \int_\Delta p(b', y^S | b, x^S, e, S, a) V^\pi(b', y^S) db' \right]$$

and $\gamma_e = \int_0^\infty e^{-\beta t} F_e(dt)$ as defined in Chapter 4.

The equivalent optimality equation can be given by:

$$V^*(b,x^S) = \max_{a\in A_{b,x^S}} \sum_{e\in\Gamma_S(b,x^S)} p(e|b,S,a) \left[r(b,e) + \gamma_e \, V^*(b^{aSe}, f(x^S,e)) \right] \tag{6.12}$$

What remains to be defined in this equation are the Γ sets and the action set for each (belief, supervisor state) pair. Since:

$$p_E(e|b, S, a) = \sum_{x \in X} p_E(e|x, S, a) b(x)$$

which means the event emission probabilities for the beliefs can be defined in terms of the equivalent quantities for the states themselves, we will start by extending the event emission function such that:

$$\forall_{e \notin \Gamma_S(x)} \;\; p(e|x, S, a) = 0$$

Additionally, we need to assume that:

$$\forall_{y \in X} \;\; e \notin \Gamma_S(x) \;\Rightarrow\; p(y|x, e) = 0$$

Technically, this makes the transition probability distribution associated with event e and state x not a well defined pdf. However, since we are doing this for $e \notin \Gamma_S(x)$, we know by definition that e will never occur in x under any condition, and we can use the extensions defined previously to be able to do $\sum_{e \in E}$.

As for the action set, In Chapter 4 we defined A_{x,x^S}[3]:

$$A_{x,x^S} = \begin{cases} \Gamma_S(x, x^S) \cap E_A & \Gamma_S(x, x^S) \setminus E_A = \emptyset \\ \big(\Gamma_S(x, x^S) \cap E_A\big) \cup \{\varepsilon_a\} & \textit{otherwise} \end{cases}$$

which seems to indicate that we need to drop the dependance on $\Gamma_S(x, x^S)$ as done for the optimality equations. We have that $\Gamma_S(x, x^S) = \Gamma(x) \cap S(x^S)$ but the only term that varies with the system (or observer) state is $\Gamma(x)$ since $S(x^S)$ only depends on the supervisor state.

Since the belief representation does not allow for an identification of the system state, we will assume the controller can choose from all the possible actions, except for the ones disabled by the supervisor. We have.

$$A_{b,x^S} = A_{x^S} = \big(S(x^S) \cap E_A\big) \cup \{\varepsilon_a\}$$

Note that it would not be possible to determine the states for which the inclusion of an idling action would not lead to a lock off the system, i.e., a situation where the controller chooses to wait for the next observable timed event to occur but the system is in a state where no events in E_T are active and will just remain in this state indefinitely. We could either not consider ε_a altogether or always include the possibility of waiting for something to happen in the environment. This choice is also somewhat related to the problem being tackled and wether there are intrinsic processes in the environment occurring spontaneously that do not depend on the choice of actions but might affect the

[3] Here we are explicitly denoting the system (or observer) state separately from the supervisor state. In Chapter 4 we assumed x referred to the extended state to avoid cluttered notation unnecessarily.

agent's plans. In the example from the previous chapters, we considered an automaton representing the arrival of people at the building where the robot lived in – that is an example of a problem where there is always an escape event caused by a process external to the robot and that can justify always including the idling action in the action space.

In the absence of such kind of processes and escape events, or the uncertainty about their existence, it is always possible to design the behavior of the agent to generate a timeout event to act as an escape for these sort of situations. In fact, timeout events are common practice in a myriad of applications and designing an automata to run in parallel with the ones defining **G** that generates such an event is fairly straightforward.

In any case, it is clear that $A_{x,x^S} \subset A_{x^S}$ for every x. For this reason, maximizing over the action set A_{x^S} never removes decision options from the agent, effectively considering more general action sets at each decision point.

Going back to the optimality equation we get:

$$V^*(b, x^S) = \max_{a \in A_{x^S}} \left[\sum_{x \in X} \sum_{e \in E} b(x) p(e|x, S, a) \left(r(x, e) + \gamma_e \, V^*(b^{aSe}, f(x^S, e)) \right) \right] \tag{6.13}$$

which has a similar form to Equation (2.24) with the differences in form residing in the fact that events, which here assume the role of observations, induce specific discount factors due to the assumption of continuous time and the fact that each event has a different time cdf associated with it. Furthermore, the value functions are defined not only in terms of beliefs but of the supervisor state. The process induced by this system is equivalent to an MDP whose state is hybrid, with a continuous component and a discrete one.

As was the case for POMDPs, there is a finite action choice at each decision point and, for each finite planning horizon, the optimal value function will also be piecewise linear and convex, which means it can be represented by a finite set of vectors.

$$V_n(b, x^S) = \max_{\alpha \in \mathcal{A}_n(x^S)} b \cdot \alpha \tag{6.14}$$

Note that now, since there is a dependence on the supervisor state, the set of vectors might be different for each supervisor state. We can continue to derive the value function expression leading to:

$$\begin{aligned} V_{n+1}(b, x^S) &= \max_{a \in A_{x^S}} \sum_{e \in E} \left[\sum_{x \in X} b(x) p(e|x, S, a) r(x, e) + \gamma_e \, p(e|b, S, a) \max_{\alpha \in \mathcal{A}_n(f(x^S, e))} b^{aSe} \cdot \alpha \right] \\ &= \max_{a \in A_{x^S}} \sum_{e \in E} \left[p(e|x, S, a) \, b \cdot r_e + \gamma_e \, p(e|b, S, a) \max_{\alpha \in \mathcal{A}_n(f(x^S, e))} b^{aSe} \cdot \alpha \right] \\ &= \max_{a \in A_{x^S}} \sum_{e \in E} \left[p(e|x, S, a) \, b \cdot r_e + \gamma_e \max_{\alpha \in \mathcal{A}_n(f(x^S, e))} \sum_{y \in X} \sum_{x \in X} p(e|x, S, a) p(y|x, e) b(x) \alpha(y) \right] \end{aligned} \tag{6.15}$$

and so we can say that

$$V_{n+1}(b, x^S) = \max_{a \in A_{x^S}} \left[\sum_{e \in E} p(e|x, S, a)\, b \cdot r_e + \sum_{e \in E} \gamma_e \max_{g_{aSe} \in \mathcal{G}_{aSe}(x^S)} b \cdot g_{aSe} \right] \tag{6.16}$$

with

$$\mathcal{G}_{aSe}(x^S) = \{ g_{aSe} : g_{aSe}(x) = \sum_{y \in X} p(e|x, S, a)\, p(y|x, e) \alpha(y) \quad , \quad \alpha \in \mathcal{A}_n(f(x^S, e)) \} \tag{6.17}$$

Finally

$$\text{backup}(b, x^S) = \arg \max_{\{g_a^b\}_{a \in A_{x^S}}} b \cdot g_a^b \tag{6.18}$$

with

$$g_a^b = \sum_{e \in E} p(e|x, S, a) r_e + \sum_{e \in E} \gamma_e \arg \max_{g_{aSe} \in \mathcal{G}_{aSe}(x^S)} b \cdot g_{aSe}^i \tag{6.19}$$

Note how the backup vector depends not only on the belief but also on the supervisor state. In fact, because the supervisor changes the actions available to the agent at each decision point, for each state of the supervisor x^S, we have a different value function $V(\cdot, x^S)$ which is PWLC. For this reason, while calculating the Bellman backup of a given vector (b, x^S), several sets of back-projected vectors g_{aSe} will be computed, depending on the state of the supervisor to which event e leads to. Worst case scenario, the number of back-projected sets will be equal to $|X^S|$.

In any case, an exact dynamic programming algorithm to solve this kind of partially observable problem will need to maintain not only one set of vectors but as many sets as there are supervisor states.

Sub-optimal equations

To achieve optimality within the restrictions imposed by the supervisor, it is important to explicitly consider its state in the optimality equations and, consequently, the existence of separate value functions (or in this case vector sets) for each of the states of the supervisor. We can think of the supervisor as dynamically switching between different problems that share some parameters like transition probabilities, but that is because the action limitations will, in general, have distinct plans as the optimal choice. Therefore, controllers have to be optimized taking into account the supervisor restrictions.

If we assume we can compute an unsupervised optimal policy for a given system, and then apply the supervisor *a posteriori*, imagining that the optimal action at a given state is then forbidden by the supervisor, there is no guarantee that the second best action under no supervision will still be so in the supervised case. The unsupervised value function does not reflect the expected sum of discounted rewards for the restricted system.

Nevertheless, for some applications it might be interesting to consider the suboptimal equations. Essentially, what we need to do is remove the dependence on x^S from the optimality equations, to obtain:

$$V^*(b) = \max_{a \in A} \left[\sum_{x \in X} \sum_{e \in E} b(x) p(e|x, S, a) r(x, e) + \sum_{x \in X} \sum_{e \in E} \gamma_e \, b(x) \, p(e|x, S, a) \, V^*(b^{aSe}) \right] \tag{6.20}$$

with the action set being defined as $A = E_A \cup \{\varepsilon_a\}$.

Under no supervision the rest of the equation will be even closer in form to the POMDP ones, retaining nevertheless some particular aspects like the event dependent discount factor. We have:

$$V_{n+1}(b) = \max_{a \in A} \left[\sum_{e \in E} p(e|x, S, a) \, b \cdot r_e + \sum_{e \in E} \gamma_e \max_{\{g^i_{aSe}\}_i} b \cdot g^i_{aSe} \right] \tag{6.21}$$

with

$$g^i_{aSe}(x) = \sum_{y \in X} p(e|x, S, a) \, p(y|x, e) \alpha^i(y)$$

for every vector $\alpha^i \in \mathcal{A}_n$. Finally

$$\text{backup}(b) = \arg \max_{\{g^b_a\}_{a \in A}} b \cdot g^b_a \tag{6.22}$$

with

$$g^b_a = \sum_{e \in E} p(e|x, S, a) r_e + \sum_{e \in E} \gamma_e \arg \max_{\{g^i_{aSe}\}_i} b \cdot g^i_{aSe} \tag{6.23}$$

The difference between these equations and the complete ones might seem like a small difference in aspect but it essentially eliminates the need for several vectors sets at each time step. All events will lead the system to a state where the same actions are available as a choice. POMDP solvers can be applied almost directly here, with the nuance that the backups, and the dynamic programming operator in general, are defined in a slightly different way because of the introduction of continuous time associated with events and how that reflects as event specific discount factors.

Looking at the equations, it is clear that events and observations share a common role in terms of providing information to improve the belief about the state we are in. However, events also influence the transition probabilities and in our case actions work by selecting which events will fire, but the transition parameters are still defined in terms of events. That is why we say events have a double role of observation and action.

Still, if we considered discount factors dependent on the observations, the POMDP equations would acquire an even closer form to these ones.

Algorithm 6.2.1 General procedure to use a Dynamic Programming POMDP algorithm. DP represents the algorithm of choice.

```
Calculate the quantities $\gamma_e$, $p(e|x, S, a)$ and $p(y|x, e)$.

for all $x^S \in X^S$ do
  $\mathcal{A}_0(x^S) \leftarrow \{\sum_{e \in E} p(e|x, S, a) r_e\}_{a \in A_{x^S}}$.
end for

loop
  for all $x^S \in X^S$ do
    $\mathcal{A}_{n+1}(x^S) \leftarrow DP(\mathcal{A}_n(\cdot))$
    {The Bellman backup, backup$(b, x^S)$, is given by Equation (6.18).}
    {The exact dynamic programming operator is given by Equation (6.12).}
  end for
end loop
```

6.2.2 Adapting POMDP Solvers

In general, an algorithm to solve a POMDP can be adapted in the following way:

As mentioned in Chapter 2, an exact algorithm will, in general, either identify the regions in which a given vector is the maximizing component or span all possible vectors and then prune dominated ones. In our case, the algorithm has to be applied several times, for each of the supervisor states, and each backup is potentially calculated using all the vector sets at a given time step, and not just the one corresponding to the supervisor state in question. That's why we wrote:

$$\mathcal{A}_{n+1}(x^S) \leftarrow DP(\mathcal{A}_n(\cdot))$$

Besides this additional complexity source, computing the emission probabilities $p(e|x, S, a)$ and the event dependent discount factors γ_e might require approximations in integration, depending on the cdfs of the events in question. We recall that:

$$p(e|x, S, \varepsilon_a) = \int_0^\infty F_e(y) \cdot dF_{W_e}(y)$$

as defined in Equation (4.8) in Chapter 4 and:

$$\gamma_e = \int_0^\infty e^{-\beta\tau} F_e(d\tau)$$

So far we have not explicitly considered the existence of random switch states but their inclusion in the equations is fairly straightforward. Essentially, the emission probabilities are given directly, as explained in Equation (4.3), and for each event in E_U we can consider that $\gamma_e = 1$ and $r(x, e) = k(x, e)$, as was explained in Chapter 4. This would cause boundedness problems but we assume we are still under the same conditions of Chapter 4, given by Assumptions 4.4.1 and 4.4.2.

6.3 Summary

If the agent has knowledge about the model parameters, under partial observability, an observer different from the one used in Chapter 5 can be constructed. In this chapter we assume we have a probabilistic observer that mimics the system not only in the logic aspect but also in the probabilities associated with the transitions between states and the emission of events. The supporting model to the observer is called a Probabilistic Discrete Event System, as defined in (Pantelic *et al.*, 2009).

With a probabilistic observer, we saw how the state representation could be distributed over the state space of the observer, representing the uncertainty about the actual state of the system. This distributed representation is essentially a probability distribution over the states and is equivalent to the notion of *belief* in Partially Observable Markov Decision Processes.

Following a derivation similar to the optimality equations for POMDPs described in Chapter 2, we obtained optimality equations for our system. Because the supervisor state is always fully observable, the state representation of the observer and supervisor has mixed observability and, for this reason, the equations we obtained are similar to the ones in Mixed Observability Markov Decision Processes (Ong *et al.*, 2009), as described briefly in Chapter 3, with some differences stemming from the event-based nature of our model and the fact that time is considered to be continuous.

Based on the derived equations, we proposed a generic algorithm to modify POMDP solvers so they can be applied to our model.

Chapter 7

Conclusions

7.1 Thesis Overview

The main goal of this work was to develop a framework that is able to integrate successfully aspects from planning, reactive control and learning under uncertainty. We presented an event-based approach to integrate deliberative knowledge, in the form of discrete event systems supervisors, and a reinforcement learning based controller, using a continuous time Q-learning algorithm, that works for Semi-Markov Decision Processes. Our approach works if full plans are provided to the agent but also if plans are loosely specified by plan options, among which the controller will learn to optimize.

The key contributions of this work are:

- **Introducing a novel approach to combine deliberative planning, reactive control and learning methods by using supervisory control of discrete event systems to provide loosely specified planning options over which a reinforcement learning based controller can optimize**. We showed how our model can be used to integrate planning, reactive control and learning using some scenarios plagued with uncertainty.

- **Defining full observability for a system based on a controllable stochastic timed automata**. Our definition of full observability requires the STA modeling to produce no unobservable events and, on the other hand, that from an initial state and a string of events the current state of the system can be univocally identified by an observer receiving that string. This translates in requiring the transition functions to be deterministic, which at first glance seems like a strong assumption. While proving the result establishing the sufficient and necessary conditions for such full observability, we discussed how the transitions probabilities, *when conditioned on the action and not the event*, would still configure a stochastic process since there is no deterministic knowledge of the event that will fire, given a certain action of the controller.
  An interesting consequence of this discussion is that events, in our model, have associated with them a double view of action and observation, patent in the two

parameters: probability of generating a given event for a certain state and action and the transition probabilities.

- **Determining the conditions that the firing of events needs to follow in order to ensure the convergence of the modified Q-learning algorithm, proving that the algorithm converges**. One of the conditions that we must have in order for the process generated by the STA to be equivalent to an SMDP is the clock reset assumption, stating that the clocks associated with each event must all be reset whenever an event fires. At first glance this assumption is not that strong if we consider, for example, that we have a robot in a certain environment and all the events are generated by the same source, the robot. However, if we consider multiple sources, the assumption is strong since it does not make sense to have the occurrence of an event in one robot reseting the temporal processes exclusively related to the other robot. In this case, many of the supervisor results will remain the same but the controller will not have a guarantee of stationary and Markovian transition functions. The study of this more general case was, nevertheless, out of the scope of this work but it would be interesting to see empirically what kind of non-Markovian problems would suffer more from using methods that are supported by a semi-Markov assumption.
  Another important condition is the fact that the time probability density functions associated with the events must not be too dense near zero, since we do not want infinite state changes to occur in finite time. For immediate events this condition is trivially broken but we showed the method would still converge if we made sure the consecutive number of immediate events was bounded. We showed how a supervisor can be constructed to ensure this condition is not broken.

- **Showing that the states of the observer, which are associated with parts of the original STA, can be proven to still provide a semi-Markov process which can be used for learning**. The use of a deterministic observer for the state of the system has the advantage of not needing more than the logic knowledge of the transition structure to be able to produce a state estimate from the sequence of events. The drawback is that, in the limit, the observer might end up reduced to a single state which corresponds to the complete original automaton, defeating the purpose of learning. Nevertheless, we shown that under the condition that each state of the observer corresponds to a semi-Markov chain with the same initial condition, we can still apply the same learning algorithms.
  Moreover, in some cases, applying the full observability algorithm directly to the states of the observer yielded good empirical results even when the system was not semi-Markovian.

- **Deriving the optimality equations for the system when the observer is a probabilistic automaton and showing how we can modify POMDP algorithms to solve our event based problem**. The use of a probabilistic automaton observer is closely tied to the concept of belief in POMDPs. For this reason, the optimality equations for an event based system such as ours have a similar form to the the POMDP ones. We showed that we also can modify Dynamic Programming methods classically used to solve POMDPs to use in our system.

One of our goals in developing this approach was that the supervisor, which was partially intended to be provided by the designer of the system, could have a representation natural enough so that the designer could easily program further restrictions in a systematic yet intuitive way. We believe automata provide one such representation but, nevertheless, other models like Petri Nets can easily be used to specify the supervisor, as shown by (Costelha and Lima, 2008). Even closer to a natural language is the work of (Lacerda and Lima, 2008), where Temporal Logic is used to provide the specifications and then transformed to an automata equivalent that can be used for supervision and which, by having a state representation, can be directly used by the reinforcement learning controller.

7.2 Directions for Future Work

There are several possible ways this work can be extended:

- In multi-agent scenarios the clock reset assumption might become too strong since multi-agents imply multiple event sources that most likely will not influence the firing mechanisms of the events from each other. This will pose a problem since, in this case, it is no longer possible to reduce the process to an equivalent of a SMDP, and ideally a memory of the time for which each event has already been active would be needed in order to provide a complete state representation. But in this situation, not only the state space becomes non-countable but, most of the times, it may not be possible to access the internal time values of each of the other agents.
  A possible simplification is that, since each agent is a potential event source, all events generated by a source will still be reset when one of them occurs. This remove the need for an extended state representation but generally it will reduce the number of sources from the number of events to the number of agents in the environment.
  The most common simplification is to assume the firing of all events is modeled by a Poisson process, effectively making the system fully markovian and removing the need for a time memory. This is what is commonly used in Stochastic Petri Nets, but in general an exponential distribution will not always adequately describe the firing of all kinds of events.

- Also in multi-agent scenarios, it is possible to define special events to synchronize between agents and ensure decisions are taken simultaneously. In that case, particularly in one-on-one adversarial situations (where the two agents have opposite objectives) or team situations (where all the agents have the same objective), the combined action of the agents will have equilibrium points (Nash, 1950) and the system will behave as a stochastic game (Shapley, 1953). In that case, the extensive body of work on multi-agent learning (Littman, 1994; Singh *et al.*, 2000*a*; Hu and Wellman, 2004; Bowling, 2003; Wang and Sandholm, 2002; Bowling *et al.*, 2004) can be applied to the problem. Another interesting work on coordination in a multi-agent scenario that could possibly be integrated with our framework is that of (Kok *et al.*, 2005).

Since our system is asynchronous by nature, the use of synchronization events would become crucial.

- Furthermore, when we consider a decentralized approach in our architecture, which is only natural in multi-agent scenarios, the results from DecSC (Lin and Wonham, 1990; Rudie and Wonham, 1992; Yoo and Lafortune, 2004) become of critical importance and any extension of this work to multi-agent scenarios will have to consider the implications of several sources potentially enabling/disabling the same events. In Figure 7.1 we show the general architecture in the decentralized case.

  Generally, since controllable events correspond to the start and stop of actions in our model, each controller policy will be defined over a separate set of events but, nevertheless, we can always assume that different controllers can actually start the same behavior, and in that case the combination of the controller policies is not trivial and the choice of a combination function would pose an interesting problem in itself.

- Predictive State Representations (Littman *et al.*, 2002; Singh *et al.*, 2004; Wolfe *et al.*, 2008) are a recent formalism to deal with typical POMDP problems that work directly on the observations and actions strings that the system generates, without explicitly using a state representation. Some of the recent work on PSRs tries to learn a predictive state representation (James and Singh, 2004; Wolfe *et al.*, 2005; McCracken and Bowling, 2006; Wingate and Singh, 2007) for the system it is addressing. This would be interesting in situations where the transition logic, and system states, of the underlying STA is not necessarily known. In fact, the way PSRs work is closely related to DES since both work on strings of symbols, in one case events and in the other observations and actions. In (Wolfe and Singh, 2006), the options formalism of (Sutton *et al.*, 1999) is approached from a PSR perspective, and since options provide a way to solve some SMDP problems, there is certainly room for related work having this thesis' approach as starting point.

- Finally, the implementation of this approach in real scenarios, e.g. robotic scenarios like the one we presented as a simulated case study, would face some more challenges that would pose interesting to overcome.

  1. One of the challenges is the existence of other processes in the environment that cause events to fire, and, as discussed previously, how the *assumption of reseting the ages of events at each state change might not necessarily hold.*

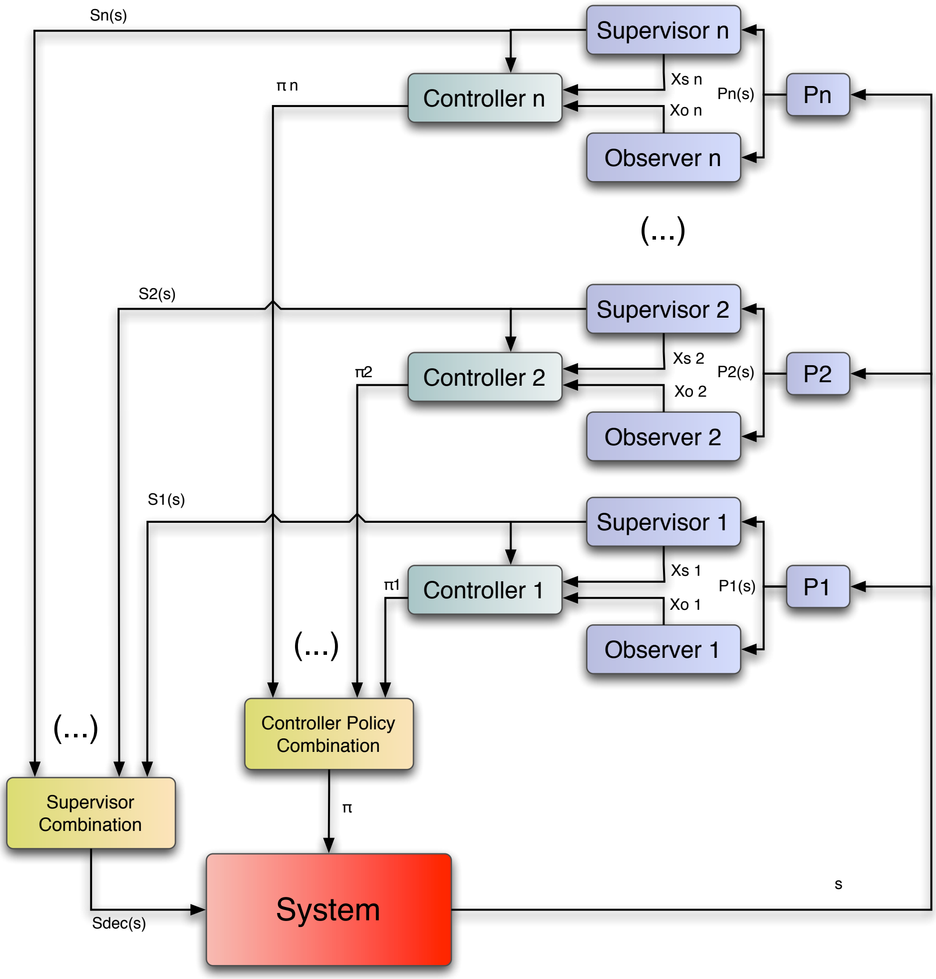


Figure 7.1: Decentralized extension of the work done in this thesis.

A common approximation is to consider inter-event times to always be exponentially distributed, which we believe to be too strong for some applications.

2. *Unsound observer models* can also pose a problem. For the method proposed in Chapter 6 the lack of knowledge about the system trivially makes it impossible to use without relying on additional methods to estimate the system parameters. But even in Chapters 4 and 5, an observer must have knowledge about the logic aspect of the system transition structure. If there is some leeway in choosing some of the events the system uses, it is always a good idea to try to pick events that univocally identify one state or a small group of states.

3. Part of the idea of integrating a learning controller in our approach is to provide some degree of adaptation. However, reinforcement learning algorithms require the system parameters to be stationary, but usually still perform well if the change in the parameters is not too fast. The *usability of the learning algorithm in fast changing environments* is another hurdle that real problems would create to the approach, because of the nature of the learning algorithms used. Nevertheless, using a supervisor is also a way of ensuring that, even if the parameters change, there are boundaries that the agent/robot will not pass, keeping itself safe and ensuring continued action. On the other hand, with development of new algorithms and increase in computational power, the real applications of reinforcement learning are becoming widespread, with recent examples like (Peters and Schaal, 2006; Kober *et al.*, 2008; Riedmiller *et al.*, 2009; Vlassis *et al.*, 2009) proposing interesting and effective applications of reinforcement learning to robot control,

4. As mentioned before, the extension to multi agent scenarios would necessarily have to include *communication and synchronization*, since our event-based model is asynchronous by nature. The models used by (Costelha and Lima, 2010) for communication mentioned in Chapter 3 offer an interesting way to integrate it in an event-based framework, particularly because they account for message travel time and communication failure.

Bibliography

Aoki, Masanao (1965). Optimal control of partially observable markovian systems. *Journal of the Franklin Institute* **280**(5), 367 – 386.

Aström, K. J. (1965). Optimal control of Markov decision processes with incomplete state estimation. *Journal of Mathematical Analysis and Applications* **10**, 174–205.

Bellman, Richard Ernest (1957). *Dynamic Programming*. Princeton Press.

Bernstein, Daniel S., Shlomo Zilberstein and Neil Immerman (2000). The complexity of decentralized control of markov decision processes. In: *UAI '00: Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence*. Morgan Kaufmann Publishers Inc.. San Francisco, CA, USA. pp. 32–37.

Bertsekas, Dimitri P. (1982). Distributed dynamic programming. *IEEE Transactions on Automatic Control* **27**, 610–616.

Bertsekas, Dimitri P. (1983). Distributed asynchronous computation of fixed points. *Mathematical Programming* **27**, 107–120.

Bertsekas, Dimitri P. (1987). *Dynamic Programming: Deterministic and Stochastic Models*. Prentice-Hall.

Bertsekas, Dimitri P. and John N. Tsitsiklis (1989). *Parallel and Distributed Computation: Numerical Methods*. Prentice-Hall.

Bertsekas, Dimitri P. and John N. Tsitsiklis (1996). *Neuro-Dynamic Programming*. Athena Scientific.

Bhulai, Sandjai (2002). Markov Decision Processes: the control of high-dimensional systems. PhD thesis. Vrije Universiteit.

Bowling, Michael (2003). Multiagent Learning in the Presence of Agents with Limitations. PhD thesis. Carnegie Mellon University. Pittsburgh.

Bowling, Michael, Brett Browning and Manuela Veloso (2004). Plays as effective multiagent plans enabling opponent-adaptive play selection. In: *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling*.

Bradtke, Steven J. and Michael O. Duff (1995). Reinforcement learning methods for continuous-time markov decision problems. In: *Advances in Neural Information Processing Systems*. pp. 393–400.

Cassandra, Anthony R., Michael L. Littman and Nevin L. Zhang (1997). Incremental pruning: A simple, fast, exact method for partially observable markov decision processes. In: *Proceedings of the 13th Annual Conference on Uncertainty in Artificial Intelligence*. Morgan Kaufmann. Providence, Rhode Island. pp. 54–61.

Cassandra, Anthony Rocco (1998). Exact and approximate algorithms for partially observable markov decision processes. PhD thesis. Brown University. Providence, RI, USA. Adviser-Kaelbling, Leslie Pack.

Cassandras, Christos G. and Stéphane Lafortune (2007). *Introduction to Discrete Event Systems*. Springer.

Cheng, Hsien-Te (1989). Algorithms for partially observable markov decision processes. PhD thesis. University of British Columbia. Supervisor-Brumelle, Shelby.

Chomsky, Noam (1955). Transformational Analysis. PhD thesis. University of Pennsylvania.

Costelha, Hugo and Pedro Lima (2008). Modelling, analysis and execution of multi-robot tasks using petri nets. In: *AAMAS '08: Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems*. International Foundation for Autonomous Agents and Multiagent Systems. Richland, SC. pp. 1187–1190.

Costelha, Hugo and Pedro Lima (2010). Petri net robotic task plan representation: Modelling, analysis and execution.. In: *Autonomous Agents*. IN-TECH.

Dietterich, Thomas G. (1998). The maxq method for hierarchical reinforcement learning. In: *ICML '98: Proceedings of the Fifteenth International Conference on Machine Learning*. Morgan Kaufmann Publishers Inc.. San Francisco, CA, USA. pp. 118–126.

Dietterich, Thomas G. (2000). Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research* **13**, 227–303.

Ferrein, Alexander, Pauli, Josef, Siebel, Nils T and Steinbauer, Gerald, Eds.) (2009). *Proceedings of the 1st International Workshop on Hybrid Control of Autonomous Systems (HYCAS 2009)*. Pasadena, USA.

Gabel, Thomas and Martin Riedmiller (2008). Evaluation of batch-mode reinforcement learning methods for solving dec-mdps with changing action sets. *Recent Advances in Reinforcement Learning* pp. 82–95.

Glynn, Peter W. (1989). A GSMP formalism for discrete event systems. In: *Proceedings of the IEEE*. Vol. 77. pp. 14–23.

Granas, Andrzej and James Dugundji (2003). *Fixed Point Theory*. Springer Monographs in Mathematics. Springer-Verlag. New York.

Hansen, Eric A., Daniel S. Bernstein and Shlomo Zilberstein (2004). Dynamic programming for partially observable stochastic games. In: *Proceedings of the Nineteenth National Conference on Artificial Intelligence*. San Jose.

Hennet, J.-C. (1993). A graph formulation of some supervisory control problems. In: *Systems, Man and Cybernetics, 1993. 'Systems Engineering in the Service of Humans', Conference Proceedings., International Conference on*. pp. 601 –606 vol.1.

Hopcroft, John E., Rajeev Motwani and Jeffrey D. Ullman (2000). *Introduction to Automata Theory, Languages, and Computation*. 2nd ed.. Addison Wesley.

Howard, Richard Author (1960). *Dynamic Programming and Markov Decision Processes*. MIT Press.

Howard, Richard Author (1963). Semi-markovian decision processes. In: *Proceedings International Statistical Institute*. pp. 625–652.

Hu, Junling and Michael P. Wellman (2004). Nash Q-learning for general-sum stochastic games. *Journal of Machine Learning Research* **4**, 1039–1069.

Istratescu, Vasille I. (1981). *Fixed Point Theory: An Introduction*. Vol. 7 of *Mathematics and Its Applications*. D. Reidel. Holland.

Jaakkola, Tommi, Michael I. Jordan and Satinder P. Singh (1994). On the convergence of stochastic iterative dynamic programming algorithms. In: *Neural Computation*. Vol. 6. pp. 1185–1201.

James, Michael R. and Satinder Singh (2004). Learning and discovery of predictive state representations in dynamical systems with reset. In: *Proceedings of the 21th International Conference on Machine Learning*.

Jung, Hwisung and M. Pedram (2006). Stochastic dynamic thermal management: A markovian decision-based approach. In: *Computer Design, 2006. ICCD 2006. International Conference on*. pp. 452 –457.

Kaelbling, Leslie Pack, Michael L. Littman and Anthony R. Cassandra (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence* **101**, 99–134.

Kajiwara, Kouji and Tatsushi Yamasaki (2009). Reinforcement learning of the supervisor based on the worst-case behavior under partial observation. In: *IEICE Tech. Rep.*. CST2009-1. Osaka. pp. 1–6. Wed, Jun 3, 2009 - Thu, Jun 4 : Setsunan University, Osaka Center (CST).

Kajiwara, Kouji and Tatsushi Yamasaki (June 2010). An optimal supervisory control for decentralized discrete event systems based on reinforcement learning. In: *IEICE Tech. Rep.*. CST2010-26. Hokkaido. pp. 145–150. Mon, Jun 21, 2010 - Tue, Jun 22 : Kitami Institute of Technology (CAS, CST, VLD, SIP).

Kober, Jens, Betty Mohler and Jan Peters (2008). Learning perceptual coupling for motor primitives. In: *in Proceedings of the IEEE/RSJ 2008 International Conference on Intelligent RObots and Systems (IROS*. pp. 834–839.

Kok, Jelle R., Matthijs T. J. Spaan and Nikos Vlassis (2005). Non-communicative multi-robot coordination in dynamic environments. *Robotics and Autonomous Systems* **50(2-3)**, 99–114.

Kulkarni, V. (1995). *Modeling and Analysis of Stochastic Systems*. Chapman & Hall.

Kumar, R. and V.K. Garg (1998). Control of stochastic discrete event systems: synthesis. In: *Decision and Control, 1998. Proceedings of the 37th IEEE Conference on*. Vol. 3. pp. 3299 –3304 vol.3.

Kumar, R. and V.K. Garg (2001). Control of stochastic discrete event systems modeled by probabilistic languages. *Automatic Control, IEEE Transactions on* **46**(4), 593 –606.

Kushner, Harold J. and G. George Yin (2003). Stochastic approximation and recursive algorithms and applications. In: *Applications of Mathematics*. Vol. 35. Springer-Verlag.

Lacerda, Bruno and Pedro Lima (2008). Linear-time temporal logic control of discrete event models of cooperative robots. *Journal of Physical Agents*.

LaValle, S. M. (2006). *Planning Algorithms*. Cambridge University Press. Cambridge, U.K. Available at http://planning.cs.uiuc.edu/.

Lawford, M. and W.M. Wonham (1993). Supervisory control of probabilistic discrete event systems. In: *Proceedings of the 36th Midwest Symposium on Circuits and Systems*. Vol. 1. IEEE. pp. 327–331.

Lin, F. and W.M. Wonham (1990). Decentralized control and coordination of discrete-event systems with partial observation. *Automatic Control, IEEE Transactions on* **35**(12), 1330 –1337.

Littman, Michael L. (1994). Markov games as a framework for multi-agent reinforcement learning. In: *Proceedings of the Thirteenth International Conference on Machine Learning*. New Brunswick. pp. 157–163.

Littman, Michael L., Richard S. Sutton and Satinder P. Singh (2002). Predictive representations of state. In: *Advances in Neural Information Processing Systems 14*. pp. 1555–1561.

Lopez, Gabriel G. Infante, Holger Hermanns and Joost pieter Katoen (2001). Beyond memoryless distributions: Model checking semi-markov chains. In: *In Proceedings of the Joint International Workshop on Process Algebra and Probabilistic Methods, Performance Modeling and Verification, volume 2165 of LNCS.* Springer-Verlag. pp. 57–70.

Lovejoy, William S. (1991). Computationally feasible bounds for partially observed markov decision processes. *Oper. Res.* **39**(1), 162–175.

Mahadevan, Sridhar (1998). Partially-observable semi-markov decision processes: Theory and applications in engineering and cognitive science. In: *AAAI Fall Symposium on Partially Observable Markov Decision Processes (POMDPs).*

Mahadevan, Sridhar and Nikfar Khaleeli (1999). Robust mobile robot navigation using partially-observable semi-markov decision processes. *51] P. Maybeck. Stochastic Models, Estimation, and Control.*

McCracken, Peter and Michael Bowling (2006). Online discovery and learning of predictive state representation. In: *Advances in Neural Information Processing System 18.*

Melo, Francisco A. (2007). Reinforcement Learning in Cooperative Navigation Tasks. PhD thesis. Instituto Superior Técnico.

Molloy, M.K. (1982). Performance analysis using stochastic petri nets. *IEEE Transactions on Computers* **31**, 913–917.

Monahan, George E. (1982). State of the Art–A Survey of Partially Observable Markov Decision Processes: Theory, Models, and Algorithms. *MANAGEMENT SCIENCE* **28**(1), 1–16.

Moor, T. and J. Raisch (2002). Think continuous, act discrete: DES techniques for continous systems. In: *Proceedings of the 10th Mediterranean Conference on Control and Automation.* Lisbon.

Morimoto, Jun and Kenji Doya (2005). Robust reinforcement learning. *Neural Comput.* **17**(2), 335–359.

Murata, T. (1989). Petri nets: Properties, analysis and applications. *Proceedings of the IEEE* **77**(4), 541 –580.

Nair, Ranjit, Milind Tambe, Makoto Yokoo, David Pynadath, Stacy Marsella, R. Nair and M. Tambe (2003). Taming decentralized pomdps: Towards efficient policy computation for multiagent settings. In: *In IJCAI.* pp. 705–711.

Nash, John F. (1950). Equilibrium points in n-person games. In: *Classics in Game Theory.* Princeton University Press.

Neto, Goncalo and Pedro Lima (2008). Combining supervisory control of discrete event systems and reinforcement learning to control multi-robot systems. In: *Proceeding of the Workshop on Formal Models and Methods for Multi-Robot Systems, AAMAS 2008 - the 7th International Conference on Autonomous Agents and Multiagent Systems*. Estoril, Portugal. pp. 23–28.

Oliehoek, Frans A., Matthijs T. J. Spaan and Nikos Vlassis (2008). Optimal and approximate q-value functions for decentralized pomdps. *J. Artif. Int. Res.* **32**(1), 289–353.

Ong, Sylvie C. W., Shao W. Png, David Hsu and Wee S. Lee (2009). Pomdps for robotic tasks with mixed observability. In: *Robotics: Science and Systems*. Vol. 5.

Pantelic, V., S.M. Postma and M. Lawford (2009). Probabilistic supervisory control of probabilistic discrete event systems. *Automatic Control, IEEE Transactions on* **54**(8), 2013 –2018.

Parr, Ronald (1998*a*). Hierarchical Control and Learning for Markov Decision Processes. PhD thesis. University of California at Berkeley.

Parr, Ronald (1998*b*). A unifying framework for temporal abstraction in stochastic processes. In: *Symposium on Abstraction Reformulation and Approximation*.

Parr, Ronald and Stuart Russel (1998). Reinforcement learning with hierarchies of machines. In: *NIPS 97*. pp. 1043 – 1049.

Perkins, Theodore J. and Andrew G. Barto (2003). Lyapunov design for safe reinforcement learning. *J. Mach. Learn. Res.* **3**, 803–832.

Peters, Jan and Stefan Schaal (2006). Policy gradient methods for robotics. In: *In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS*.

Petri, Carl Adam (1966). Kommunikation mit automaten. Technical report. New York: Griffiss Air Force Base. English translation.

Pineau, Joelle, Geoff Gordon and Sebastian Thrun (2003). Point-based value iteration: An anytime algorithm for pomdps. In: *Proc. Int. Joint Conf. on Artificial Intelligence*. Acapulco, Mexico.

Postma, Steven and Mark Lawford (2004). Computation of probabilistic supervisory controllers for model matching. In: *Proceedings of the 42nd Annual Allerton Conference on Communication, Control, and Computing*.

Precup, Doina (2000). Temporal Abstraction in Reinforcement Learning. PhD thesis. University of Massachusetts. Amherst, MA.

Puterman, Martin L. (1994). *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley-Interscience.

Rabin, Michael O. (1963). Probabilistic automata. *Information and Control* **6**(3), 230 – 245.

Ramadge, P. and W. Wonham (1984). Supervisory control of a class of discrete event processes. *Analysis and Optimization of Systems* pp. 475–498.

Ramadge, P. J. (1983). Control and Supervision of Discrete Event Processes. PhD thesis. University of Toronto.

Ramadge, P. J. and W.M. Wonham (1982*a*). Supervision of discrete event processes. In: *Proceedings of the 21st IEEE Conference on Decision and Control.* pp. 1228–1229.

Ramadge, P. J. and W.M. Wonham (1982*b*). Supervisory control of discrete event processes. In: *Feedback Control of Linear and Nonlinear Systems.* pp. 202–214. Number 39 In: *Lecture Notes in Control and Information Sciences.* Springer-Verlag. Berlin.

Ramadge, P. J. and W.M. Wonham (1987). Supervisory control of a class of discrete event processes. *SIAM Journal of Control and Optimization* **25**(1), 206–230.

Ray, A. (2005). Signed real measure of regular languages for discrete event supervisory control. *International Journal of Control* **78**(12), 949–967.

Riedmiller, Martin, Thomas Gabel, Roland Hafner and Sascha Lange (2009). Reinforcement learning for robot soccer. *Autonomous Robots* **27**(1), 55–73.

Rosenstein, M. T. and Andrew G. Barto (2004). Supervised actor-critic reinforcement learning. In: *Learning and Approximate Dynamic Programming: Scaling Up to the Real World.* pp. 359–380. John Wiley & Sons.

Ross, Sheldon M. (1983). *Introduction to Stochastic Dynamic Programming.* Academic Press.

Rudie, K. and W.M. Wonham (1992). Think globally, act locally: decentralized supervisory control. *Automatic Control, IEEE Transactions on* **37**(11), 1692–1708.

Rudie, Karen (2002). The current state of decentralized discrete-event control systems. In: *Proceedings of the 10th Mediterranean Conference on Control and Automation.* Lisbon.

Rudie, Karen, Stéphane Lafortune and Feng Lin (2003). Minimal communication in a distributed discrete-event system. *IEEE Transactions on Automatic Control.*

Rummery, G. A. and M. Niranjan (1994). On-line q-learning using connectionist systems. Technical Report CUED/F-INFENG/TR166. Cambridge University.

Russell, S. J. and Norvig (2003). *Artificial Intelligence: A Modern Approach (Second Edition).* Prentice Hall.

Shapley, L. S. (1953). Stochastic games. In: *Classics in Game Theory.* Princeton University Press.

Singh, Satinder P., Michael J. Kearns and Yishay Mansour (2000*a*). Nash convergence of gradient dynamics in general-sum games. In: *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence.*

Singh, Satinder P., Michael R. James and Matthew R. Rudary (2004). Predictive state representations: A new theory for modeling dynamical systems. In: *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence.* pp. 512–519.

Singh, Satinder P., Tommi Jaakkola, Michael L. Littman and Csaba Szepesvári (2000*b*). Convergence results for single-step on-policy reinforcement learning algorithms. *Machine Learning* **38**, 287–308.

Smallwood, Richard D. and Edward J. Sondik (1973). The Optimal Control of Partially Observable Markov Processes over a Finite Horizon. *OPERATIONS RESEARCH* **21**(5), 1071–1088.

Sondik, E. J. (1971). The optimal control of partially observable Markov processes. PhD thesis. Stanford University.

Spaan, Matthijs T. J. (2006). Approximate planning under uncertainty in partially observable environments. PhD thesis. Universiteit van Amsterdam.

Spaan, Matthijs T. J. and Nikos Vlassis (2004). A point-based POMDP algorithm for robot planning. In: *Proceedings of the IEEE International Conference on Robotics and Automation.* New Orleans, Louisiana.

Stolle, Martin and Doina Precup (2002). Learning options in reinforcement learning. *Abstraction, Reformulation, and Approximation* pp. 212–223.

Sutton, Richard S. (1996). Generalization in reinforcement learning: Successful examples using sparse coding. In: *Advances in Neural Information Processing Systems 8.*

Sutton, Richard S. and Andrew G. Barto (1998). *Reinforcement Learning.* MIT Press.

Sutton, Richard S., Doina Precup and Satinder P. Singh (1999). Between MDPs and semi-MDPs: a framework for temporal abstraction in reinforcement learning. *Artificial Intelligence* **112**, 181–211.

Thrun, Sebastian B. (1992). The role of exploration in learning control. In: *Handbook of Intelligent Control.* Van Nostrand Reinhold.

Vlassis, Nikos, Marc Toussaint, Georgios Kontes and Savas Piperidis (2009). Learning model-free robot control by a Monte Carlo EM algorithm. *Autonomous Robots* **27**(2), 123–130.

Wang, Xi and A. Ray (2002). Signed real measure of regular languages. In: *American Control Conference, 2002. Proceedings of the 2002*. Vol. 5. pp. 3937 – 3942 vol.5.

Wang, Xiaofeng and Tuomas Sandholm (2002). Reinforcement learning to play an optimal nash equilibrium in team markov games. In: *Advances in Neural Information Processing Systems 14 (NIPS)*.

Watkins, Christopher J. C. H. (1989). Learning from Delayed Rewards. PhD thesis. King's College. Cambridge, UK.

Watkins, Christopher J. C. H. and Peter Dayan (1992). Technical note: Q-learning. *Machine Learning* **8**, 279–292.

Wingate, David and Satinder Singh (2007). On discovery and learning of models with predictive representations of state for agents with continuous actions and observations. In: *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems*.

Wolfe, Britton and Satinder Singh (2006). Predictive state representations with options. In: *Proceedings of the 23rd International Conference on Machine Learning*.

Wolfe, Britton, Michael R. James and Satinder Singh (2005). Learning predictive state representations in dynamical systems without reset. In: *Proceedings of the 22nd international conference on Machine learning*.

Wolfe, Britton, Michael R. James and Satinder Singh (2008). Approximate predictive state representations. In: *Proceedings of the Seventh International Conference on Autonomous Agents and Multiagent Systems*.

Wonham, W. and P. Ramadge (1988). Modular supervisory control of discrete-event systems. *Mathematics of Control, Signals, and Systems (MCSS)* **1**(1), 13–30.

Wonham, W.M. (1997). Supervisory control of discrete-event systems. Technical report. Systems Control Group, University of Toronto.

Yamasaki, T., K. Taniguchi and T. Ushio (2005). Reinforcement learning of optimal supervisor based on language measure. In: *Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC '05. 44th IEEE Conference on*. pp. 126 – 131.

Yamasaki, Tatsushi and Toshimitsu Ushio (2003). Supervisory control of partially observed discrete event systems based on reinforcement learning. In: *Proceedings of the 2003 IEEE International Conference on Systems, Man & Cybernetics*. pp. 2956–2961.

Yamasaki, Tatsushi and Toshimitsu Ushio (2005). Decentralized supervisory control of discrete event systems based on reinforcement learning. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.* **E88-A**(11), 3045–3050.

Yamasaki, Tatsushi and Toshimitsu Ushio (2008). Supervisory control of timed discrete event systems based on reinforcement learning. In: *IEICE Tech. Rep.*. CST2008-10. Aichi. pp. 31–36. Mon, Jun 2, 2008 - Tue, Jun 3 : Nagoyo University, Noyori Conference Hall (CST).

Yoo, Tae-Sic and S. Lafortune (2002*a*). Decentralized supervisory control: a new architecture with a dynamic decision fusion rule. In: *Discrete Event Systems, 2002. Proceedings. Sixth International Workshop on*. pp. 11 – 17.

Yoo, Tae-Sic and S. Lafortune (2004). Decentralized supervisory control with conditional decisions: supervisor existence. *Automatic Control, IEEE Transactions on* **49**(11), 1886–1904.

Yoo, Tae-Sic and Stéphane Lafortune (2000). New results on decentralized supervisory control of discrete-event systems. In: *IEEE Conference on Decision and Control*. pp. 1–6.

Yoo, Tae-Sic and Stéphane Lafortune (2002*b*). A general architecture for decentralized supervisory control of discrete-event systems. *Discrete Event Dynamic Systems: Theory and Applications* **12**(3), 335–377.

Yu, Huizhen (2006). Approximate solution methods for partially observable markov and semi-markov decision processes. PhD thesis. Massachusetts Institute of Technology. Cambridge, MA, USA. Adviser-Bertsekas, Dimitri P.

Appendix A

Some Fixed Point Results

The existence of a solution to many dynamic programming algorithms like value iteration is supported on the existence of fixed points for the dynamic programming operator being considered. Under certain conditions, the uniqueness of a fixed point can be guaranteed and the iterative application of the operator will converge to that fixed point. In this appendix we present some fixed point theorems that support classical results of dynamic programming and some of the results of this work. The results presented in this chapter can be found in (Istratescu, 1981; Granas and Dugundji, 2003)

A.1 Basic Definitions

We will start with some basic definitions that support the fixed point theorems used.

Definition A.1.1. *Let X be a nonempty set and $d : X \times X \to \mathbb{R}$.*

The function d is called a metric on X (or distance) iff the following properties hold:

1. *$d(x, y) = 0$ iff $x = y$*
2. *$d(x, y) = d(y, x)$ for all $x, y \in S$*
3. *$d(x, z) \leq d(x, y) + d(y, z)$ for all $x, y, z \in S$*

The number $d(x, y)$ is called the distance between x and y and the pair (X, d) is called a metric space*.*

An important class of metric spaces are called *complete metric spaces.*

Definition A.1.2. *A metric space (X, d) is called* complete *if any Cauchy sequence x_n in X has the property that it converges to a point of X.*

The notion of *norm* can be defined as:

Definition A.1.3. *Let X be a topological vector space over a topological field K (most often $\mathbb{C}$ or $\mathbb{R}$). A* norm *on X is any function $p : X \to \mathbb{R}$ with the following properties:*

1. $p(x+y) \leq p(x) + p(y)$
2. $p(ax) = |a|p(x)$, $a \in K$
3. $p(x) = 0$ *iff* $x = 0$

We often write a norm $p(x)$ *as* $\|x\|$.

A space X endowed with a norm $\|.\|$ is called a *normed space*. An important class of normed spaces are called *Banach spaces* which can be defined as:

Definition A.1.4. *A normed space* X *with the norm* $\|.\|$ *is called a* Banach space *if* X *is a complete metric space for the metric* d *defined on* X *by:*

$$d(x,y) = \|x - y\|$$

A.2 Fixed Point Theorems

Before referring to fixed point theorems, let us consider some definitions. Suppose (X,d) is a complete metric space and $f : X \to X$ is any function.

Definition A.2.1. *The function* f *is said to satisfy a Lipschitz condition with constant* $k > 0$ *if:*

$$d(f(x), f(y)) \leq k\, d(x,y)$$

holds for all $x, y \in X$.

Definition A.2.2. *If* f *satisfies a Lipschitz condition with* $0 < k < 1$ *it is called a* contraction mapping. *If* $k = 1$ *the mapping is called* nonexpansive.

The fact that an operator is a contraction mapping will guarantee an existence and uniqueness of a fixed point for the operator, i.e., a point for which any application of the operator will leave it unchanged. The classical result that guarantees this is called the *Banach fixed point theorem* or *Contraction mapping theorem.*

Theorem A.2.1 (Banach fixed point theorem)**.** *If* $T : X \to X$ *is a contraction mapping with Lipschitz constant* $0 < k < 1$ *on a complete metric space* (X,d) *then there exists a unique fixed point* x^* *of* T *and for any point* $x \in X$ *and* $x_n = T^n x$:

1. $d(x_n, x^*) \leq \frac{k^n}{1-k}$
2. $\lim x_n = x^*$

Particularly, this principle is often applied to Banach spaces with the associated norm inducing the metric. Such is the case of the dynamic programming operators defined for MDPs and SMDPs in Chapter 2.

Several extensions of the fixed point theorem exist and we're particularly interested in one that is supported on the notion of a local power contraction mapping.

Definition A.2.3 (Local Power Contraction Mapping)**.** *Let (X, d) be a complete metric space and $T : X \to X$ be a continuous mapping. The mapping T is called a local power contraction mapping if there exists a constant $k < 1$ and for each $x \in X$ there exists an integer $n = n(x)$ such that, for all $y \in X$,*

$$d(T^n x, T^n y) \leq k\, d(x, y)$$

The fixed point theorem presented previously can be extended for local power contraction mappings in the following way:

Theorem A.2.2. *If T is a local power contraction mapping then there exists a unique fixed point of T.*