WARSAW UNIVERSITY

OF TECHNOLOGY

Faculty of Electronics and Information Technology

Ph.D. THESIS

Bartosz Papis, M.Sc.

State Abstraction in Reinforcement Learning

Supervisor
Professor Andrzej Pacut, Ph.D., D.Sc.

Warsaw, 14.IV.2015

Acknowledgements

I wish to thank my advisor, Professor Andrzej Pacut for sharing his vast knowledge, for his inexhaustible reserves of patience, and for discussions so interesting that they usually went far beyond the scope of this work. I am also very grateful for his direct and friendly attitude, which made our collaboration natural and enjoyable.

I would also like to express my gratitude to Dr. Paweł Wawrzyński for his advices, ideas and support.

إلى ندى

Streszczenie

Niniejsza praca dotyczy zagadnienia abstrakcji stanu (wyodrębniania stanu) - jednego z powszechnie proponowanych rozwiązań problemu *przekleństwa wymiarowości*. Szczególny rodzaj abstrakcji stanu - abstrakcja przestrzeni stanu jest analizowany jako problem selekcji zmiennych. Efektem tej analizy jest zapropowany w tej pracy przyrostowy algorytm abstrakcji stanu, inspirowany pojęciami *warunkowania instrumentalnego*, *niejednoznaczności* i *domknięcia* z psychologii behawioralnej. Algorytm ten poprawnie rozwiązuje problem selekcji zmiennych poprzez dodawanie lub usuwanie pojedynczych zmiennych. Jest to pierwsze wśród istniejących rozwiązań działające nie tylko dla problemów dyskretnych, ale także ciągłych.

Abstract

This work concerns state abstraction - one of commonly proposed solutions to the *curse of dimensionality* problem. A particular type of state abstraction - state space abstraction is analyzed as a variable selection issue. As an effect of this analysis, an incremental state abstraction algorithm is introduced, inspired by the notions of *stimulus discrimination*, *ambiguity* and *closure* from behavioral psychology. This algorithm correctly solves the variable selection problem by including or removing variables one by one. It is the first among existing solutions to work not only for discrete problems, but also continuous ones.

Contents

Notation and Abbreviations

- x - a scalar
- $\mathbf{x}$ - a vector
- $\mathbf{x}_i$ - $\mathbf{x}$'s i-th element
- $(x^1,\ldots,x^i,\ldots,x^n)$ - a vector with given elements (all vectors are row vectors)
- $\mathfrak{X}$ - a set or a space of measurables, with its vector elements $\mathbf{x} \in \mathfrak{X}$ called measurements or vectors of measurables
- f_x - transition function
- f_r - reinforcement function
- $u(t)$ - action/control at time instant t
- $\mathbf{x}(t)$ - measurement at time instant t ($\mathbf{x}(t+1) = f_x(\mathbf{x}(t), u(t))$)
- $r(t)$ - reinforcement at time instant t ($r(t) = f_r(\mathbf{x}(t), u(t))$)
- $[x]$ - an equivalence class of an element x
- $\hat{x}$ - an abstract element, being a representative of the equivalence class $[x]$
- $\hat{\mathbf{x}}$ - an abstract vector, being a representative of the equivalence class of measurements $[\mathbf{x}]$
- $\hat{\mathfrak{X}}$ - an abstract set or an abstract space of measurables, being a set of representatives of all equivalence classes
- ϕ - an abstraction mapping (an abstraction)
- $\|\mathfrak{X}\|$ - $\mathfrak{X}$'s cardinality
- LHS, RHS - left hand side, right hand side
- $\mathbf{x} > \mathbf{y}$ - a predicate that is true $\iff \underset{i \in 1,\ldots,\text{len}(x)}{\forall} x_i > y_i$ (similarly for $<$)
- $len(\mathbf{x})$ - the number of elements in the given vector $\mathbf{x}$
- $|\mathbf{x}|$ - vector of absolute values of elements of the given vector $\mathbf{x}$: $|\mathbf{x}| = (|\mathbf{x}_1|,\ldots,|\mathbf{x}_i|,\ldots,|\mathbf{x}_{len(\mathbf{x})}|)$

- $\mathscr{E}$ - mean
- $\mathscr{V}$ - variance

Chapter 1

Introduction

This dissertation is dedicated to the problem of accidental redundancy. In Computer Science, this phenomenon occurs in such areas as industrial control systems, robotics, image processing, sensor array data analysis, gene micro-array data analysis, medical records processing, weather forecasting or document processing. Accidental redundancy can occur when vast amounts of general-purpose data are automatically recorded and then reused for different specific tasks [85]. Apart from that, accidental redundancy can stem from intricacies of the observed object. In either case, the problem of identifying which of the attributes are redundant in the task at hand is generally known as *variable selection* [43], *variable subset selection* [108], *attribute selection*[44], *state abstraction* [2], [72] or *feature selection* [63], [80], [25] problem (as opposed to the *feature extraction* [113] problem). This issue is strongly related to the *curse of dimensionality* [7]: when the number of input dimensions increases linearly, algorithm's data, memory and/or computational complexity grow exponentially.

Biology and psychology have often been a source of inspiration for successful Artificial Intelligence (AI) methods, Artificial Neural Networks, in their origins, being the most notable example. Reinforcement Learning (RL) [112] is another example of a biologically inspired AI approach, with some success in solving complex tasks [104]. Typically, Markov Decision Processes (MDP) [6] are used to analyze RL algorithms. In the MDP context, the given observations form a space of variables, called also the state space. Thus, the *variable selection* problem consists in choosing which of the available variables are redundant. The main difference from general *variable selection* problems is that the input data represents a trajectory of a discrete time stochastic control process, as opposed to time-invariant data samples.

In this work we present an algorithm that solves the *variable selection* problem in the MDP

context. It is inspired by two aspects of the *need for closure* psychological phenomenon [48], namely the aversion towards *ambiguity* and the *cognitive dissonance* [69], which manifest in human behaviour [99]. We use *state abstraction* as the main paradigm of this work. In general, *state abstraction* can be thought of like creating an alternative, in some sense simpler, space of variables, for the same task, which in turn yields an alternative MDP, which preserves some properties of the original MDP. *Variable selection* is thus a specific form of *state abstraction*.

In RL we are given a *reinforcements mechanism* in a form of a reward/penalty function designed for the purpose of a particular RL task. Under the assumption that we have been given a set of variables, some of them being redundant, the proposed algorithm identifies the non-redundant ones incrementally. It starts with an empty set of variables and successively refines the abstraction, using the introduced notion of *ambiguity* to find the solution to the discussed problem. This is referred to as the *bottom-up* approach (as opposed to the *top-down* approach - starting with all variables and removing them one by one). The subsequent refinements lead to a minimal set of variables that is sufficient for the resulting MDP model to induce the same behaviour in terms of the *reinforcements mechanism* as the given one. Such abstraction is called *model-preserving* abstraction. The presented approach is designed in a way that allows easy application to continuous domains, which is necessary to be suitable to real-world problems.

The main thesis of this work is therefore:

our ambiguity-based, bottom-up approach is a valid solution of the Reinforcement Learning variable selection problem

This thesis is supported by the following sub-theses:

1. the problem of finding the model-preserving variable selection abstraction is an inapproximable NP-hard problem

2. the proposed notion of ambiguity correctly reflects the quality of a solution of a variable selection task

3. the proposed approach, based on the notion of ambiguity correctly solves variable selection tasks for RL, including tasks with continuous transition and reinforcement functions

To verify these sub-theses, this dissertation is organized as follows. Chapter 3 overviews the theoretical background and the related work. It introduces the most basic notions, including

state and *state abstraction*, along with the relevant notation. Also, it presents the most important, from this work's point of view, parts of current research. Section 4.3 of the subsequent Chapt. 4 contains the proof of the first sub-thesis, which legitimizes devising a heuristic algorithm, based on the proposed framework. Section 4.4 of this chapter introduces core notions of this thesis, namely *ambiguity* and *ambiguity functional*. In the following Sec. 4.5 we additionally argue that starting from an empty set of variables and inserting them incrementally is generally better than the opposite. The following Chapt. 5 presents the proposed approach in the context of RL domains with discrete sets of states. The next chapter 6 extends this reasoning to continuous domains. The second sub-thesis supports the main thesis by showing that the notion of *ambiguity* used in the proposed approach works as intended and it is verified experimentally throughout the Chapters 5 and 6. The last sub-thesis directly supports the main claim, by verifying the proposed approach with computer simulations in Chapt. 7. Chapter 8 presents the conclusion, critical overview of the presented approach and indicates directions of possible future work. Further chapter is the appendix that describes technical details of this study.

1.1 Psychological context

The main idea of this work is inspired by the notion of *ambiguity*. *Ambiguous* situations are defined as situations involving uncertainty, when the probabilities of outcomes of different actions are unknown [18]. Such situations induce the feeling of discomfort called the *cognitive dissonance* [69], [99]. This is connected with the *aversion towards ambiguity*, which stems from the fact that ambiguous situations are being perceived as a source of threat [39]. This in turn makes people motivated to reach for a *closure*, that is, an explanation of the ambiguous situation [48]. Additional motivation comes from the fact that such information helps in the ability to predict the surrounding world [69]. Simple explanations are preferred over the complicated ones [95] and they incrementally improve individual's mental model of the surrounding world [11].

This is on the other hand closely related to the research on *stimulus discrimination* [102] problem or the *selective observation* problem [102]. Animals, when handling simple tasks are able to distinguish only a subset of signals, sufficient for a particular task, from the surrounding environment. However, in more complicated tasks, when there are too many variables that seem to be valid discriminators [102] (i.e. variables that allow to distinguish between some two events) a problem similar to the *curse of dimensionality* arises. Similarly, in the context of

variable selection, when too few data samples are presented, any variable can be chosen as a valid discriminator. In this work we propose a method to deal with this issue in the context of Markov Decision Processes in a way inspired by mechanisms described in this section.

The principle of the algorithm presented in this work is related to these mechanisms. At each step, the algorithm has a list of variables that are assumed to be sufficient to model the observed environment. When confronted with a set of transitions that contradict that, i.e. making the surrounding world, the environment, appear nondeterministic/ambiguous, it seeks for a simple explanation of this fact. The simplest one is to assume that one of the currently ignored signals is in fact non-redundant.

Chapter 2

Test environments

In this chapter we introduce test environments (also called *domains*, *tasks* or *plants*), which are used throughout this work in examples and in experimental evaluation of discussed algorithms. We start with a simple deterministic, discrete environment, namely the *Discrete Labyrinth*, which illustrates presented ideas in the discrete state domain context. The next presented domain is a simple stochastic, discrete environment, which is used to demonstrate that even though the algorithm presented in this work is developed in a deterministic context, it can work for stochastic plants. Then we introduce more challenging tasks: the *Continuous Labyrinth*, which is a continuous state version of the *Discrete Labyrinth*, and then, the *Cart-Pole Swing-Up*, the *Mountain Car* and the *Pinball* - slightly more complex (in terms of nonlinearity/stochasticity) environments.

2.1 The Discrete Labyrinth

This environment is a simple *grid-world* with ten rows and ten columns, where the agent's goal is to reach a distinguished square. The labyrinth contains one wall, over which the agent can not pass and a target spot, which, when reached gives a positive reinforcement and ends the episode. A graphical interpretation of this environment is presented in Fig. 2.1. Plant's output consists of the following integer measurables: x, y denoting agent's horizontal and vertical position within the labyrinth, reps., and the following functions of these measurables: $s = x + y, d = |x - y|, m = xy, q_x = \lfloor \sqrt{x} \rfloor, q_y = \lfloor \sqrt{y} \rfloor$. This results in a 9-dimensional initial space of measurables ($\mathfrak{X}$). Plant's control space consists of the following actions: $\mathcal{U} = \{LEFT, UP, RIGHT, DOWN\}$. The transition function f_x changes agent's coordinates to move it one square according to the

appropriate direction denoted by action's name with one exception: should the agent be next to the wall (side of the labyrinth, or the wall within the labyrinth, presented in Fig. 2.1) and choose an action that attempts to move through the wall, the action has no effect. The reinforcement function f_r gives -1 for ordinary move, -5 for attempting to cross the wall and 10 for reaching the target square, placed at $(x = 8, y = 8)$. Visualization of this environment is presented in Fig. 2.1.

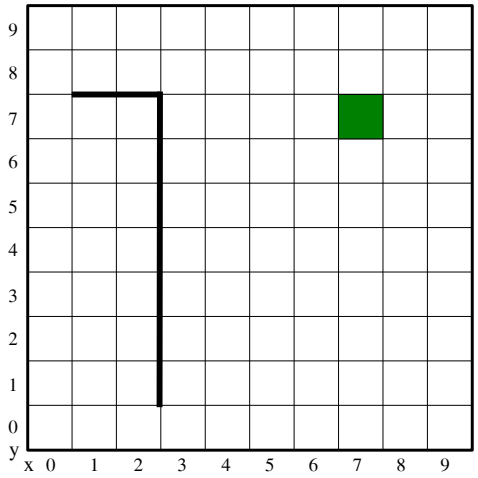

Figure 2.1: The Discrete/Continuous Labyrinth environment. The thick line represents a wall over which the agent can not pass, and receives -5 penalty if tries to do that. The green square at position $(X = 8, Y = 8)$ is the target point, giving a reinforcement with value equal to 10 end ending an episode when reached.

2.2 The Coffee Task

The Coffee Task [9] problem is very common in RL literature, but to the knowledge of the authors, it is not precisely defined. In general, agent's goal is to execute a correct sequence of steps, which lead to a robot delivering a coffee, without getting wet, if it rains. For the concrete definition, in this work we assume what follows.

Plant's state consists of the following binary variables: SH, SC, SR, SU, SL, SW, with the following meaning: $SH = \{\text{the user has the coffee}\}$; $SC = \{\text{the robot has the coffee}\}$; $SR = \{\text{it is raining}\}$; $SU = \{\text{the robot has the umbrella}\}$; $SL = \{\text{the robot is in the office}\}$; $SW = \{\text{the robot is wet}\}$. The boolean values of measurables are encoded as 1 for *true* and 0 for *false*. Plant's control space consists of the following actions: $\mathcal{U} = \{GO, BUY, GIVE, TAKE\}$. The transition function f_x for the Coffee Task is presented in Tab. 2.1. The state is terminal

Table 2.1: Transition function f_x for the Coffee Task problem.

	$u(t)$	
$\mathbf{x(t+1)}$	GO	BUY
$SH(t+1)$	$SH(t)$	$SH(t)$
$SW(t+1)$	$\begin{cases} SW(t) \vee (\neg SU(t) \wedge SR(t)) & SL(t+1) \neq SL(t) \\ SW(t) & otherwise \end{cases}$	$SW(t)$
$SC(t+1)$	$SC(t)$	$SC(t) \vee \neg SL(t)$
$SR(t+1)$	`true` w.p. 0.5	`true` w.p. 0.5
$SU(t+1)$	$SU(t)$	$SU(t)$
$SL(t+1)$	`not` $SL(t)$ w.p. 0.95	$SL(t)$
	$u(t)$	
$\mathbf{x(t+1)}$	GIVE	TAKE
$SH(t+1)$	$\begin{cases} \texttt{true} & SH(t) \\ \texttt{true w.p. 0.95} & SL(t) \wedge SC(t) \end{cases}$	$SH(t)$
$SW(t+1)$	$SW(t)$	$SW(t)$
$SC(t+1)$	$SC(t)$	$SC(t)$
$SR(t+1)$	`true` w.p. 0.5	`true` w.p. 0.5
$SU(t+1)$	$SU(t)$	$\begin{cases} \texttt{true} & SU(t) \\ \texttt{true w.p. 0.95} & SL(t) \end{cases}$
$SL(t+1)$	$SL(t)$	$SL(t)$

$\iff SH$.

We assume we are given four additional boolean functions of the base 6 measurables, namely: $S6 = \neg SR \vee SU$; $S7 = \neg SC \wedge SL \wedge SW$; $S8 = SU \vee \neg SL \vee SW$; $S9 = SU$. Thus, plant's output is represented by the following space of measurables $\mathcal{X} = \{$ SH, SW, SC, SR, SU, SL, $S6$, $S7$, $S8$, $S9$ $\}$. It is possible in practice, that some of plant's state variables may be measured more than once - hence the $S9 = SU$ duplication in this example.

2.3 The Continuous Labyrinth

This environment is an intermediate step between discrete and continuous domains. While still being a simple labyrinth it introduces dynamics, as agent's actions do not modify its position directly, but only its velocity. The labyrinth's shape is the same as in the Discrete Labyrinth task, with one target spot, but the agent's and goal's positions are a tuple of real numbers instead of integers. Plant's output consists of the following real valued measurables: $\mathcal{X} = \{x, y, v_x, v_y\}$, denoting agent's position within the labyrinth and its velocity along both axes. Plant's control space consists of the following actions: $\mathcal{U} = \{LEFT, UP, RIGHT, DOWN\}$. The transition function f_x changes agent's velocity, increasing it by 0.1 in the appropriate direction denoted by action's name. Additionally, after each step agent's velocities are multiplied by a factor of 0.3 and clamped to the range $[-1, -1]$. Similarly to the Discrete Labyrinth, when the agent hits the wall effects of the executed action are rolled back. Additionally, agent's velocities are set to 0. The reinforcement function behaves in the same way as in the Discrete Labyrinth task: f_r gives -1 for ordinary move, -5 for attempting to cross the wall and 10 for reaching the target square, placed at $(x = 8, y = 8)$. Visualization of this environment is the same as for the Discrete Labyrinth, and is presented in Fig. 2.1. The transition function modifies agent's state

as follows, assuming the agent will not hit the wall:

$$v_x(t+1) = \begin{cases} \frac{v_x(t)+0.1}{3} & u(t) = \text{RIGHT} \\ \frac{v_x(t)-0.1}{3} & u(t) = \text{LEFT} \\ \frac{v_x(t)}{3} & u(t) \in \{\text{UP}, \text{DOWN}\} \end{cases}$$

$$v_y(t+1) = \begin{cases} \frac{v_y(t)+0.1}{3} & u(t) = \text{UP} \\ \frac{v_y(t)-0.1}{3} & u(t) = \text{DOWN} \\ \frac{v_y(t)}{3} & u(t) \in \{\text{LEFT}, \text{RIGHT}\} \end{cases}$$

$$x(t+1) = x(t) + \begin{cases} v_x(t)+0.1 & u(t) = \text{RIGHT} \\ v_x(t)-0.1 & u(t) = \text{LEFT} \\ v_x(t) & u(t) \in \{\text{UP}, \text{DOWN}\} \end{cases}$$

$$y(t+1) = y(t) + \begin{cases} v_y(t)+0.1 & u(t) = \text{UP} \\ v_y(t)-0.1 & u(t) = \text{DOWN} \\ v_y(t) & u(t) \in \{\text{LEFT}, \text{RIGHT}\} \end{cases}$$

In case of hitting the wall, the transition function behaves the following way:

$$\begin{aligned} v_x(t+1) &= 0 \\ v_y(t+1) &= 0 \\ x(t+1) &= x(t) \\ y(t+1) &= y(t) \end{aligned}$$

The equations above omit clamping the values to the predefined ranges, i.e. velocities to $[-1,1]$ and coordinates to $[0,10]$, for simplicity.

Similarly to the Discrete Labyrinth task, the following five additional measurables are introduced: $s = x+y, d = |x-y|, m = xy, q_x = \sqrt{x}, q_y = \sqrt{y}$. This results in a 9-dimensional space of measurables ($\mathfrak{X}$).

2.4 The Cart-Pole Swing-Up

In this environment, introduced by Barto, Sutton and Anderson [5], agent's goal is to balance a pole mounted to a cart so it maintains an upright position. The Cart-Pole Swing-Up's output

consists of: cart's position $x \in [-2.4, 2.4]$, its velocity $\dot{x} \in (-\infty, \infty)$, pole's angle $\theta \in [0, 2\pi)$ (0 when the pole points straight up) and pole's angular velocity $\dot{\theta} \in (-\infty, \infty)$. The plant is controlled by a force $u \in \mathcal{U} = \{-5, -0.1, 5, 0.1\}$. An episode ends if $|x_t| > 2.4$. The plant is presented in Fig. 2.2. Assuming denotation: x for cart's position, θ for pole's angle, m_c, m_p

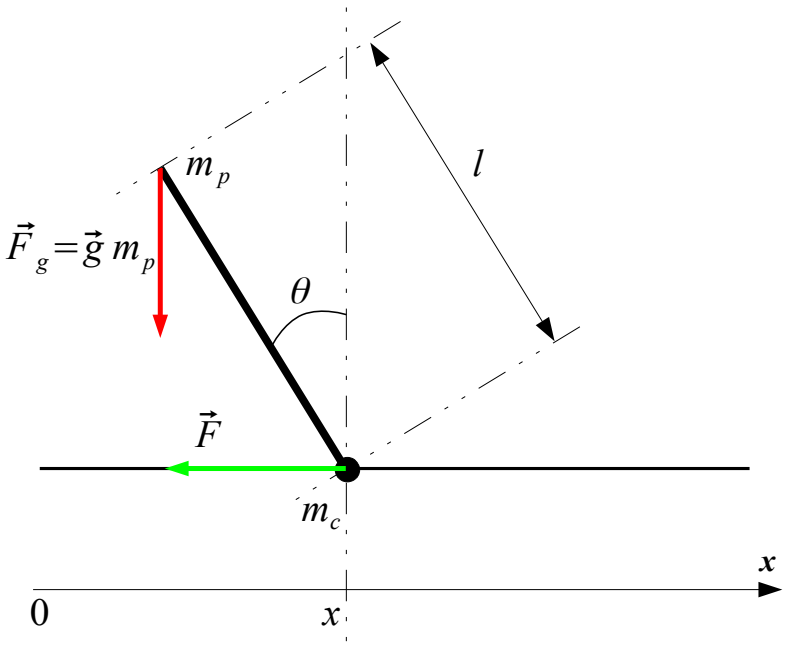

Figure 2.2: The Cart-Pole Swing-Up environment. The thickest black line represents the pole to be balanced. The horizontal line is the rail, along which the cart is moving.

for the masses of cart and pole resp., l for pole's length, μ_c, μ_p for the cart-track and cart-pole friction factors resp. and g for gravitational constant, the state equations for this environment have the form [5]:

$$\ddot{x} = \frac{u + m_p l \left[\dot{\theta}^2 \sin\theta - \ddot{\theta}\cos\theta\right] - \mu_c \operatorname{sgn}\dot{x}}{m_c + m_p}$$

$$\ddot{\theta} = \frac{g\sin\theta + \cos\theta\left[\frac{-u - m_p l \dot{\theta}^2 \sin\theta + \mu_c \operatorname{sgn}\dot{x}}{m_c + m_p}\right] - \frac{\mu_p \dot{\theta}}{m_p l}}{l\left[\frac{4}{3} - \frac{m_p \cos^2\theta}{m_c + m_p}\right]}$$

This is the formulation that corresponds to the definition of this problem presented in [5]. While we are aware of the error in the state equations ([32]), we chose to use this version as it became common in RL related literature.

We assume that we are given the following set of measurables (with some measurables

having the same denotation as state variables):

$$x = x$$
$$v = \dot{x}$$
$$s = \sin\theta$$
$$c = \cos\theta$$
$$a = \dot{\theta}$$
$$h = x + 2\dot{x}$$
$$i = \sqrt{|\dot{x}|}$$
$$j = s + c$$
$$k = \tan\theta$$
$$l = \dot{\theta} + sc - x$$

This set of measurables is redundant, because five-element sets exist that form a correct solution (e.g. (x, v, s, c, a) or (x, s, j, h, l)). This redundancy will allow to demonstrate the capabilities of the presented algorithm. Again, assume that these are the variables initially identified by an engineer as the potential state variables, with no explicit knowledge of the transition/reinforcement functions. In particular, even though the explicit information about the pole's angle is available through the values of θ, we assume that we incorrectly identified it as two quantities instead: $s = sin(\theta)$ and $c = cos(\theta)$.

To illustrate the behavior of our algorithm we test it against two versions of this environment. The first employs a simple reinforcement function:

$$r(t) = \cos\theta(t) \tag{2.1}$$

We call this version the *Cart-Pole Swing-Up with a simple reinforcement function*. The second, more sophisticated version of the reinforcement function is defined as:

$$r(t) = \begin{cases} \cos\theta(t) & \text{if } |a| \leq 2\pi \text{ and } |x(t)| \leq 2.4 \\ -1 & \text{if } |a| > 2\pi \text{ and } |x(t)| \leq 2.4 \\ -100 & \text{otherwise} \end{cases} \tag{2.2}$$

We refer to this version as *Cart-Pole Swing-Up with a complex reinforcement function*.

2.5 The Mountain Car

The Mountain Car environment [83], [105] consists of an underpowered car on a hill. Its goal is to use one slope to gain enough speed for reaching the goal position on the top of the other

slope. This domain consists of the following state variables: car's position $x \in [-1.2, 0.6]$ and its velocity $v \in [-0.07, 0.07]$. The agent can control the car's acceleration with an action $u \in \{-0.001, 0, 0.001\}$. An episode ends iff $x \geq 0.5$. We consider two versions of this environment: the *simple Mountain Car* and the *stochastic Mountain Car*. The state equations for the first version of this environment have the following form:

$$\dot{x} = v$$
$$\dot{v} = u - 0.0025 \cos 3x$$

The state equations for the stochastic version are as follows:

$$\dot{x} = v$$
$$\dot{v} = U(-0.0005, 0.0005) + u - 0.0025 \cos 3x$$

where $U(a,b)$ denotes a uniform distribution in the interval $[a,b]$.

The reinforcement function is defined as:

$$r(t) = \begin{cases} -0.1 & x < 0.5 \\ 0.0 & x \geq 0.5 \end{cases}$$

The plant is presented in Fig. 2.3. We assume that we are given the following set of measurables

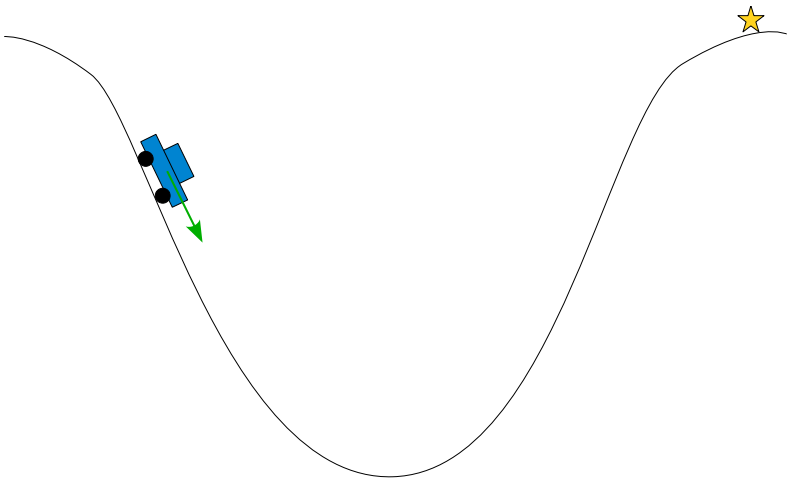

Figure 2.3: The Mountain Car environment. The yellow star represents the goal, where x = 0.5, the agent receives the highest reinforcement, and episode ends.

(some of them being denoted with the same name as the underlying state variable):

$$
\begin{aligned}
x &= x \\
v &= v \\
F_3 &= x + v \\
F_4 &= |xv| \\
F_5 &= \cos x \\
F_6 &= \cos v \\
F_7 &= x - 2v \\
F_8 &= xv \\
F_9 &= x^2 \\
F_{10} &= v^2 \\
F_{11} &= \sqrt{|x|} \\
F_{12} &= \sqrt{|v|} \\
F_{13} &= \sin x \cos v \\
F_{14} &= \tfrac{x}{100} \\
F_{15} &= \tfrac{v}{100} \\
F_{16} &= 100x \\
F_{17} &= 100v \\
F_{18} &= |x| - |v| \\
F_{19} &= \sin x - \cos v \\
F_{20} &= v - 2x
\end{aligned}
$$

2.6 The Pinball

The Pinball domain [65] is similar to the Continuous Labyrinth environment, being essentially a two-dimensional labyrinth with continuous transition function and a target spot for the agent to reach. It is however much more complicated, because there are more, irregular, obstacles and collisions with them are elastic, causing the agent to bounce. The plant is visualized in Fig. 2.4. The state is represented by four variables: two-dimensional position $x \in [0,1]$ and $y \in [0,1]$ and two-dimensional velocity $\dot{x} \in [-1,1]$ and $\dot{y} \in [-1,1]$. There are four possible actions, represented by two-element tuples with each element denoting agent's influence on $\dot{x}$ and $\dot{y}$ variables respectively: $u \in \{(0.2,0),(0,0.2),(-0.2,0),(0,-0.2)\}$. We omit the state equations for this domain because they are very complicated, mainly due to elastic collisions.

Its implementation is available in the *dotRL* platform [93] or at the web site [66] associated with the publication it was introduced in [65].

Because in this environment there are only two, fixed starting points, away from the target spot, it is virtually impossible for an agent that randomly selects its actions to reach it. Because of that, we define the reinforcement function for this environment as follows, t_x, t_y being target spot's position:

$$r(t) = \sqrt{(x - t_x)^2 + (y - t_y)^2}$$

We assume that we are given the following set of measurables (with some of them being

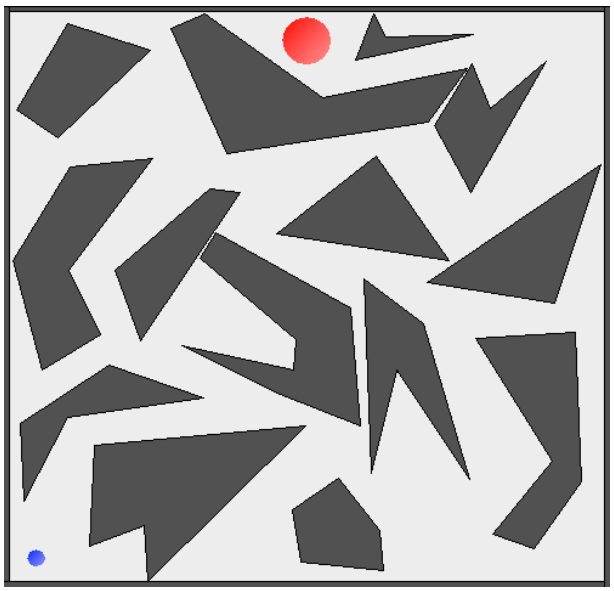

Figure 2.4: The Pinball environment. The red circle represents the goal position, where the agent receives the highest reinforcement and episode ends. The agent is represented by the blue circle. Dark-gray shapes are obstacles. The picture was taken from the web site [66] associated with the publication that introduces this environment [65].

denoted with the same name as the underlying state variable):

$$x = x$$
$$y = y$$
$$v_x = \dot{x}$$
$$v_y = \dot{y}$$
$$F_5 = xy$$
$$F_6 = x + y$$
$$F_7 = \dot{x}\dot{y}$$
$$F_8 = \dot{y} + \dot{y}$$
$$F_9 = x\dot{x}$$
$$F_{10} = y\dot{y}$$

Chapter 3

State of the art in state abstraction in Reinforcement Learning

The main idea of this work is expressed in the *state abstraction* paradigm. State abstraction deals with simplifying the given space of variables (in our case: measurables). In this work, we focus on a particular type of state abstraction, namely the *state abstraction by variable selection*, applied to the Reinforcement Learning paradigm.

Reinforcement Learning is a learning framework that consists of two interacting entities, namely an *agent* and an *environment*. The agent can influence the environment using *actions* (or *controls*), perceives its *state*, and receives a *reinforcement*. The notions of the state and the reinforcement are defined in Sec. 3.1 and Sec. 3.2 of this chapter. Agent's goal is to learn how to optimize the received reinforcements, according to some criterion, only by interacting with the environment. The way the agent chooses its actions is called the *control policy* or just *policy*. The optimal control policy guarantees that the agent will receive the best possible (in the context of some optimality criterion) reinforcements. This setting is presented in Fig. 3.1.

Current research in the area of *state abstraction* in the RL context encompasses various properties for grouping the vectors of measurables. One part of this research consists of methods that aim to find a suitable *state abstraction* for a simple task, representing a wider group of similar tasks, but for which an optimal control policy is already known. This is done under the assumption that such result will be useful for learning algorithms in a context of much more complicated, but structurally similar tasks. This approach is known as the *knowledge transfer* approach [67]. The other aspect of the *state abstraction* research aims to determine a *state abstraction* without knowing the optimal policy beforehand. This work focuses on this second

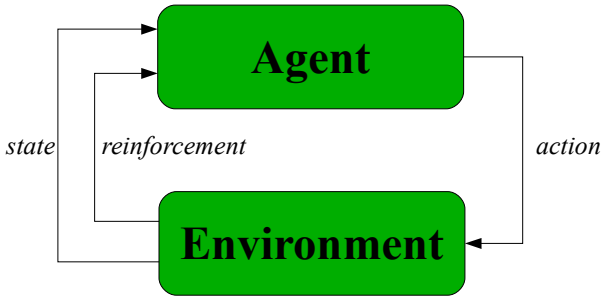

Figure 3.1: The general Reinforcement Learning paradigm. An agent interacts with an environment, influencing it with actions and perceiving its state and reinforcement values.

aspect of *state abstraction* research.

Although our work concerns *state abstraction* for Markov decision processes (MDP), which are a typical framework for analysis of the RL algorithms, the main idea can be easily seen in a more general context, not attached to either of these frameworks. The experiments are carried out in the RL context.

In this chapter, we start by explaining how the term *state* is understood in this work (Sec. 3.1). Then we present our notation and cover the definition of *state abstraction* in Secs. 3.2 and 3.3. Sections 3.5 and 3.4 narrow the view to a particular, special case of *state abstraction*, which we wish to focus on. Finally, in Sec. 3.7 refers to current research.

3.1 State

State abstraction can be thought like a way of grouping together *states* that share some property. The term *state* yet needs to be fixed, since it can be interpreted twofold: from the RL perspective or from the Control Theory perspective.

The first, Reinforcement Learning perspective is based mainly on the Markov Decision Processes (MDP) framework, in which the considered MDP model consists of state and action spaces and transition and reinforcement distributions. Thus, the *state* in RL denotes values of variables of the state space of the considered MDP model [6], [52], [97], [8]. An extension of this framework - the Partially Observable Markov Decision Processes (POMDP) introduces the notion of *observations* as potentially incomplete and indirect information about the hidden *state*. In the second perspective, namely the Control Theory perspective, the *state* denotes a set of values of *state variables* of the underlying system's description in the *state-variable form*

[60], [90], [13], [35]. In this meaning, the set of *state variables* is the minimum possible set of variables that is sufficient to describe *system's behaviour* ("behaviour of the system" [60], "evolution of the system" [35]). System's behaviour can be defined as what is perceived through *observations* available to the system's operator.

We arrange these two perspectives as follows. The general RL setting presented in Fig. 3.1 is suitable for decision problems. In our case, to consider a dynamical system control scenario, the environment block can be split into a block representing the unknown system we wish to control and a sensor or measurements block, which defines how can we perceive the system. Additionally, we extend the diagram with the "abstraction selector" block that produces an abstract representation of given measurements. The *state* in the RL context, i.e. the output of the whole "Environment" block in Fig. 3.1, becomes *observations* or *measurements* in Fig. 3.2. The *underlying state* consists of values of the *state variables* in the Control Theory sense, defined in the previous paragraph. The *vector of measurables*, $\mathbf{x}$ is a vector that consists of values of all given variables called *measurables*. For brevity, sometimes we use the term *measurement* instead of saying "vector of measurables". The set of measurables can be redundant, i.e. contain more variables than needed to capture all the information about the *underlying state*, and thus sufficient to determine system's evolution. The role of the abstraction selector is to produce a minimal representation that does not loose any of this information. We call the result of the abstraction selector the *abstract state*. This work is thus focused on constructing the abstraction selector block. Denotations used in Fig. 3.2 are defined in the following sections.

In general, the selection of particular *state variables* to mathematically describe a plant is ambiguous - even though in practice it is intuitive [35]. This in turn makes the choice of *measurables* ambiguous as well. The simplest solution is to define the system's behaviour as the values of all available measurables. The aforementioned Fig. 3.2 presents this scenario. A more extended approach would introduce some hierarchy of the measurables, similarly to the stimuli hierarchy investigated in psychology (e.g. [34]). Another possibility is to define system's behaviour basing on a single specific observation. In such scenario this specific observation is the only important stimulus. In Reinforcement Learning framework such signal is called the *reinforcement*. In this context system's behaviour is defined by values of the reinforcement function, as that is the only signal which influences the quality evaluation of learning algorithms [112]. In this work we focus on this scenario (Fig. 3.3). All examples and experiments refer to particular measurables using their names, indicating their conceptual meaning, e.g. x - verti-

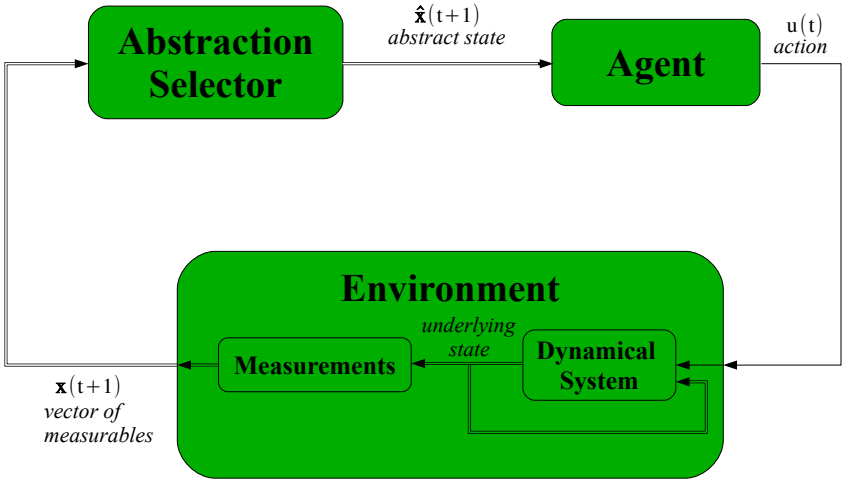

Figure 3.2: The extended Reinforcement Learning paradigm. An agent interacts with an environment that consists of an unknown system and a set of sensors. The abstraction block creates a minimal representation of measurables that contains all information about system's state. Agent can interact with the system with actions ($u(t)$) and can perceive environment's abstract states ($\hat{\mathbf{x}}(t+1)$). t denotes discrete time. This work is focused on the goal of implementing the abstraction selector block.

cal position or v - velocity. In some examples, when there are many measurables that have no particular interpretation we use the notation $F_{\square}$ with measurable's ordinal number for $\square$ (F is referring either to *fluent* [40], or *feature* [53]). Measurables which are simple functions of other measurables are denoted explicitly, e.g. $x+y$.

3.2 Basic definitions

Because our work concerns only discrete time [60] systems, terms *discrete* and *continuous* are unambiguously used throughout this work to denote quantized and continuous transition functions resp. With a space of measurables $\mathcal{X} \subseteq \mathcal{R}^n$, a control space $\mathcal{U} \subseteq \mathcal{N}$ the transition function $f_x : \mathcal{X} \times \mathcal{U} \longrightarrow \mathcal{X}$ and reinforcement function $f_r : \mathcal{X} \times \mathcal{U} \longrightarrow \mathcal{R}$, we define a continuous or discrete (when $\mathcal{X} \subseteq \mathcal{N}$), deterministic model as a tuple $\langle \mathcal{X}, \mathcal{U}, f_x, f_r \rangle$. This model is a combination of commonly used continuous space MDPs [45], [116] and deterministic MDPs [74], which can be easily generalized to a stochastic version. We denote the measurement vector that consists of values of measurables at time $t \in \mathcal{N}$ by $\mathbf{x(t)}$ (the *vector of measurables*), the control applied at time t by $u(t)$ and the reinforcement received by $r(t)$. In most cases however, we either con-

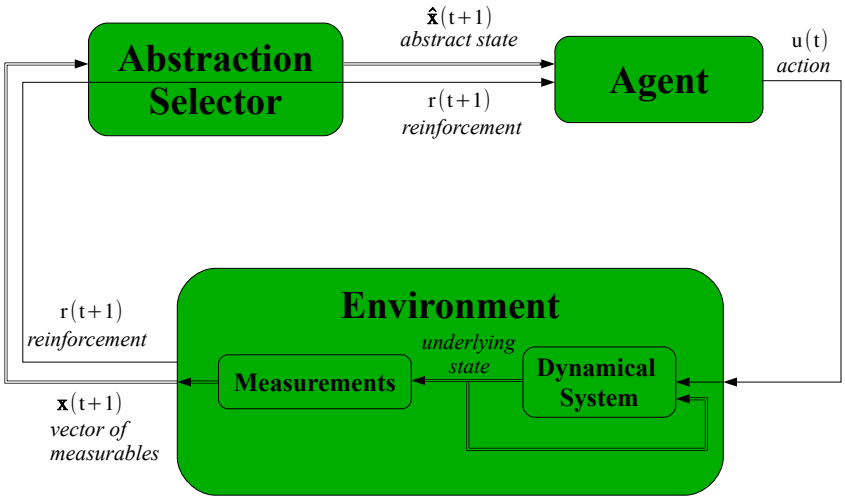

Figure 3.3: The extended Reinforcement Learning paradigm. An agent interacts with an environment that consists of an unknown system and a set of sensors. The abstraction block creates a minimal representation of measurables that contains all information about system's state. Agent can interact with the system with actions ($u(t)$) and can perceive environment's abstract states ($\hat{\mathbf{x}}(t+1)$) and reinforcements ($r(t+1)$). t denotes discrete time. This work is focused on this paradigm and on constructing the abstraction selector block.

sider only one time instant, or two consecutive time instants, with their ordering being clear, and thus we skip the time index (t) for brevity. The *space of measurables* is a space formed by all given *measurables* (refer to the previous discussion in Sec. 3.1). Similarly to MDP analysis, we assume that the *space of measurables* $\mathcal{X}$ and *control space* $\mathcal{U}$) are known and the transition function f_x and the reinforcement function f_r) are unknown. The measurables can be seen as the *output* of the system [60], [62], [35]. The assumption that the measurables contain complete information about system's state (see Sec. 3.1) implies that the system is *observable* [35].

The problem considered is to find which measurables can be seen as the *state variables* forming the *state space* of the underlying system in terms of its behaviour, assuming that the given model has more measurables than needed. The behaviour is defined in this work as the sequence of values of the reinforcement function f_r. This quantity can be one of measurables as well.

It is possible that the given set of measurables is insufficient to describe system's behaviour. However, we consider such situation an ill-posed problem and focus on the scenario when the given measurables are sufficient and redundant. Usually, the set of measurables sufficient to describe system's behaviour is not unique - in such case we accept any possible solution. In the

context of a particular solution, measurables not included in the solution are called *redundant variables* [35] or *redundant measurables*, just *redundant* or simply *ignored*. The other ones are called *preserved*.

In practice, $\mathfrak{X}$ can be interpreted as engineer's intuition on what quantities could possibly be important, when the goal is to discover an abstract space that is the minimal space (in terms of the number of measurables), but sufficient to describe the system's behaviour. Any such abstract space is called a correct solution. This goal is achieved by testing subsequent hypotheses of the form: $\hat{\mathfrak{X}}$ is a correct solution. Abstract spaces are defined in the following section.

The algorithm presented in this dissertation works by means of processing a set of *samples* obtained by observing how an agent interacts with a plant. A *sample* is defined as a tuple: $\langle \hat{\mathbf{x}}(t), u(t), \hat{\mathbf{x}}(t+1), r(t+1) \rangle$, which represents one step of agent's interaction with a plant: i) observed abstract state is $\hat{\mathbf{x}}(t)$ ii) agent executes the action $u(t)$ iii) observed abstract state changes to $\hat{\mathbf{x}}(t+1)$ iv) reinforcement value $r(t)$ is observed.

Two cases of data sources can be considered, regarding the source of *samples*:

- simulated data: samples generated on demand, starting from particular points of the space of measurables, allowing the algorithm to quickly obtain the information necessary to calculate all required quantities
- "real" data: a random set of samples with random actions, forcing the algorithm to wait for some particular combinations of abstract state/action to occur

The first case assumes that we have an access to a "generator" that contains a numerical model of the plant, and allows us to generate samples on demand. This assumption is referred to as the *simulator assumption*. This implies that the f_x and f_r are known (either in analytical or at least numerical form, because they are somehow employed by the given simulator). This is often not the case in practice. However, it is useful to consider it because we can quickly verify if a particular version of the algorithm would work, if it had been given all the required data straightaway. Note also that this assumption is sometimes possible to be met in real-life applications. For example, in gas network control, very accurate simulators exist ([120], [106]), but still it is hard for autonomous agents to learn to control the plant.

The "real" data case is when the algorithm can not influence what samples it gets, which relates to a wider range of practical situations.

In Chapters 4, 6 we first evaluate the proposed ideas using the simulator assumption. Then, in Chapt. 7 we present results with purely random sample of state transitions.

Example 3.2.1. When analyzing a plant we start with either knowledge of its internals, or with intuition only. For the Cart-Pole Swing-Up plant, we could infer, knowing laws of physics, that the behaviour related to pole's angle and movement depends on cart's position, its velocity, pole's angle and pole's angular velocity. Thus, we consider some measurables as functions of these quantities, e.g. $s = sin(\theta)$ (θ being pole's angle) which form the initial set of variables that describes the $\mathcal{X}$ space in the factored form. The goal is to discover which subset of these features is minimal and sufficient to make possible to learn to stabilize the pole in its up-vertical position ($f_r = cos(\theta)$).

□

3.3 General form of state abstraction

In this section we provide theoretical background related to state abstraction. This short introduction is a basis for Secs. 3.4 and 3.5, where we define particular types of abstractions we focus on in this work, and their properties.

To define what the state abstraction is, we first recall some basic properties of equivalence relations [119]. Let $\sim$ be an equivalence relation on a $\mathcal{S}$. Denote an equivalence class of an element $e \in \mathcal{S}$ by

$$[e]_{\sim} = \left\{e' \in \mathcal{S} : e \sim e'\right\} \tag{3.1}$$

The quotient set $\mathcal{S}/\sim$ is defined as a set of equivalence classes of $\sim$ on $\mathcal{S}$:

$$\mathcal{S}/\sim = \{[e]_{\sim} : e \in \mathcal{S}\} \tag{3.2}$$

These equivalence classes arrange elements of $\mathcal{S}$ into disjoint sets. For each of these sets we can choose an arbitrary element as its *representative*. Denote by $\hat{e} \in [e]_{\sim}$ a representative of the equivalence class $[e]_{\sim}$. We call each *representative* an *abstract element*, and the set:

$$\hat{\mathcal{S}}_{\sim} = \{\hat{e} : e \in \mathcal{S}/\sim\} \tag{3.3}$$

the *abstract set*. With a little abuse of notation, we also say $[\hat{e}]$ to denote the equivalence class of the representative $\hat{e}$.

The same reasoning can be applied to the space of measurables $\mathcal{X}$. We form the set of equivalence classes, $\mathcal{X}/\sim$ (the quotient set), for some equivalence relation $\sim$. A set of representatives of all equivalence classes from $\mathcal{X}/\sim$ forms a space $\hat{\mathcal{X}}_{\sim}$ called the *abstract space* [3] (the *abstracted space* [2], the *abstraction space* [2]).

There is a link between the elements of $\mathfrak{X}$ and the abstract space $\hat{\mathfrak{X}}_{\sim}$: each equivalence class from $\mathfrak{X}$ has a representative in $\hat{\mathfrak{X}}_{\sim}$. This can be represented by a mapping defined as ([72]):

$$\begin{aligned} \phi &: \mathfrak{X} \longrightarrow \hat{\mathfrak{X}}_{\sim} \\ \phi(\mathbf{x}) &= \hat{\mathbf{x}} \end{aligned} \tag{3.4}$$

This mapping is visualized in Fig. 3.4. Throughout this work, quotient spaces are not needed in any derivations, so for simplicity we omit them in all figures. We identify equivalence classes with abstract states and denote them with blue shapes that aggregate measurements directly in the $\mathfrak{X}$. This simplified concept of state abstraction is presented in Fig. 3.5.

We say that the abstract space $\hat{\mathfrak{X}}_{\sim}$ is *implied* by the abstraction function ϕ. The $\sim$ index is usually omitted, since in the context of a particular abstraction function ϕ only one equivalence relation is relevant, namely the one that defines this abstraction. Each *abstract state* $\hat{\mathbf{x}} \in \hat{\mathfrak{X}}$ conceptually represents one or more *vectors of measurables* of the original space $\mathfrak{X}$, where the equivalence class $[\hat{\mathbf{x}}] \subseteq \mathfrak{X}$ is represented by $\hat{\mathbf{x}} \in \hat{\mathfrak{X}}$.

The inverse mapping ϕ^{-1} returns a set of vectors that are represented by a given abstract state:

$$\begin{aligned} \phi^{-1} &: \hat{\mathfrak{X}} \longrightarrow 2^{\mathfrak{X}} \\ \phi^{-1}(\hat{\mathbf{x}}) &= [\hat{\mathbf{x}}] \end{aligned} \tag{3.5}$$

Abstraction mappings can be compared by analyzing the subsets in the quotient space they induce. For two abstraction mappings ϕ^1, ϕ^2 consider their quotient spaces, namely $\mathfrak{X}/\sim^1$ and $\mathfrak{X}/\sim^2$, respectively. If each subset from the quotient space $\mathfrak{X}/\sim^1$ is a contained within a subset in quotient space $\mathfrak{X}/\sim^2$, then we say that ϕ^1 is *finer* than ϕ^2. Intuitively, this is because it groups the elements into smaller (finer) sets. On the other hand, ϕ^2 is *coarser*, because its equivalence classes contain more elements. Such situation is presented in Fig. 3.6.

We say that ϕ^1 is *finer* than ϕ^2, $\phi^1 \succeq \phi^2$ if:

$$\phi^1 \succeq \phi^2 \iff \underset{\mathbf{s},\mathbf{r}\in\mathfrak{X}}{\forall} \phi^1(\mathbf{s}) = \phi^1(\mathbf{r}) \implies \phi^2(\mathbf{s}) = \phi^2(\mathbf{r}) \tag{3.6}$$

or equivalently:

$$\phi^1 \succeq \phi^2 \iff \underset{\hat{\mathbf{x}}\in\hat{\mathfrak{X}}^1}{\forall} \underset{\hat{\mathbf{y}}\in\hat{\mathfrak{X}}^2}{\exists} [\hat{\mathbf{x}}] \subseteq [\hat{\mathbf{y}}] \tag{3.7}$$

and that ϕ^1 is *coarser* than ϕ^2, $\phi^2 \succeq \phi^1$ when the opposite is true. In both cases ϕ^1 and ϕ^2 are *comparable*. If neither $\phi^1 \succeq \phi^2$ nor $\phi^2 \succeq \phi^1$ then they are *incomparable*. The $\succeq$ relation induces partial ordering of abstraction functions [72].

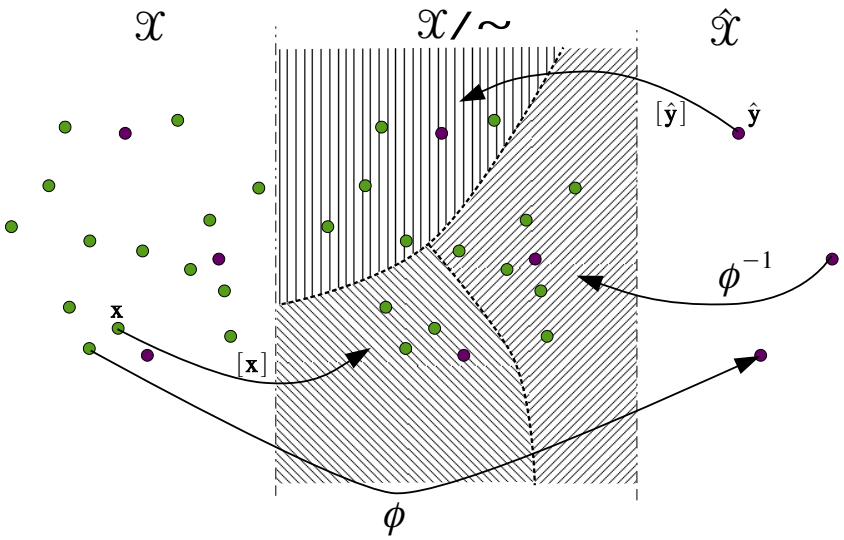

Figure 3.4: Measurements (left), quotient space (middle), abstract space (right). Exemplary arrows show relations between diagram elements: vectors from the $\mathfrak{X}$ are mapped by the abstraction function ϕ to their representatives (purple vectors) from the abstract space, which correspond to equivalence classes from the quotient space $\mathfrak{X}/\sim$. The inverse mapping, ϕ^{-1} maps the representative to the corresponding equivalence class. The [] operation maps a vector ($\mathbf{x}$) to its equivalence class ($[\mathbf{x}]$). With a little abuse of notation we also apply this operator to representatives - in the figure the abstract vector $\hat{\mathbf{y}}$ is mapped to the equivalence class $[\hat{\mathbf{y}}]$ it represents.

Example 3.3.1. Let us illustrate the *finer* notion in the context of the Cart-Pole Swing-Up example. Consider an output that consists of the following measurables: cart's position x and its velocity v. Let the vectors of measurables have the form $\mathbf{x} = (x, v)$, so $\mathbf{x}_1 = x$ and $\mathbf{x}_2 = v$. Given two vectors, $\mathbf{x}$ and $\mathbf{y}$, consider a relation $\sim^a$ defined as:

$$\mathbf{x} \sim^a \mathbf{y} \iff \text{sgn}(\mathbf{x}_2) = sgn(\mathbf{y}_2)$$

and another relation $\sim^b$:

$$\mathbf{x} \sim^b \mathbf{y} \iff \text{sgn}(\mathbf{x}_2) = sgn(\mathbf{y}_2) \wedge \text{sgn}(\mathbf{x}_2 - 0.5) = sgn(\mathbf{y}_2 - 0.5)$$

The first relation, $\sim^a$ induces an abstract space $\hat{\mathfrak{X}}_a$, which consists of two abstract states: $\hat{\mathbf{x}}^a$ that groups measurements with nonnegative value of cart's velocity, and $\hat{\mathbf{y}}^a$ aggregating measurements with negative velocity. The second relation, $\sim^b$ groups measurements into three abstract states, $\hat{\mathbf{x}}^b, \hat{\mathbf{y}}^b, \hat{\mathbf{z}}^b$, forming an abstract state space $\hat{\mathfrak{X}}_b$. These abstract states aggregate measurements with: (i) velocity greater or equal to 0.5 (ii) velocity in range $[0, 0.5)$ (iii) neg-

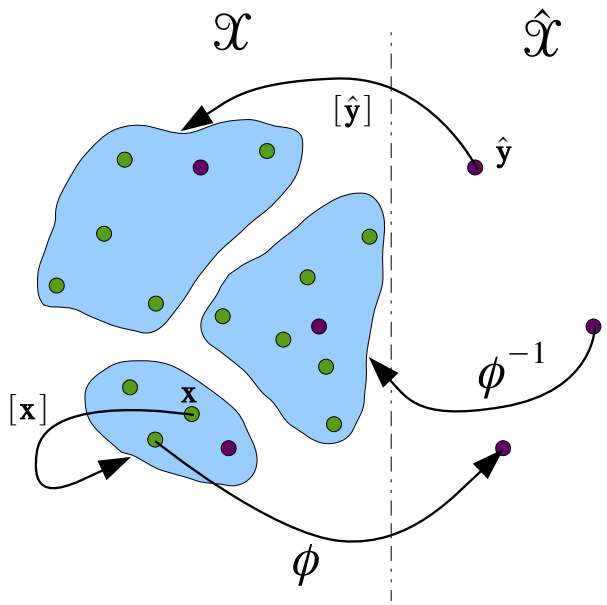

Figure 3.5: The measurements (left), and the abstract space (right). A simplified visualization of the abstraction mapping from Fig. 3.4 without the quotient space. For simplicity, equivalence classes from the quotient space are drawn directly in the source space $\mathcal{X}$. Exemplary arrows show relations between diagram elements: vectors from the $\mathcal{X}$ are mapped by the abstraction function ϕ to their representatives (purple vectors) from the abstract space, which correspond to equivalence classes. The inverse mapping, ϕ^{-1} maps the representative to the corresponding equivalence class. The [] operation maps a vector $\mathbf{x}$ to its equivalence class $[\mathbf{x}]$. With a little abuse of notation we also apply this operator to representatives - in the figure the abstract vector $\hat{\mathbf{y}}$ is mapped to the equivalence class $[\hat{\mathbf{y}}]$ it represents.

ative velocity. Further, considering abstraction functions $\phi_a : \mathcal{X} \longrightarrow \hat{\mathcal{X}}_a$ and $\phi_b : \mathcal{X} \longrightarrow \hat{\mathcal{X}}_b$ defined as in (3.4) we have, according to (3.6):

$$\phi_b \succeq \phi_a$$

□

When applying the state abstraction to dynamical systems we can consider two ways of categorizing abstractions. One relates to the way the abstraction affects the properties of the system, and the other to some specific way the abstraction simplifies the state space. The following Sec. 3.4 presents a specific category of abstractions we focus on in this work in the first sense. Further, in Sec. 3.5 we describe specific type of abstractions we are interested in, in the second meaning.

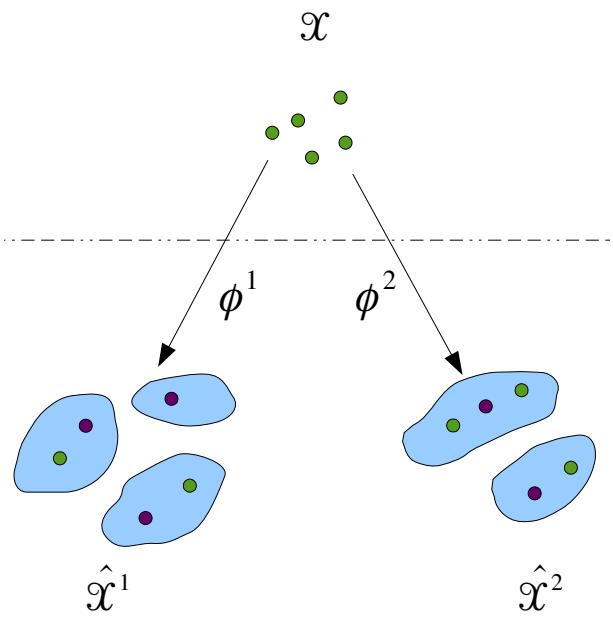

Figure 3.6: Two different equivalence relations on one set $\mathfrak{X}$ that induce two abstraction mappings ϕ^1 and ϕ^2. For simplicity, in the bottom of the picture, we present the sets corresponding to the equivalence classes, with their chosen representatives and denote them with the names of sets containing only the representatives, namely $\hat{\mathfrak{X}}^1$ and $\hat{\mathfrak{X}}^2$. According to (3.6), ϕ^1 is finer than ϕ^2. This is because the top subset on the right side of the figure covers the left and the top subset on the left side of the figure, and the bottom sets are equal.

3.4 Model-preserving abstractions

In this section we impose additional conditions to control the way the abstractions affect the dynamical system. We want to simplify the space of measurables as much as possible, but not at the cost of loosing the possibility to accurately model the system's behaviour.

Let us consider an abstraction function ϕ that transforms a given state space $\mathfrak{X}$ into an abstract state space $\hat{\mathfrak{X}}$. We focus first on the simplest assumption: we do not want to loose any information about the behaviour of the transition function f_x and the reinforcement function f_r. This is equivalent of finding another model that is in a *bisimulation* relation [94] with the given model. The term bisimulation comes from Automata Theory, and denotes a binary relation between two automata that is true, if one system can emulate the other one and vice-versa in terms of the observed output for the given input. In our context, this means that we demand that the new model, built on the abstract space $\hat{\mathfrak{X}}$ behaves in the same way as the original model, using $\mathfrak{X}$. "Behaving in the same way" means that for an appropriate labeling of measurements

of these systems they are indistinguishable. A simple example of a bisimulation relation is presented in Fig. 3.7. This requirement can be formulated as follows:

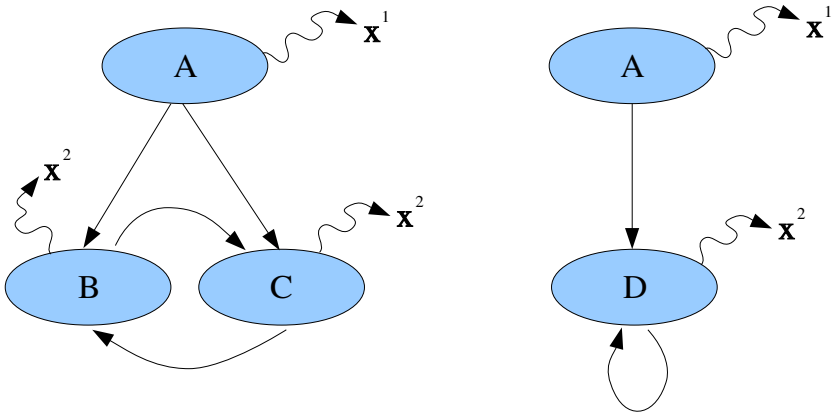

Figure 3.7: A simple example of two models with only one possible action being in a bisimulation relation. The states "C" and "D" may be different in some way, but as long as they produce the same measurements ($\mathbf{x}^2$ in this case) and thus induce the same system's behaviour, they can be folded into one state.

$$\forall_{\mathbf{x},\mathbf{y}\in\mathcal{X}} \forall_{u\in\mathcal{U}} \phi(\mathbf{x}) = \phi(\mathbf{y}) \implies f_x(\mathbf{x},u) = f_x(\mathbf{y},u) \tag{3.8}$$

Model-preserving abstractions in Reinforcement Learning A similar requirement can be reached within the Reinforcement Learning context. First, let us briefly present the most basic version of this setting. The following reasoning applies to both discrete and continuous domains however, for simplicity, we consider a discrete domain, with a MDP: $\langle \mathcal{X}, \mathcal{U}, f_x, f_r \rangle$. Denote with $r_{t+i}, i = 1,2,\ldots$ subsequent results of the reinforcement function f_r when interacting with the plant, according to a policy π that chooses an action for a given measurement: $\phi : \mathcal{X} \longrightarrow \mathcal{U}$. Then, the *value function* for a given measurement in the context of the policy π is defined as:

$$V^{\pi}(\mathbf{x}) = \mathcal{E}\{r_{t+1}, \gamma r_{t+2}, \gamma^2 r_{t+3}, \ldots \mid \mathbf{x}_t = \mathbf{x}, \pi\} \tag{3.9}$$

where $0 \leq \gamma < 1$ is a discount factor [4]. An optimal value function is defined as follows:

$$V^*(\mathbf{x}) = \max_{u\in\mathcal{U}} [f_r(\mathbf{x},u) + \gamma V^*(f_x(\mathbf{x},u))] \tag{3.10}$$

The most basic *Dynamic Programming*[6] learning algorithm for approximating the optimal value function at each iteration k successively improves its estimate:

$$V_{k+1}(\mathbf{x}) = \max_{u\in\mathcal{U}} [f_r(\mathbf{x},u) + \gamma V_k(f_x(\mathbf{x},u))] \tag{3.11}$$

This operation is called a *backup* [4] because it improves the information about the value function in the predecessor measurement $\mathbf{x}$ using information from the successor measurements (a result of $f_x(\mathbf{x},u)$). Results in each step depend on the outcomes of the transition and the reinforcement functions, f_x and f_r respectively. For a particular action u this pair of functions is called the *one-step model* of action u [4].

Since the presented reinforcement *backup* behaviour of the RL learning algorithms, we require that a valid abstraction function ϕ preserves information about reinforcements for all measurements. Thus, all measurements aggregated into one abstract state must share the same immediate and future reinforcement values for all actions:

$$\underset{\mathbf{x},\mathbf{y}\in\mathcal{X}}{\forall}\ \underset{u\in\mathcal{U}}{\forall}\ \phi(\mathbf{x})=\phi(\mathbf{y}) \implies f_r(\mathbf{x},u)=f_r(\mathbf{y},u) \tag{3.12}$$

$$\underset{\mathbf{x},\mathbf{y}\in\mathcal{X}}{\forall}\ \underset{u^1,u^2\in\mathcal{U}}{\forall}\ \phi(\mathbf{x})=\phi(\mathbf{y}) \implies f_r(\underline{f_x(\mathbf{x},u^1)},u^2)=f_r(\underline{f_x(\mathbf{y},u^1)},u^2) \tag{3.13}$$

Equality of the underlined terms is sufficient (but not necessary) for the second condition to hold. Although demanding this equality means imposing stronger constraints than needed it leads to a simple problem statement. Substituting the second condition by the above, stronger constraints produces an intuitive requirement of preserving the *one-step model* of each action of the environment:

$$\underset{\mathbf{x},\mathbf{y}\in\mathcal{X}}{\forall}\ \underset{u\in\mathcal{U}}{\forall}\ \phi(\mathbf{x})=\phi(\mathbf{y}) \implies f_r(\mathbf{x},u)=f_r(\mathbf{y},u) \tag{3.14}$$

$$\underset{\mathbf{x},\mathbf{y}\in\mathcal{X}}{\forall}\ \underset{u\in\mathcal{U}}{\forall}\ \phi(\mathbf{x})=\phi(\mathbf{y}) \implies f_x(\mathbf{x},u)=f_x(\mathbf{y},u) \tag{3.15}$$

If we treat the reinforcement function as one of the measurables, we reach a similar formula as in (3.8), except the set of measurables is extended. Throughout this work we consider an abstraction to be *correct* if it satisfies this requirement.

In Li, Walsh and Littman [72] this type of abstraction is called the ϕ_{model} abstraction.

Thought Experiment 3.4.1. Let us illustrate this particular, model-preserving, type of variable selection state abstraction with an example of the Discrete Labyrinth environment. Introduce the following set of measurables: x - horizontal position in the labyrinth, y - vertical position in the labyrinth. We will show that the model-preserving abstraction function ϕ, which ignores the vertical position, is not correct since it does not fulfill the conditions in (3.8). It is sufficient to find a pair of measurements $\mathbf{x}$, $\mathbf{y}$ represented by a single abstract state $\hat{\mathbf{x}}$, such that there exists an action u, for which the results of transition or the corresponding reinforcement function are

different. Such a pair of measurements is presented in Fig. 3.8, and the result of applying the transition function can be seen in Fig. 3.9. Because the "vertical wall" between $x = 2$ and

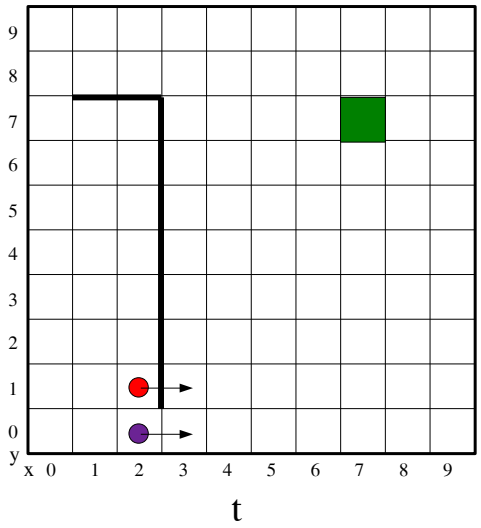

Figure 3.8: A pair of measurements (red and purple points) represented by one abstract state $\hat{\mathbf{x}} = (2)$. Action $u =$ RIGHT means, in this environment, an attempt to move one step to the right.

$x = 3$ starts at $y = 1$ and goes up the board, we choose the following measurements: $\mathbf{x} = (2,0)$ and $\mathbf{y} = (2,1)$, which belong to the same abstract state, namely:

$$\phi(\mathbf{x}) = \phi_{\{x\}}((2,0)) = (2) = \hat{\mathbf{x}}$$
$$\phi(\mathbf{y}) = \phi_{\{x\}}((2,1)) = (2) = \hat{\mathbf{x}}$$

Now, for an action $u =$ RIGHT we obtain the following:

$$f_x(\mathbf{x},u) = f_x((2,0),\text{RIGHT}) = (3,0) \quad \text{// agent moves one step to the right}$$
$$f_x(\mathbf{y},u) = f_x((2,1),\text{RIGHT}) = (2,0) \quad \text{// agent hits the wall}$$

Also, the reinforcement function yields different results:

$$f_r(\mathbf{x},u) = f_r((2,0),\text{RIGHT}) = -1 \quad \text{// agent moves one step to the right}$$
$$f_r(\mathbf{y},u) = f_r((2,1),\text{RIGHT}) = -5 \quad \text{// agent hits the wall}$$

Either of these discrepancies is enough to prove that ϕ is not a valid model-preserving abstraction.

□

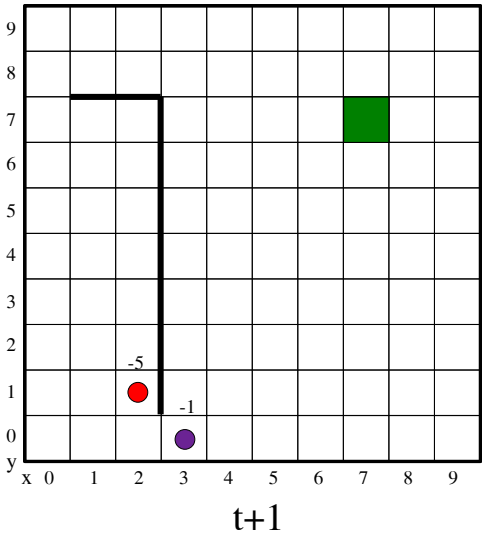

Figure 3.9: Results of applying the transition and reinforcement functions to the measurements presented in Fig. 3.8. Despite being represented by a single abstract state, the red and purple measurements produce different outcomes in terms of the successor measurement (presented in this picture) and the received reinforcement (numbers above the measurements).

3.5 State abstraction by variable selection

In this section we describe a specific subset of abstraction functions we focus on in this work, namely the *variable selection* abstractions. The *variable selection* problem consists of selecting particular variables from a given set that forms the *space of measurables* $\mathfrak{X}$. In the context of Sec. 3.3, this is a special case of state abstraction. Choosing only from a set of given measurables, we consider only certain measurables (in general, variables or coordinates) of $\mathfrak{X}$, disregarding the others (Fig. 3.10) . As mentioned in Sec. 3.2, measurables that are disregarded by a particular abstraction are called *ignored* or *redundant*, whereas we refer to the others as to *preserved*. The variable selection induces a particular type of equivalence classes (3.2), namely, those that aggregate vectors whose all coordinates (vector elements) except the ignored ones are identical. Constraining the abstractions in such a way makes it easy to reason about incremental modifications of such abstractions, because we can focus on certain measurables/coordinates instead of individual measurements. Also, it is easier to apply our ideas to continuous domains (Chapt. 6). Although we consider only variable selection abstractions, in other figures in this work we visualize the abstract states as arbitrary, irregular, blue shapes, as in Sec. 3.3 for clarity.

Analyzing such abstractions is convenient when referring to measurables from the given

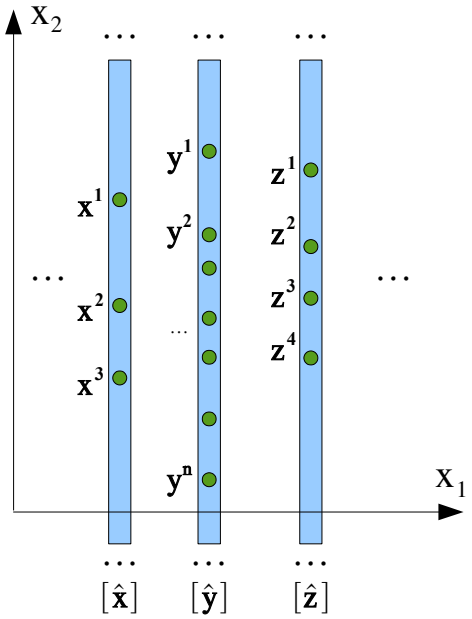

Figure 3.10: State abstraction by variable selection. The measurable $\mathbf{x}_2$ above is *ignored* and $\mathbf{x}_1$ is *preserved*. Effectively, this means that all measurements with equal values of the first element ($\mathbf{x}_1$) are grouped into a single equivalence class. Each of these equivalence classes has a *representative* in the abstract space $\hat{\mathcal{X}}$. The representatives are abstract states, which, in this case, have only single elements, namely values of measurable $\mathbf{x}_1$.

space of measurables using their ordinal numbers, called *indices*, taking into account the order in which they are introduced. For example, in the Cart-Pole Swing-Up task, first five of its measurables x, v, s, c, a have indices $0, 1, 2, 3, 4$ resp. Considering sets containing these *indices of measurables*, we can compactly denote sets of measurables that are *preserved* by an abstraction. We call such sets the *index sets*. So, each abstraction when applied to a vector, produces another vector created from the elements at indices corresponding to the measurables *preserved* by the abstraction. More formally, consider the index set $\mathcal{I} \subseteq \{1, \ldots, n\}$, with m elements: $\|\mathcal{I}\| = m$. Then, having an n-dimensional space of measurables $\mathcal{X}$, we consider the following family of abstraction functions:

$$\phi_{\mathcal{I}}(\mathbf{x}) = \hat{\mathbf{x}} = (\mathbf{x}_{i^1}, \ldots, \mathbf{x}_{i^m}), i^1, \ldots, i^m \in \mathcal{I} \tag{3.16}$$

The index set $\mathcal{I}$ in abstraction $\phi_{\mathcal{I}}$ is the set of indices of measurables, and in turn indices of vector elements, which are *preserved* by the abstraction (the others are *ignored*). Because there are 2^n possible index sets, there are 2^n possible abstraction functions of this type. We call such abstractions the *variable selection abstractions* (in [40] such abstractions are called

fluentwise[1]). We sometimes write just ϕ to denote any abstraction function of this type. We say that we *insert a measurable* into an abstraction to indicate that we modify the abstraction's index set by inserting the appropriate index. Similarly, we say that we *remove a measurable* from an abstraction, when we modify its index set by removing the appropriate index.

For a n-dimensional space of measurables, $\mathcal{X} \subseteq \mathcal{R}^n$ we call the abstraction that preserves all the variables $\phi_{\{1,\ldots,n\}}$ the *trivial abstraction*. The abstraction that ignores all variables $\phi_{\emptyset}$ is called the *null abstraction*.

Thought Experiment 3.5.1. We will now show a simple example of a variable selection state abstraction. Consider the Cart-Pole Swing-Up example again. Similarly to Example 3.3.1, assume we have the following measurables: $\{x, v, sin(\theta), cos(\theta)\}$ (where x is cart's position, v its velocity and θ the pole's angle). Thus, vectors from the space formed by these measurables will have the form $\mathbf{x} = (x, v, sin(\theta), cos(\theta))$, so $\mathbf{x}_1 = x$, $\mathbf{x}_2 = v$, $\mathbf{x}_3 = sin(\theta)$ and $\mathbf{x}_4 = cos(\theta)$. All possible variable selection abstractions are represented by all possible subsets of the index set $\mathcal{I} = \{1,2,3,4\}$. For example, an equivalence relation ignores the cart's position ($\mathbf{x}_1$) and cosine of the pole's angle ($\mathbf{x}_4$) looks as follows:

$$\mathbf{x} \sim \mathbf{y} \iff \mathbf{x}_2 = \mathbf{y}_2 \wedge \mathbf{x}_3 = \mathbf{y}_3$$

and corresponds with an index set $\{2,3\}$ leading to the following variable selection abstraction function:

$$\phi_{\{2,3\}}(\mathbf{x}) = (\mathbf{x}_2, \mathbf{x}_3)$$

Beside making cart's position unknown, this abstraction also makes it impossible to tell the measurements when the pole is pointing straight up from the measurements having it pointing straight down apart:

$$\begin{aligned}\phi_{\{2,3\}}((x = 0.9, v = 0.2, sin(\theta) = 0, cos(\theta) = 1)) \\ = \phi_{\{2,3\}}((x = 0.1, v = 0.2, sin(\theta) = 0, cos(\theta) = -1)) \\ = (v = 0.2, sin(\theta) = 0)\end{aligned}$$

□

[1]The name *fluentwise* comes from the term *fluent*, and this is how the authors call *variables*

3.6 Other types of abstractions

For completeness in this section we present briefly other types of abstractions than the one presented in Sec. 3.4. They are not related directly to this work, however their description provides a complete view on how abstractions can affect an MDP. They are also mentioned in Sec. 3.7, where we present other works related to state abstraction in RL.

Recall the RL system introduced in Sec. 3.4. Related to the *value function* defined in (3.9) is the *action-value function*, defined as:

$$Q^{\pi}(\mathbf{x},u) = \mathscr{E}\{r_{t+1}, \gamma r_{t+2}, \gamma^2 r_{t+3}, \ldots \mid \mathbf{x}_t = \mathbf{x}, u_t = u\} \tag{3.17}$$

Similarly to the *value function*, this function returns the expected reinforcement the agent will receive assuming it starts from a given state and assuming its first executed action. This first action does not have to be necessary conformable with the policy π, but the subsequent actions do. Li, Walsh and Littman [72], besides the model-preserving abstraction define also:

- ϕ_{model} - abstractions that preserve the model
- $\phi_{Q^{\pi}}$ - abstractions that preserve the *action-value function* (for all possible policies)
- ϕ_{Q^*} - abstractions that preserve the *action-value function* only for the optimal policy
- ϕ_{u^*} - abstractions that preserve the *action-value function* values only for optimal actions
- ϕ_{π^*} - abstractions that preserve the optimal actions (i.e. the fact that they have the highest values of the *action-value function*)

They prove that the following holds:

$$\phi_{model} \succeq \phi_{Q^{\pi}} \succeq \phi_{Q^*} \succeq \phi_{a^*} \succeq \phi_{\pi^*} \tag{3.18}$$

Because in the RL the algorithm's quality is solely dependent on the accuracy of the estimated value function, $\phi_{Q^{\pi}}$, ϕ_{Q^*} and ϕ_{u^*} are the most popular types of abstractions in the RL-related research. ϕ_{model} abstraction, discussed in Sec. 3.4, is the only one among these that is not attached to the RL framework and can be possibly applied in a wider context, because it does not depend on value nor action-value functions. Also, in the context of a particular plant, ϕ_{model} abstractions can be easily analyzed and verified by domain experts, as they preserve physical relations between the measurables. Also, to the best knowledge of the author, there are no

reported results for ϕ_{model} abstractions in the continuous domain. This work is thus focused on ϕ_{model} abstractions, which are the *finest* abstractions. This means that they minimally simplify the state space, but retain the most information.

3.7 Related work

Our direct inspiration comes from the work of Givan, Dean and Grieg [40] in which they apply the *bisimulation* notion taken from the Finite Automata Theory to Markov Decision Processes. The follow up research by Ferns et al. [30] and by Comanici et al. [16] apply their ideas to MDPs with continuous transition and reinforcement functions. In Sec. 4.2 we reformulate the underlying theory presented in [40] in the context of *variable selection abstractions* (see Sec. 3.5), which also enables its application in the continuous context in a simpler and more suitable for direct practical application manner than in [30]. Because we focus only on the *model-preserving* abstractions ([72]), our idea is applicable to a broader range of models. A similar idea, also based on *bisimulation*, is presented by Gol et al. [41] in the context of switched linear systems. McCallum in [77] and Seipp and Helmert in [101] introduce the incremental bottom-up approaches to state abstraction, which is also an important paradigm in this work. Jonsson and Barto in [57] present an algorithm which uses a given Bayesian network model to derive ϕ_{model} abstractions for discovered options. It is an interesting idea, as it seems plausible that domain experts should be able to provide such a model for their domain. In this work, however, we do not assume that such model is available. Similar works aim to model the environment with a discovered dynamic Bayesian network model. Such approaches are presented in [21], [47] [23] and [86]. Most of these works are based on models with function providing relative effects of actions instead of transition function. Problems with using Dynamic Bayesian Networks (DBNs) include impossibility to model synchronous arcs, and necessity to maintain models for all, even irrelevant variables. For relative actions, DBN models must be evaluated for all states, which seems unfeasible for large, continuous domains. The algorithm presented by Nguyen et al. [86] is the only one, known to the authors, capable to tackle high-dimensional domains, however only binary features are considered.

In Tables 3.1 and 3.2 we present results from these works for discrete and continuous domains resp. The approach presented in this work has been included in the last row of each table for comparison. The lack of results in the continuous state case demonstrates that this setting

is a challenge. Current theoretical results rely on estimating the distance between probability distributions, which is hard to implement in practice. The simplification we propose in Sec. 4.2 deals with this issue.

Table 3.1: Papers presenting the model-preserving abstraction approach for discrete domains.

Paper	Year	Solved task	State count	Tools
[40]	2003	"Expon" binary domain	512	Bisimulation
[16]	2012	Labyrinth	900	Bisimulation
[57]	2006	Coffee task	32(1)	Given DBN
[21]	2006	Process planning	131072	DBN
[47]	2009	Labyrinth	121	DBN
[23]	2009	System administrator	256	DBN
[89]	2010	Taxi	500	Graph, given MAXQ hierarchy
[86]	2013	Robot navigation (real-world)	512	DBN
This work	2015	Labyrinth	250000	Bisimulation

(1)Even though the Coffee task consists of 6 binary measurables, its effective number of states is 32 because the variable indicating whether the user has coffee changes its value only at the natural end of the episode.

3.7.1 Related work on other types of abstraction in RL

In this section we briefly present approaches that rely abstractions of different type than ϕ_{model}. Tables 3.3 and 3.4 present summary of works for these paradigms, for discrete and continuous domains, respectively. Apart from the classification presented in [72], and discussed in Sec. 3.6, research trends concerning state abstraction in RL can be divided into two groups, namely the *state aggregation* and the *state space abstraction*. While the state space abstraction algorithms are special cases of state aggregation algorithms, in practice they differ significantly. This is because the first are based on some properties relevant to the whole state space, whereas the latter consider particular states. To the best knowledge of the author, there are no works with practical evaluation of ϕ_{model} *state space abstraction* algorithms for the continuous domains. One of aims of this work is to fill this gap.

Table 3.2: Papers presenting the model-preserving abstraction approach for continuous domains.

Paper	Year	Solved task	Number of measurables	Tools
[20]	1997	(Theory only)	(Theory only)	Bounded MDP
[31]	2004	(Theory only)	(Theory only)	Bisimulation
[30]	2011	(Theory only)	(Theory only)	Bisimulation
This work	2015	Cart-Pole Swing-Up, Mountain Car	10, 20, resp.	Bisimulation

The state aggregation group consists of algorithms that aggregate the states according to some similarity measure. McCallum in [77] presents an algorithm that groups states according to their optimal action, which conforms to ϕ_{u^*} type of abstraction. More recent works [76], [73], [114] describe the methods that group the states sharing the same values of the value function. The graph approach presented in [76] is based on detecting "frequent states". Juang [59] presents an interesting solution, which generates the features in a form of fuzzy-rules. Parameters of these rules are trained using a genetic algorithm. Nouri and Littman in [87] propose to use a decision tree, introducing a concept of continuous *knownness* of a state. Algorithms that tackle domains with continuous transition functions are the most interesting, since they bring RL closer to being applicable to real-world problems. Provost, Kuiper and Miikkulainen [96] present an algorithm that performs vector quantization using Kohonen network with adaptive structure. The vectors in network's units are then used as a basis of an abstract state space. The algorithm is evaluated on a high-dimensional, continuous domain which is a very promising result. The basis function construction idea is further considered by Huang, Xu and Zuo in [53].

Methods from the state space abstraction group aim to discover a subspace that preserves some of the properties of the original state space. Konidaris and Barto [68] describe an algorithm that chooses one of predefined sets of basis functions to form a set of *options* ([109]). This algorithm presents a valuable concept of incorporating domain knowledge into abstraction algorithms in the form of predefined abstractions. The algorithm presented by Jong ([56]) attempts to determine the measurables that are irrelevant in terms of the target optimal policy. Many approaches from this group aim to discover features, also called *basis functions* that can

be linearly combined to approximate the value function. Mahadevan [75] presents a theoretical foundation of basis function based approximation and presents an algorithm deriving a set of basis functions to approximate the value function from environment's transition graph. Other works, like [89] by Osentoski and Mahadevan or [107] by Sprague present algorithms capable of automatic generation of such features. A combination of bisimulation metrics and basis function oriented approach is presented by Comanici and Precup in [17]. Cobo et al. present an interesting approach on incorporating domain knowledge - learning abstraction from human demonstration.

There is also a similar research trend in the Control Theory context, namely the Model Order Reduction methods. These techniques include, for example, Truncated Balanced Realization [42] and Krylov Subspace methods [103]. Both methods project the original state space into a smaller space in a way that is optimal in terms of preserving controllability and/or observability of the plant. This somehow resembles MDP methods that aim to preserve the value function of a model, if we look at the reinforcement as the output of the system. They are however limited to linear systems.

Most papers employ the *top-down* approach that consists of reducing the abstract state space from finer abstractions towards the coarser ones (see (3.6)). Some of them, like [77], propose the bottom-up direction: extending the abstract state space, making it finer.

Table 3.3: Papers presenting approaches with types of abstractions different than model-preserving in discrete transition functions.

Paper	**Year**	**Abstraction type**	**Solved task**	**State count**	**Tools**
[77]	1995	ϕ_{u^*}	Spaceship docking	10	Decision tree
[75]	2005	ϕ_{Q^*}	Labyrinth	1260	Graph
[56]	2005	ϕ_{π^*}	Taxi	500	Given π^*
[15]	2014		Frogger	$\sim 10^{251}$	Decision tree, human demonstration

Table 3.4: Papers presenting approaches with types of abstractions different than model-preserving in continuous transition functions.

Paper	**Year**	**Abstraction type**	**Solved task**	**Number of measurables**	**Tools**
[76]	2004	ϕ_{Q^*}	Car-Hill	2	CMAC
[59]	2005		Cart-Pole	4	Fuzzy rules
[114]	2006		Plane navigation	3	ART network
[96]	2006		Corridor navigation	180	Kohonen network
[87]	2009		Car-Hill	2	Decision tree
[73]	2010		Cart-Pole	4	Graph
[107]	2007	ϕ_{Q^*}	Robot navigation	121	Neighborhood component analysis
[68]	2009		Play room	18	Bayesian information criterion, given basis functions

Chapter 4

Ambiguity resolving approach

This chapter presents an introduction to the main idea, which originates from the existing works on *bisimulation* in RL [40], [30], [16], [92]. We present a formal problem statement and then introduce the *ambiguity functional* in its general form, which is the core notion in our framework. We analyze its properties in the context of the presented problem statement. This is a basis for the following Chapters 5 and 6, where we propose concrete definitions suitable for discrete and continuous domains respectively.

4.1 Problem statement

Our goal is to design an algorithm that finds a valid, model-preserving, abstraction function $\phi_{\mathcal{J}}$ (see Sec. 3.4), in general applicable to both the discrete and the continuous settings. $\mathcal{J}$ represents measurables preserved by the abstraction function we wish to find. We express this goal as the following non-linear minimization problem. Since want to find the smallest possible set of measurables, we minimize the number of elements in the abstraction's index set $\mathcal{J}$. The index set $\mathcal{J}$ being minimized implies the abstract state space $\hat{\mathfrak{X}} = \hat{\mathfrak{X}}_{\mathcal{J}}$. Constraints of such minimization consist of conditions that define a valid model-preserving abstraction defined in (3.14) and (3.15). Expressing these conditions for the continuous case is possible in more than one way: we can demand that *almost all* measurements being represented by one abstract state satisfy the conditions, that all of them violate the condition by no more than a predefined constant, or a combination of these two. We choose the simplest approach, namely that we demand that all representatives of all measurements satisfy the *model-preserving* abstraction conditions - this however can be easily extended to any of the aforementioned, more complex

approaches. This gives us the following problem statement:

$$\begin{aligned} min \quad & \|\mathcal{F}\| \\ \text{s.t.} \quad & \underset{u\in\mathcal{U}}{\forall}\ \underset{\hat{\mathbf{x}}\in\hat{\mathfrak{X}}}{\forall}\ \underset{\mathbf{x},\mathbf{y}\in[\hat{\mathbf{x}}]}{\forall} |\phi_{\mathcal{F}}(f_x(\mathbf{x},u)) - \phi_{\mathcal{F}}(f_x(\mathbf{y},u))| \leq k_x \\ & \underset{u\in\mathcal{U}}{\forall}\ \underset{\hat{\mathbf{x}}\in\hat{\mathfrak{X}}}{\forall}\ \underset{\mathbf{x},\mathbf{y}\in[\hat{\mathbf{x}}]}{\forall} |f_r(\mathbf{x},u) - f_r(\mathbf{y},u)| \leq k_r \\ & \|\mathcal{F}\| > 0 \end{aligned} \tag{4.1}$$

where parameters k_x and k_r are related to the Lipschitz continuity of transition and reinforcement functions. For the discrete setting, we change the constraints to an exact equality of the successor abstract states and reinforcements by choosing $k_x = k_r = 0$:

$$\begin{aligned} min \quad & \|\mathcal{F}\| \\ \text{s.t.} \quad & \underset{u\in\mathcal{U}}{\forall}\ \underset{\hat{\mathbf{x}}\in\hat{\mathfrak{X}}}{\forall}\ \underset{\mathbf{x},\mathbf{y}\in[\hat{\mathbf{x}}]}{\forall} |\phi_{\mathcal{F}}(f_x(\mathbf{x},u)) - \phi_{\mathcal{F}}(f_x(\mathbf{y},u))| = 0 \\ & \underset{u\in\mathcal{U}}{\forall}\ \underset{\hat{\mathbf{x}}\in\hat{\mathfrak{X}}}{\forall}\ \underset{\mathbf{x},\mathbf{y}\in[\hat{\mathbf{x}}]}{\forall} |f_r(\mathbf{x},u) - f_r(\mathbf{y},u)| = 0 \\ & \|\mathcal{F}\| > 0 \end{aligned} \tag{4.2}$$

Note that because the choice of abstractions is restricted to the variable selection abstractions (as described in Sec. 3.5) the problem statement is much simpler than in [30] despite dealing with continuous transition functions. The general state abstraction problem, not restricted to any particular type of abstractions is an *NP-hard* problem [64]. In Sec. 4.3 we will show that restricting the type of state abstraction to the particular type we focus on in this work does not change that, i.e. the state abstraction problem, restricted to the variable selection abstractions is also an *NP-hard* problem. In further sections we first analyze the discrete case for the sake of simplicity, and then extend the reasoning to the continuous case.

Thought Experiment 4.1.1. We will now present a simple example that consists in solving the minimization problem (4.1) by hand. Consider the Continuous Labyrinth environment with the following set of measurables: x - horizontal position in the labyrinth, y - vertical position in the labyrinth, v_x - horizontal velocity, v_y - vertical velocity, $x+y$, $|x-y|$, xy. Assume $k_x = 0.12$ and $k_r = 1$. We will find the solution to the minimization program (4.1). For better readability we put names of the measurables rather than their indices in the *index sets*.

Consider two abstraction functions $\phi_{\{x,v_x,v_y,|x-y|\}}$ and $\phi_{\{y,v_x,v_y,x+y\}}$. The first abstraction ignores the composite measurables $x+y$ and xy, and the vertical position y, attempting to employ

$|x-y|$ in their place. The second one ignores the composite features $|x-y|$ and xy, and the horizontal position x, with coordinate sum $x+y$ instead.

In the context of abstraction $\phi_{\{x,v_x,v_y,|x-y|\}}$, consider the following abstract state: $(x=3, v_x = 0, v_y = 0, |x-y| = 3)$. This abstract state represents, in particular, two measurements, namely: $\mathbf{x}^1 = (x=3, y=0, v_x=0, v_y=0, \ldots)$ and $\mathbf{x}^2 = (x=3, y=6, v_x=0, v_y=0, \ldots)$. The results of the transition function for these states given the action $u = \text{LEFT}$ are as follows:

$$\mathbf{x}^1_{t+1} = f_x(\mathbf{x}^1, \text{LEFT}) = (x=2.9, y=0, v_x=-0.033\ldots, v_y=0, \ldots)$$
$$\mathbf{x}^2_{t+1} = f_x(\mathbf{x}^2, \text{LEFT}) = (x=3, y=6, v_x=0, v_y=0, \ldots)$$

In the second case, the agent hits the wall, thus its position does not change. The LHS of the first condition in Eq. (4.1) is:

$$\|\phi_{\{x,v_x,v_y,|x-y|\}}(\mathbf{x}^1_{t+1}) - \phi_{\{x,v_x,v_y,|x-y|\}}(\mathbf{x}^2_{t+1})\| = \|(2.9, -0.033\ldots, 0, 2.9) - (3,0,0,3)\| \sim 0.145$$

Because $0.145 > k_x$ the first condition is not met, so abstraction $\phi_{\{x,v_x,v_y,|x-y|\}}$ is not a valid solution.

Note that the second abstraction does not loose any information about the environment's state. This can be shown by indicating a bijection between environment's state (x, y, v_x, v_y) and this abstraction's vector of measurables: $(y, v_x, v_y, x+y)$:

$$g((y, v_x, v_y, x+y)) = (x = x+y-y, y = y, v_x = v_x, v_y = v_y)$$

Thus all LHS terms in both conditions in the minimization problem in Eq. (4.1) will be 0, therefore this abstraction potentially is a solution to this program. Similarly, abstraction with xy measurable instead of $x+y$ is a potential solution.

To verify whether it is a final solution, i.e. the one having as few measurables as possible, all possible abstractions for which $\|\mathcal{F}\| = 3$, $\|\mathcal{F}\| = 2$ and $\|\mathcal{F}\| = 1$ must be verified. First, note that measurables v_x and v_y are indispensable, as they are the only measurables carrying information about agent's velocity. Without them, the states close to the wall would lead to nondeterministic outcomes in terms of reinforcement (even small difference in velocity might determine if the agent will hit the wall or not). Similar reasoning leads to the conclusion that beside that, some information about agent's position in the labyrinth is essential. Therefore, we can assume that $\|\mathcal{F}\| > 2$ and that $2 \in \mathcal{F}$ ($v_x \in \mathcal{F}$) and $3 \in \mathcal{F}$ ($v_y \in \mathcal{F}$). Because the labyrinth contains both vertical and horizontal walls, it is safe to assume that only x or only y in addition

to v_x and v_y will not produce a valid solution. Therefore, there are 3 abstractions left to consider, namely: $\phi_{\{v_x,v_y,|x-y|\}}$, $\phi_{\{v_x,v_y,x+y\}}$ and $\phi_{\{v_x,v_y,xy\}}$.

The abstraction $\phi_{\{v_x,v_y,|x-y|\}}$ seems easiest to analyze. Since $\phi_{\{x,v_x,v_y,|x-y|\}} \succeq \phi_{\{v_x,v_y,|x-y|\}}$, every pair of measurements that is aggregated by $\phi_{\{x,v_x,v_y,|x-y|\}}$ is also aggregated by $\phi_{\{v_x,v_y,|x-y|\}}$. Thus, the same pair of states that was shown to violate minimization conditions for $\phi_{\{x,v_x,v_y,|x-y|\}}$ above, will violate these conditions for $\phi_{\{v_x,v_y,|x-y|\}}$. Therefore, $\phi_{\{v_x,v_y,|x-y|\}}$ is also an invalid solution. It can be noted that if $\phi_{\{x,v_x,v_y,|x-y|\}}$ is invalid and $\phi_{\{v_x,v_y,|x-y|\}}$ looses even more information about plant's state then it is not against intuition that $\phi_{\{v_x,v_y,|x-y|\}}$ is also invalid.

Let us now consider $\phi_{\{v_x,v_y,x+y\}}$. Similar reasoning to the one presented above for $\phi_{\{x,v_x,v_y,|x-y|\}}$ can be carried out for the following states $\mathbf{x}^1 = (x = 2, y = 2, v_x = 0, v_y = 0, \ldots)$ and $\mathbf{x}^2 = (x = 3, y = 1, v_x = 0, v_y = 0, \ldots)$. Both are contained within the same abstract state: $\phi_{\{v_x,v_y,x+y\}}(\mathbf{x}^1) = \phi_{\{v_x,v_y,x+y\}}(\mathbf{x}^2) = (v_x = 0, v_y = 0, x + y = 4)$. The results of the transition function for these states given action $u = \text{LEFT}$ are as follows:

$$\mathbf{x}^1_{t+1} = f_x(\mathbf{x}^1, \text{LEFT}) = (x = 1.9, y = 2, v_x = -0.033\ldots, v_y = 0, \ldots)$$
$$\mathbf{x}^2_{t+1} = f_x(\mathbf{x}^2, \text{LEFT}) = (x = 3, y = 1, v_x = 0, v_y = 0, \ldots)$$

In the second case, the agent hits the wall, thus its position does not change. The LHS of the first condition in Eq. (4.1) is:

$$\|\phi_{\{v_x,v_y,x+y\}}(\mathbf{x}^1_{t+1}) - \phi_{\{v_x,v_y,x+y\}}(\mathbf{x}^2_{t+1})\| = \|(-0.033\ldots, 0, 3.9) - (0, 0, 4)\| \sim 0.105$$

Because $0.105 < k_x$ the first condition is met. However, the second condition, regarding the reinforcement values, is violated, since executing the action LEFT for measurement $\mathbf{x}^1$ will make the agent increase its speed and receive -1 reinforcement, and for measurement $\mathbf{x}^2$ will result in hitting the wall, and thus receiving the -5 penalty:

$$r^1_{t+1} = f_r(\mathbf{x}^1, \text{LEFT}) = -1$$
$$r^2_{t+1} = f_r(\mathbf{x}^2, \text{LEFT}) = -5$$

The difference between these outcomes is more than the assumed tolerance:

$$|r^1_{t+1} - r^2_{t+1}| = 4 > k_r$$

Therefore, no abstraction with $\|\mathcal{F}\| = 3$ can be a solution to the minimization problem defined in Eq. (4.1) and the abstraction $(y, v_x, v_y, x + y)$ is a valid solution. Note that the solution is not unique: (x, y, v_x, v_y), among others, is equally valid.

□

One important conclusion from this example is that the Euclidean distance with one threshold value (k_x) does not always work well for our purpose. Different measurables influence the length of the measurement vector in a different way, and thus it is impossible to find a universal k_x value. In our example, we could have used a smaller value, but this in turn leads to the algorithm being oversensitive for small differences, resulting, for example, from transition function's Lipschitz constant being larger than 1. The proposed modification is thus to use a vector of thresholds, dedicated to each measurable independently, and to use the Chebyshev distance. This idea is presented in the following sections.

4.2 Bisimulation and equivalence

In this section we describe the process of constructing a ϕ model-preserving abstraction (see Sec. 3.4) by variable selection (see Sec. 3.5), basing on the derivation given in [40]. We generalize it to a continuous case in the context of variable selection abstractions. Also, as mentioned in Example 4.1.1 in the previous section, we introduce a vector parameter $\theta^{(x)}$ in place of scalar k_x from (4.1).

Givan, Dean and Greig define a function H that refines an abstraction (see Eq. (3.6)). This is achieved by disaggregating states that cause violation of the bisimulation constraints (see Eq. (3.15)). We propose to modify this approach to focus on variable selection only. In case of variable selection abstractions (as described in Sec. 3.5), states disaggregation consists of inserting a new measurable to the current model, thus forming a new higher-dimensional abstract state space. To do this, consider a variable selection abstraction function for an index set $\mathcal{I}$: $\phi_{\mathcal{I}}$, as defined in Eq. (3.16). Let $H : 2^n \longrightarrow 2^n$, be a function that transforms an index set into another index set. Similarly as in [40] we define H to split abstract states, so that measurements grouped in these abstract states do not violate bisimulation constraints anymore. The difference is that we split the states only by means of inserting a new measurable. In case of continuous transition function bisimulation constraints need to be verified in terms of small distance instead of equality. In this order we introduce two parameters: a vector $\theta^{(x)}$ for the transition function, and a scalar $\theta^{(r)}$ for the reinforcement function. The function $H(\mathcal{I})$ is constructed to create a new index set by inserting indices to satisfy the *model-preserving* conditions defined in Eqs. (3.14, 3.15). Thus, the indices are inserted to the index set $\mathcal{I}$ forming a new, refined abstraction

$\phi_{\mathcal{F}}$, until the following conditions are satisfied:

$$\begin{aligned}
&\neg \exists_{\hat{\mathbf{x}}\in\hat{\mathfrak{X}},u\in\mathcal{U}} \exists_{\mathbf{x},\mathbf{y}\in\phi_{\mathcal{F}}^{-1}(\hat{\mathbf{x}})} |f_r(\mathbf{x},u) - f_r(\mathbf{y},u)| > \theta^{(r)} \\
&\neg \exists_{\hat{\mathbf{x}}\in\hat{\mathfrak{X}},u\in\mathcal{U}} \exists_{\mathbf{x},\mathbf{y}\in\phi_{\mathcal{F}}^{-1}(\hat{\mathbf{x}})} |\phi_{\mathcal{F}}(f_x(\mathbf{x},u)) - \phi_{\mathcal{F}}(f_x(\mathbf{y},u))| > \theta^{(x)}
\end{aligned} \tag{4.3}$$

Applying H to the subsequent abstractions removes the violations of the *model-preserving* conditions (4.3) and in turn results in an abstraction that does not induce such violations. Such abstraction is regarded as a valid solution to the considered variable selection state abstraction problem. Parameters $\theta^{(x)}$ and $\theta^{(r)}$ correspond to parameters k_x and k_r from (4.1).

We now describe this modified refinement process in analogy with what is presented in [40], to show that the process behaves in the same way, leading to the correct solution. Consider a sequence of index sets $\mathcal{F}^0, \mathcal{F}^1, \ldots$ (accompanied by sequence of abstractions $\phi_{\mathcal{F}^0}, \phi_{\mathcal{F}^1}, \ldots$) where $\mathcal{F}^0 = \emptyset$ (which is equivalent to an abstraction that aggregate all states into one abstract state) and $\mathcal{F}^{j+1} = H(\mathcal{F}^j)$. Note that because H includes indices, making abstractions finer, and thus splitting the abstract states into smaller abstract states, it follows that $\mathcal{F}^j \subseteq \mathcal{F}^{j+1}$ and $\phi_{\mathcal{F}^{j+1}} \succeq \phi_{\mathcal{F}^j}$ (relation $\succeq$ is defined in (3.6)). This is a special case of the block splitting presented by Givan and Greig in [40], yet with the difference that the blocks are split by creating a new representation with an additional measurable. This sequence leads to an index set $\mathcal{F}$ being a fixed-point of H [40], which induces a *model-preserving* abstraction. Denoting this abstraction by $\phi_{\mathcal{F}}$ we note that they constitute a *quotient model* [40] of the original model: $\langle \hat{\mathfrak{X}}, \mathcal{U}, \hat{f}_x, \hat{f}_r \rangle$ with $\hat{f}_x$, $\hat{f}_r$ defined as follows:

$$\begin{aligned}
\hat{f}_x(\hat{\mathbf{x}},u) &= \phi_{\mathcal{F}}(f_x(\mathbf{x},u)) \\
\hat{f}_r(\hat{\mathbf{x}},u) &= f_r(\mathbf{x},u)
\end{aligned}$$

where $\mathbf{x}$ is an arbitrary measurement that $\mathbf{x} \in [\hat{\mathbf{x}}]$.

Because the resulting index set $\mathcal{F}$ constitutes a *model-preserving* variable selection abstraction, it satisfies the constraints of the non-linear minimization problem defined in (4.1). However, it is not necessary the global minimum. This depends on the order in which indices are being added to $\mathcal{F}$ in order to satisfy conditions of (3.14) and (3.15) when calculating intermediate results of the H function, as defined in (4.3). We deal with this issue, and with the problem of estimating $\theta^{(x)}$ and $\theta^{(r)}$ in Sec. 6.5.

If $\theta^{(x)}$ and $\theta^{(r)}$ are substituted by $\mathbf{0}$ and 0, the formulation is suitable for a discrete case.

Thought Experiment 4.2.1. We will present an example that shows that the presented idea is still an unsatisfactory heuristic for our problem. We will apply the incremental process presented above to find a valid abstraction functions (one being optimal, and one not) for the Discrete Labyrinth environment, with measurables: $x, y, x+y, |x-y|, xy$. Similar reasoning can be presented for the continuous version. Let us start with $\phi_{\mathcal{F}^0}$, with $\mathcal{F}^0 = \emptyset$. $\phi_{\mathcal{F}^0}$ ignores all measurables, so its abstract state space consists of only one abstract state: $\hat{\mathbf{x}}$. Thus, $\mathbf{x} \in [\hat{\mathbf{x}}], \mathbf{y} \in [\hat{\mathbf{x}}]$ for every possible pair of measurements $\mathbf{x}$, $\mathbf{y}$. Consider $\mathbf{x} = (x = 8, y = 7, \ldots)$ and $\mathbf{y} = (x = 7, y = 7, \ldots)$. Conditions from (4.3) (with $\theta^{(x)} = \mathbf{0}$ and $\theta^{(r)} = 0$ for the discrete case) are violated because:

$$f_r(\mathbf{x}, \text{UP}) = 10$$
$$f_r(\mathbf{y}, \text{UP}) = -1$$

(in the first case, the agent reaches the target square in the labyrinth, for which a reward of 10 is returned). This can be rectified by inserting the x measurable (for example) to the abstraction. Thus, $\phi_{\mathcal{F}^1}$ is defined by $\mathcal{F}^1 = \{x\}$. For this abstraction, similarly, another pair of states can easily be found the violates conditions defined in (4.3), namely $\mathbf{x} = (x = 9, y = 8, \ldots)$ and $\mathbf{y} = (x = 9, y = 9, \ldots)$, for action $u = \text{LEFT}$. Let us consider two possibilities now. If the measurable x was chosen for the abstraction $\phi_{\mathcal{F}^2}$ not to violate the conditions for these states, we would reach a valid solution $\mathcal{F}^2 = \{x, y\}$, and the process would finish. However, let us consider a more interesting scenario: note that the measurable $|x-y|$ also makes states $\mathbf{x}$ and $\mathbf{y}$ different from the abstraction's point of view, and thus can be chosen at this step. Let us assume then that $\mathcal{F}^2 = \{y, |x-y|\}$. Recall from the previous example that this abstraction is still not valid. This can be easily verified when considering a pair of states next to the wall: $\mathbf{x} = (x = 3, y = 0, \ldots)$ and $\mathbf{y} = (x = 3, y = 6, \ldots)$ for action $u = \text{LEFT}$. Note that $\phi_{\mathcal{F}^2}(\mathbf{x}) = \phi_{\mathcal{F}^2}(\mathbf{y}) = (0, 3)$ (because $|3-6| = |3-0| = 3$). Therefore, these states are within one abstract state, but the action $u = \text{LEFT}$ results in hitting the wall when starting from one of them, and moving to the left when starting from the other one. Therefore, one abstract state could possibly lead to two different outcomes in terms of transition and reinforcement functions. Therefore in this scenario, the process would have to insert another measurable, and would finish with abstraction with 3 measurables, $\phi_{\mathcal{F}^3}$, with $\mathcal{F}^3 = \{y, x+y, |x-y|\}$, for example.

□

This example shows that merely inserting measurables at each step can lead to a non-optimal

solution, because the feature $|x-y|$ in the last result is redundant. This is related to the fact that this problem is a *NP-hard* problem, as we will show in the following section. A heuristic that addresses this issue - mechanism of removing measurables is presented in Sec. 5.5.

4.3 Hardness of the variable selection state abstraction problem

In this section we will prove that the problem presented here is *NP-hard* and inapproximable. To do that, it is enough to prove two properties [70], [51]:

1. Given a solution, its correctness can be verified in polynomial time
2. Given a problem that is known to be *NP-hard* and inapproximable, it is possible to transform it to an instance of variable selection state abstraction problem in polynomial time

The first property is trivial for the discrete case since any abstraction mapping can be verified in linear time w.r.t. the number of possible states multiplied by the number of possible actions, by enumerating these states and verifying conditions from (3.15). In case of continuous transition and reinforcement functions it is impossible to enumerate all possible states - however, correctness can be verified by means of a grid search process, for any desired accuracy.

Let us now focus on the second property. First, we will present a semi-formal reasoning that intuitively shows that it is plausible. Next, we will present a strict proof.

Recall the incremental process presented in the previous Sec. 4.2. We are inserting subsequent measurables to remove violations of correct state abstraction conditions (see (3.15)). In this context, consider a problem with n measurables and thus an index set $\mathcal{I} = \{1,2,\ldots,n\}$. Suppose we are at step k of the process, i.e. we have added some measurables, represented by indices $i_1, i_2, \ldots, i_k \in \mathcal{I}$, but still some abstract states violating the conditions of correct abstraction remain. Denote the current abstraction mapping by ϕ_k. Let us consider a set of all correctness-violating abstract states, at step k:

$$\hat{\mathcal{V}_k} = \left\{\hat{\mathbf{x}}_k^1, \hat{\mathbf{x}}_k^2, \ldots, \hat{\mathbf{x}}_k^m\right\} \tag{4.4}$$

Recall from (3.15) that such abstract states have the following property: any two measurements exist that are being represented by this abstract state that for the same control u produce different

reinforcement values, or lead to different abstract states. This can be written as:

$$\underset{\hat{\mathbf{x}}\in\hat{\mathcal{V}}}{\forall}\ \underset{\mathbf{x}^1,\mathbf{x}^2\in[\hat{\mathbf{x}}],u\in\mathcal{U}}{\exists} \phi_k(f_x(\mathbf{x}^1,u)) \neq \phi_k(f_x(\mathbf{x}^2,u)) \vee f_r(\mathbf{x}^1,u) \neq f_r(\mathbf{x}^2,u) \tag{4.5}$$

We say that a measurable with index *i shatters* an abstract state if it causes two states $\mathbf{x}^1$, $\mathbf{x}^2$ to no longer belong to the same abstract state. Consequently, this removes one or more examples of violation of correct abstraction properties. The ultimate goal is to get rid of all violations. When the process is at step k, it can potentially choose among different indices of measurables: $i_{k+1}, i_{k+2}, \ldots, i_n$, each of them *shattering* different correctness-violating abstract states from $\hat{\mathcal{V}}_k$. This situation is presented in Fig. 4.1. Let as assign a subset of $\hat{\mathcal{V}}_k$ to each available (i.e. not already added) index of a measurable $i \in \{i_{k+1}, \ldots, i_n\}$, denoting abstract states being *shattered* by corresponding measurable. As a result, we get a set of subsets $\left\{\mathcal{V}_k^1, \mathcal{V}_k^2, \ldots, \mathcal{V}_k^{n-k}\right\}$, each $\mathcal{V}_k^j \subseteq \hat{\mathcal{V}}_k$. To reach the correct abstraction we need to *shatter* all correctness-violating abstract states. To find an optimal solution, we additionally require to achieve that with the least possible number of measurables. In terms of $\hat{\mathcal{V}}_k$ and its subsets, we wish to find the least number of subsets $\mathcal{V}_k^j \subseteq \hat{\mathcal{V}}_k$ that cover the whole $\hat{\mathcal{V}}_k$. This is clearly an instance of a well known *NP-hard* problem, namely the *set cover problem* [61]. Also, the *set cover problem* is inapproximable, and the *greedy* algorithm is the best possible approach, as shown by Feige in [29]. We have shown that any step from the incremental abstraction refinement process, described in the previous Sec. 4.2, can be formulated in terms of the *set cover problem*. However, to prove that the variable selection state abstraction problem is *NP-hard* we need to show the converse: that any instance of the *set cover problem* can be transformed into a variable selection state abstraction problem (in at most polynomial time). The reasoning is as follows:

Theorem 4.3.1. The problem of finding a model-preserving variable selection abstraction is an inapproximable *NP-hard* problem.

Proof. Consider an instance of the *set cover problem*: for a set $\mathcal{C} = \{1, 2, ..., N\}$ of N elements, and a family of its M subsets $\mathcal{S} = \left\{\mathcal{S}_j : \mathcal{S}_j \subseteq \mathcal{C}, 1 \leq j \leq M\right\}$. Assume that $\bigcup_j \mathcal{S}_j = \mathcal{C}$, i.e. a solution exists. We will now construct a Markov Decision Process, corresponding to this instance. Denote with $\hat{\mathbf{x}}_l$ an abstract state corresponding with element $l \in \mathcal{C}$. We can encode such abstract states for all elements from $\mathcal{C}$ with $m = \lceil \log_2 N \rceil$ binary measurables: $i_1, i_2, \ldots, i_m$ (each $i_l \in \{0, 1\}$). To focus on the simplest possible case for this constructed MDP, it is enough to have a single possible action $\mathcal{U} = \{u\}$ and a simple transition function $f_x(\mathbf{x}, u) = \mathbf{x}$. Now, we would like to design a reinforcement function in such a way that abstract states $\hat{\mathbf{x}}_l$ violate the

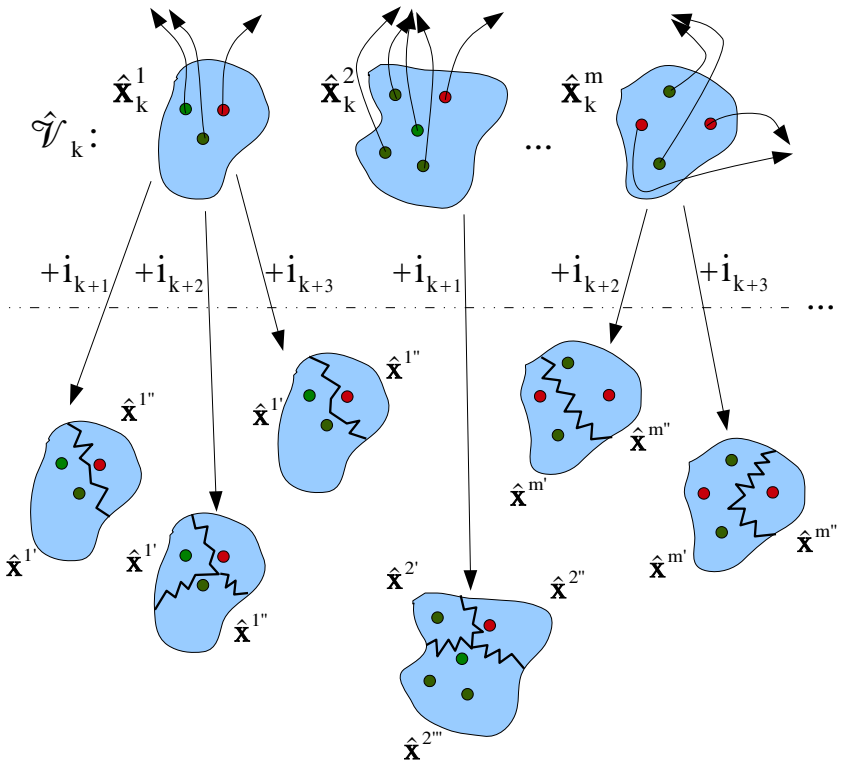

Figure 4.1: The top row represents $\hat{\mathcal{V}}_k$: the set of ambiguous abstract states (indicated by red underlying states, which produce different measurements - arrows leading in different directions). Straight arrows pointing downward present possible measurables (indices of measurables) that shatter the abstract states. Shattering an abstract state forms some smaller abstract states, each of them being less ambiguous than the initial one. Depending on which index/measurable will be chosen, abstract states will be shattered differently.

constraints of the correct abstraction, and that subsets $\mathcal{S}_j$ correspond to additional measurables that *shatter* these abstract states. For each *set cover problem* subset $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_M$ we introduce additional, corresponding measurables: $i_{m+1}, i_{m+2}, \ldots, i_{m+M}$. If a subset $\mathcal{S}_j$ contains an element l we require that the additional measurable $i_j, m < j \leq M$ *shatters* the abstract state $\hat{\mathbf{x}}_l$. This can be easily achieved by making the f_r depend on measurable i_j *iff* $l \in \mathcal{S}_j$, other measurables being irrelevant, when the current state is represented by $\hat{\mathbf{x}}_l$. Thus, f_r can be defined as:

$$\underset{\hat{\mathbf{x}}_l : l \in \mathscr{C}}{\forall} \underset{\mathcal{S}_j : l \in S_j}{\forall} \underset{\mathbf{x} \in [\hat{\mathbf{x}}_l]}{\forall} f_r(\mathbf{x}, u) = \begin{cases} -1 & i_j = 0 \\ -2 & i_j = 1 \end{cases} \tag{4.6}$$

This defines f_r for all possible states, because of the assumption that the problem is solvable. Note that f_r is well-defined, because considering all elements l from $\mathscr{C}$ determines considering all possible abstract states $\hat{\mathbf{x}}_l$, and for each such abstract state, we define the reinforcement

function for every possible underlying state $\mathbf{x} \in [\hat{\mathbf{x}}]$.

We have thus defined a MDP: $\langle \mathcal{X}, \mathcal{U}, f_x, f_r \rangle$, with the space of measurables $\mathcal{X}$ defined in a factored form: $i_1 \times i_2 \times \ldots \times i_{m+M}$, and action set $\mathcal{U}$, transition function f_x and reinforcement function f_r as above. After m steps of incremental abstraction refinement process we reach an incorrect abstraction mapping, ϕ_m, defined by the index set $\mathcal{I}_m = \{i_1, i_2, \ldots, i_m\}$ (i.e. the abstraction ignores measurables $i_{m+1}, \ldots, i_{m+M}$). The problem of further refinement of this abstraction is a result of transformation of the initial *set cover problem* instance. This transformation is linear in terms of number of elements N, and number of subsets M, each multiplied by the number of its elements. This gives complexity smaller than $O(N + MN)$. This proves that the variable selection state abstraction problem is *NP-hard*. This also additionally implies that the decision version of this problem (e.g. deciding whether a correct abstraction mapping exist with at most k measurables) is *NP-complete* [61]. □

This proves the first sub-thesis of this work, namely that:

the problem of finding the model-preserving variable selection abstraction is an inapproximable NP-hard problem.

4.4 Ambiguity functional

In this section we define a functional that allows to qualify to which extent the *model-preserving* abstraction (Sec. 3.5) constraints are violated (see (3.15)) by a given abstraction function.

Our goal is to define a functional that allows to compare abstractions in a way suitable for practical application, in particular in continuous domains. The authors of [40] use the term *unstable partition* to describe the abstract states (partitions) that violate the *bisimulation* principle. An equivalent notion of not preserving environment's model is used in [72]. In these terms, *unstable partitions* are abstract states that violate the constraints of the model-preserving abstractions. Similarly to [31], we propose to measure how much these constraints are disturbed. To achieve this, we introduce the *ambiguity functional*. Unlike metrics defined in [30] and [16], we consider deterministic systems only, so there is no need to estimate the distance between probability distributions. Despite this simplification, it is still possible for our approach to work in the nondeterministic case, as we demonstrate in the additional experimental evaluation in Sec 7.2.

Psychological context The intuition behind the ambiguity functional we are about to propose is to relate it to the way animals judge the quality of their internal model of the surrounding world. The more often unpredictable situations occur, the less accurate the model is. In case of the proposed algorithm, the more the constraints of model-preserving abstraction are being violated, the more information about system's behaviour is lost. In psychology such situation makes animals strive to obtain closure, i.e. an explanation of the ambiguous situation [48]. In case of the proposed algorithm, such explanation is obtained in the form of including a missing measurable in the abstraction, which makes the surrounding world less unpredictable. □

The definition of the *ambiguity functional* is based on the definition of *model-preserving* abstraction in (3.15), and is related to the problem statement in (4.2). The intent is to measure to which extent the plant is behaving in an nondeterministic way, in the context of a given, particular abstraction. The term *behave* refers to the transition and reinforcement functions (f_x, f_r), which, for a given control u_t, map a measurement $\mathbf{x_t}$ to the next measurement $\mathbf{x}_{t+1}$ and the reinforcement value $r_{t+1} \in \mathcal{R}$. Consider a situation where an abstract state groups together measurements that result in a different outcome for a particular action. When the system reaches these states, they appear equal - because the abstraction maps them to a single abstract state. Thus the system will appear to be indeterministic, because after applying the same action, it will reach different abstract states and/or emit different reinforcements. This will happen if the abstraction ignores a non-redundant measurable that determines the outcome of the transition function and/or the reinforcement function. We say that an abstract state exhibiting such behaviour is *ambiguous*. In other words, when the plant is in an abstract state $\hat{\mathbf{x}}$, it can be in any of the states $\mathbf{x}^1 \in [\hat{\mathbf{x}}], \mathbf{x}^2 \in [\hat{\mathbf{x}}], \ldots$, which can have different successors for the same action. The more different abstract states and reinforcement values can be reached via the transition/reinforcement functions from an ambiguous abstract state, the more ambiguous it is. This is visualized in Fig. 4.2. We require that the *ambiguity functional*, we are going the define, has the following properties:

Properties 1.

1. Abstractions that yield abstract states that are more ambiguous should determine larger value of the ambiguity functional.

2. The ambiguity functional must yield its lowest possible value for *model-preserving* abstractions.

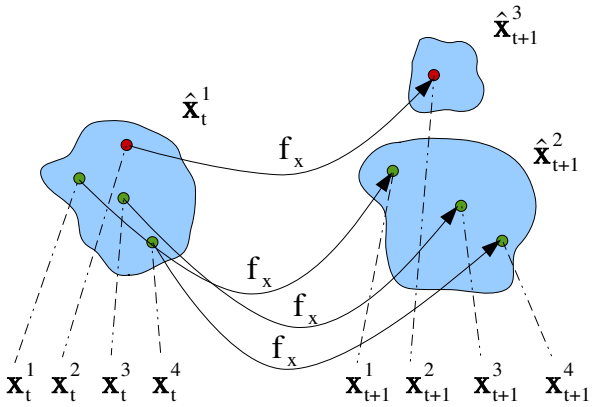

Figure 4.2: An ambiguous abstract state $\hat{\mathbf{x}}^1$. When the plant is in an underlying state that produces measurements being represented by this abstract state, it can be, in fact, in any of the underlying states that produce measurements: $\mathbf{x}^1_t, \mathbf{x}^2_t, \mathbf{x}^3_t, \mathbf{x}^4_t$. In the next step, the transition function f_x transforms one of them into one of next underlying states, producing one of measurements: $\mathbf{x}^1_{t+1}, \mathbf{x}^2_{t+1}, \mathbf{x}^3_{t+1}, \mathbf{x}^4_{t+1}$. If they all would have been represented by a single abstract state the abstraction would be a correct model-preserving abstraction. In this case it is not - because of $\mathbf{x}^2$ the model behaves nondeterministically - being in $\hat{\mathbf{x}}^1_t$ it sometimes goes to $\hat{\mathbf{x}}^2_{t+1}$, and sometimes to $\hat{\mathbf{x}}^3_{t+1}$. The same reasoning applies to the reinforcement function f_r.

3. Given two abstractions $\phi_{\mathcal{J}}$ and $\phi_{\mathcal{K}}$ with index sets that differ by one variable the ambiguity functional must yield lower value for the abstraction with the larger index set. Thus, the following must hold:

$$\|\mathcal{J} \setminus \mathcal{K}\| = 1 \implies A(\phi_{\mathcal{J}}) \leq A(\phi_{\mathcal{K}}) \tag{4.7}$$

These properties guarantee that the correct abstraction, if it exists, will be found.

Theorem 4.4.1. The incremental process presented in Sec. 4.2 based on the ambiguity functional (4.10) that has the properties (1) always finds a model-preserving variable selection abstraction, if such abstraction exists.

Proof. Denote with $\mathcal{J}^j$ an index set with j indices and consider a state abstraction problem with n given measurables. Note that, $\phi_{\mathcal{J}^0} = \phi_{\emptyset}$ denotes the null abstraction and $\phi_{\mathcal{J}^n}$ denotes the trivial abstraction (both defined in Sec. 3.5). Let us assume that $\phi_{\mathcal{J}^n}$ (all measurables, so no abstraction at all) preserves the model, and that $\phi_{\mathcal{J}^1}$ for one particular set $\mathcal{J}^1$ does not preserve model. Assume that the ambiguity functional can be 0 at minimum. This means that $A(\phi_{\mathcal{J}^1}) > 0$. Then, including measurables into the index set one by one, will induce subsequent

abstractions, $\phi_{\mathcal{F}^2}, \phi_{\mathcal{F}^3}, \ldots$. Because these measurements are being grouped into abstract states only as a result of some measurables being ignored by the abstraction mapping, the sequence of values of the ambiguity functional for the considered sequence of index sets will be decreasing, namely:

$$A(\phi_{\mathcal{F}^1}) \geq A(\phi_{\mathcal{F}^2}) \geq \ldots \geq A(\phi_{\mathcal{F}^n}) \tag{4.8}$$

Therefore, either for some index set $\mathcal{F}^i$, $A(\phi_{\mathcal{F}^i}) = 0, \mathcal{F}^i \neq \mathcal{F}^n$, so $\phi_{\mathcal{F}^i}$ will preserve the model or all measurables will be eventually included, and the abstraction $\phi_{\mathcal{F}^n}$ preserves the model by assumption. Thus, such process guarantees that a *model-preserving* abstraction will be reached, exactly in the same way as in the reasoning presented in Sec. 4.2. Note that also as in reasoning presented in Sec. 4.2 this does not guarantee that the result will be optimal in terms of the size of its index set $\mathcal{F}$. □

Now we will propose a definition of the ambiguity functional in a general form that meets these requirements.

To calculate the value of the ambiguity functional for an abstraction function we choose to take into account the most ambiguous state, i.e. the resulting quantity will be a result of maximization over all abstract states and all possible actions (controls): $\max_{\hat{\mathbf{x}} \in \hat{\mathcal{X}}, u \in \mathcal{U}}$. Given a particular abstract state $\hat{\mathbf{x}}$ we must examine all measurements aggregated by this abstract state: $\mathbf{x} \in \hat{\mathbf{x}}$ to get all possible outcomes of the transition function f_x and the reinforcement function f_r. Thus, for a given abstraction mapping ϕ, when considering a particular abstract state $\hat{\mathbf{x}}$, we must iterate over all elements of the inverse mapping: $\phi^{-1}(\hat{\mathbf{x}})$ (recall from (3.5) that the result of ϕ^{-1} is a set: $[\hat{\mathbf{x}}]$). Figure 4.3 presents a graphical representation of this process, taking into account only the transition function f_x. For the reinforcement function f_r the bottom part of the diagram would contain only points representing different real numbers (reinforcement values). For convenience, let us introduce the following symbol that given a measurement vector and a reinforcement value pair, produces a pair of abstract state for the given measurement and the unaffected value of the given reinforcement:

$$\bar{\bar{\phi}}(\mathbf{x}, r) = (\phi(\mathbf{x}), r) \tag{4.9}$$

Then, the *abstraction ambiguity functional* can be defined as follows, summarizing the description presented above:

$$A(\phi) = \max_{\hat{\mathbf{x}} \in \hat{\mathcal{X}}, u \in \mathcal{U}} \Lambda(\left\{ \bar{\bar{\phi}}(f_x(\mathbf{x}, u), f_r(\mathbf{x}, u)) : \mathbf{x} \in [\hat{\mathbf{x}}] \right\}) \tag{4.10}$$

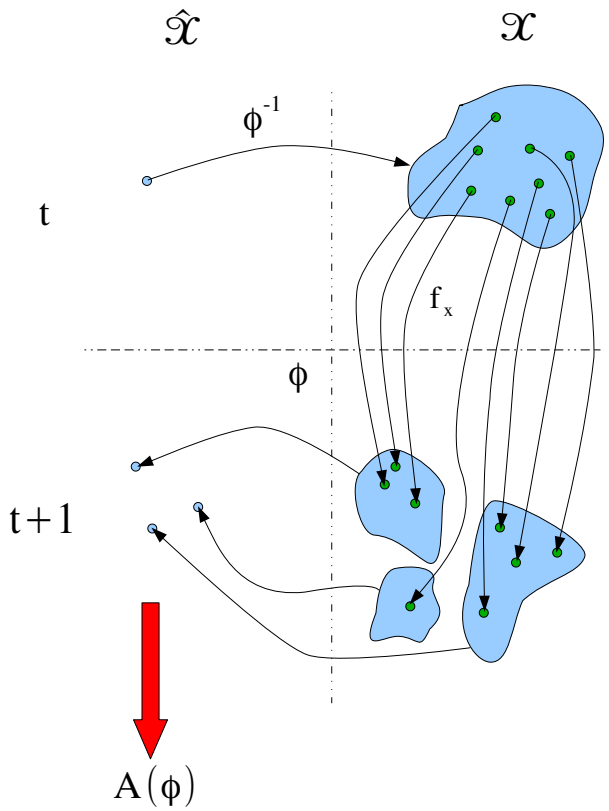

Figure 4.3: The general concept of calculating the ambiguity functional for a given abstraction function ϕ. The left side of the picture represents the abstract space $\hat{\mathcal{X}}$ induced by the abstraction function ϕ. The corresponding view of the $\mathcal{X}$ space is presented at the right side. Points represent vectors: abstract states on the left side, measurements on the right side. Blue shapes represent groups of measurements belonging to the same equivalence class, i.e. the same abstract state. The top of the picture presents the measurements at time t, before applying the transition function f_x. The bottom represents the time instant $t+1$, i.e. after f_x was applied.

The concrete definition of the Λ function depends on whether we are dealing with discrete or continuous transition and reinforcement functions. Its purpose is to evaluate all possible outcomes of the transition and reinforcement functions, for all measurements represented by one abstract state. For the properties of the ambiguity functional (1) to hold, the Λ function is required to satisfy the following:

Properties 2.

1. Abstract states that are more ambiguous should determine larger value of the Λ function.

2. Unambiguous abstract states should yield the lowest possible value of the Λ function.

These properties of the Λ function ensure that properties formulated for the ambiguity functional in (1) will be satisfied.

Theorem 4.4.2. The properties (2) of the Λ function ensure that the ambiguity functional (4.10) has the properties (1).

Proof. The first property from (2) ensures directly that the first property from (1) is satisfied.

Because model-preserving abstractions yield only unambiguous abstract states, the second property from 2 also makes the second property from (1) satisfied.

The first property from (2) also ensures that the third property from (1) is satisfied. This can be proven by contradiction. Let us assume we are given a Λ function that has properties defined in (2), two abstraction functions $\phi_{\mathcal{I}}$, $\phi_{\mathcal{K}}$, where $\|\mathcal{I} \setminus \mathcal{K}\| = 1$, and that $A(\phi_{\mathcal{I}}) > A(\phi_{\mathcal{K}})$, i.e. the third condition from (1) is false. The reasoning leading to a contradiction is as follows.

$A(\phi_{\mathcal{I}}) > A(\phi_{\mathcal{K}})$ implies that there exists an abstract state produced by abstraction $\phi_{\mathcal{I}}$ that is more ambiguous (i.e. transition and reinforcement functions transform it at time t, to a larger number of different abstract states at time $t+1$) than any other abstract state produced by abstraction $\phi_{\mathcal{K}}$. Let us call this abstract state $\hat{\mathbf{x}}$. This abstract state represents an equivalence class of measurements that have equal values of measurables indicated by indices from the set $\mathcal{I}$. Because of the assumption that $\|\mathcal{I} \setminus \mathcal{K}\| = 1$ the index set $\mathcal{K}$ has one index less than $\mathcal{I}$. This imposes more relaxed condition on aggregation of the same measurements that are represented by $\hat{\mathbf{x}}$, because lesser number of measurables need to be equal to be aggregated into one equivalence class. This in turn implies that there exists an abstract state, $\hat{\mathbf{y}}$ yielded by $\phi_{\mathcal{K}}$ that represents no less measurables than $\hat{\mathbf{x}}$. Thus, it is at least as much ambiguous as $\hat{\mathbf{x}}$. Because of the maximization term in (4.10) this implies that $A(\phi_{\mathcal{K}})$ can not be less than $A(\phi_{\mathcal{I}}$, which contradicts the assumption that $A(\phi_{\mathcal{I}}) > A(\phi_{\mathcal{K}})$. □

In Chapters 5 and 6 we propose concrete definitions of the Λ function, which satisfy properties (2) and are suitable for discrete and continuous domains respectively.

4.5 Incremental state abstraction paradigms

Before we introduce concrete definitions of the ambiguity functional, let us analyze in general, possible approaches to the incremental state abstraction problem. In this section we argue that the process presented in the previous Sec. 4.2, going from *coarser* abstractions towards *finer*

ones is in general better than the opposite approach, which would start with no abstraction at all, and would successively remove measurables. We call the first approach the *bottom-up* approach, and the second one the *top-down* approach. We also analyze a *binary-search* paradigm, an idea based on the fact the the ambiguity functional is monotonous with regard to the number of variables preserved by a given abstraction.

Psychological context The idea of preferring the *bottom-up* approach to the *top-down* approach is also related to the hypothesis that people learn incrementally, successively improving their mental model of the surrounding world, basing on the knowledge they currently have [11]. Also, simpler models are preferred over the complicated ones [95]. □

We assume that the given model contains redundant measurables. As mentioned above, the bottom-up approach starts with an assumption that no measurables are needed to describe the *system's behaviour* (see Chapt. 3). If this assumption turns out false, one measurable is added and the process is repeated. We discuss how to verify such assumptions and how to choose measurables to insert in the following sections of this chapter. The other one, the top-down approach, starts with an assumption that all measurables are relevant and then verifies if any of them can be abandoned. In the following sections we also show a way of verifying whether a measurable is redundant.

We argue that the bottom-up approach is better, *in general*, in terms of processing time. We say *in general*, because the calculations presented in this section assume evaluating all possible abstractions with given number of measurables, which is not the case in practice. However, the reasoning presented here gives some intuition and insight on what the difference between the bottom-up and top-down approaches is.

Let us denote with n the number of measurables in the given model and the number of measurables in the smallest sufficient set of measurables with m ($1 \leq m \leq n$). Let us assume that the time complexity of verifying whether the set of measurables is sufficient or that contains redundant measurables is exponential: $O(e^i)$ with i being the number of measurables. Then, the time complexity of the bottom-up approach is:

$$\text{BottomUp}(n,m) = \sum_{i=1}^{m} \binom{n}{i} \exp(i) \tag{4.11}$$

And the time complexity for the top-down approach is:

$$\text{TopDown}(n,m) = \sum_{i=m}^{n} \binom{n}{i} \exp(i) \tag{4.12}$$

Let us assume that for each n, all cases (i.e. number of measurables in the solution - m) are equally probable. Then, using Eqs. (4.11) and 4.12 average processing time for all possible cases can be estimated as:

$$\text{BottomUp}(n) = \frac{\sum_{m=1}^{m=n} (\text{BottomUp}(n,m))}{n} \tag{4.13}$$

for the bottom-up approach and analogously:

$$\text{TopDown}(n) = \frac{\sum_{m=1}^{m=n} (\text{TopDown}(n,m))}{n} \tag{4.14}$$

for the top-down approach.

Solution of (4.13) is as follows:

$$\text{BottomUp}(n) = \frac{\sum_{i=1}^{i=n} (n-i+1)\binom{n}{i}\exp(i)}{n} = \frac{n(1+e)^{n-1} + (1+e)^n - (n+1)}{n} \tag{4.15}$$

And the solution of (4.14):

$$\text{TopDown}(n) = \frac{\sum_{i=1}^{i=n} i\binom{n}{i}\exp(i)}{n} = \frac{e(1+e)^{n-1}n}{n} \tag{4.16}$$

To verify which approach is more time consuming on average consider a function:

$$f(n) = \frac{TopDown(n)}{BottomUp(n)} = \frac{e(1+e)^{n-1}n}{n(1+e)^{n-1} + (1+e)^n - (n+1)} \tag{4.17}$$

Figure 4.4 presents a plot of this function for $n = 1, \ldots, 500$. It is strictly above 1 for $n > 1$, has no roots and its limit as n approaches infinity is also greater than 1:

$$\lim_{n\to\infty} f(n) = e \tag{4.18}$$

This proves that the *bottom-down* approach is on average better than top-down. Suppose now we happen to have some domain knowledge about the problem being solved, and can suspect that the solution is likely to contain most of the given measurables (i.e. $m \geq \frac{n}{2}$). Then, the same reasoning can be applied. The bottom-up approach starts with all possible combinations of $\frac{n}{2}$ measurables and incrementally includes measurables up to n, and the top-down approach starts as before from n measurables and attempts to remove measurables, but no more than $\frac{n}{2}$. In this case, the equations will take the following form:

$$\text{BottomUp}(n,m) = \frac{\sum_{i=\frac{n}{2}}^{n} i\binom{n}{i}\exp(i)}{n} \tag{4.19}$$

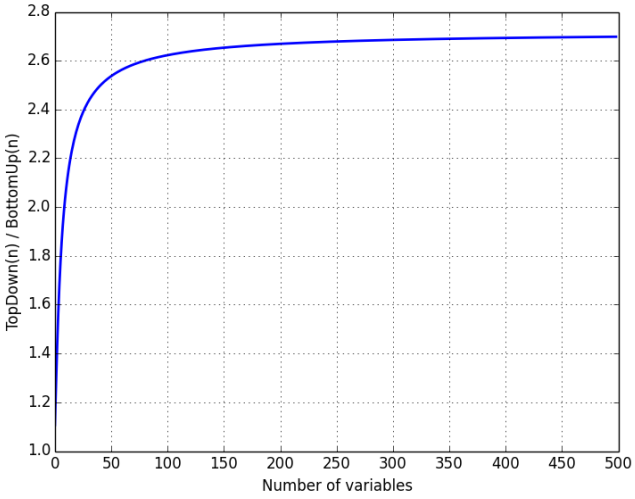

Figure 4.4: Comparison of the estimated computational complexity of bottom-up and top-down incremental approaches to the state abstraction problem. The complexity ratio is greater than 1, which suggests that in general the bottom-up approach is better.

$$\text{TopDown}(n,m) = \frac{\sum_{i=\frac{n}{2}}^{n} (i - \frac{n}{2} + 1)\binom{n}{i} \exp(i)}{n} \tag{4.20}$$

Also, consider an opposite case: we happen to know that the solution contains no more than $\frac{n}{2}$ measurables. Then, the equations for the bottom-up and top-down will look as follows:

$$\text{BottomUp}(n) = \frac{\sum_{i=1}^{\frac{n}{2}} (\frac{n}{2} - i + 1)\binom{n}{i} \exp(i)}{n} \tag{4.21}$$

$$\text{TopDown}(n) = \frac{\sum_{i=1}^{\frac{n}{2}} i\binom{n}{i} \exp(i)}{n} \tag{4.22}$$

Solutions of these four equations involve hyper-geometric functions and thus are hard to analyze. However, they can be calculated numerically. Plots presented in Fig. 4.5 and Fig. 4.6 suggest that the top-down approach is better under the assumption that the solution contains at least $\frac{n}{2}$ measurables, and that bottom-up is better under the assumption solution that the solution contains no more than $\frac{n}{2}$. In this work we make no assumptions regarding the number of measurables in the solution, and therefore we focus on the bottom-up paradigm. However, the presented approach can be used in both ways. Another possible order of evaluating abstractions can be based on the monotonicity property of the ambiguity functional, presented in (4.7).

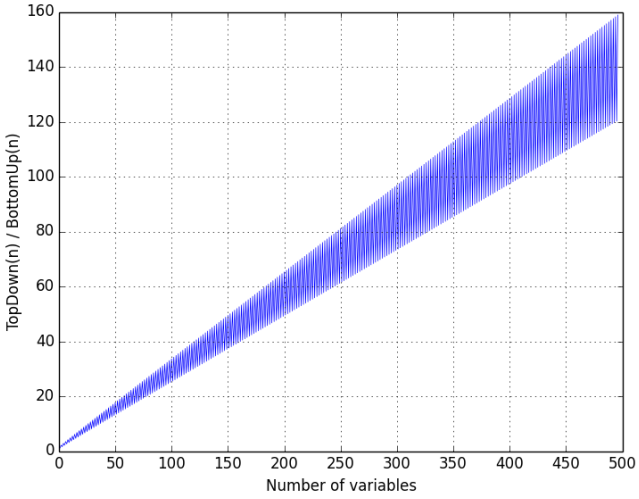

Figure 4.5: Comparison of the estimated time complexity of bottom-up and top-down incremental approaches to the state abstraction problem under the assumption that the solution has no more than $\frac{n}{2}$ measurables. Similarly as in Fig. 4.4, the bottom-up approach is better in terms of computational complexity.

Denote with ϕ_n a set of state abstractions defined by all possible index sets of cardinality n. If there is an abstraction $\phi_n \in \phi_n$ for which $A(\phi_n) = 0$, then by (4.7), it means that the optimal solution has at most n measurables. If on the other hand all abstractions in ϕ_n have ambiguity functional greater than 0 then the optimal solution has more than n measurables. Sets ϕ_n for all possible numbers of measurables can be thus ordered by the value: $\min_{\phi_n} A(\phi_n)$. Because there exist a total ordering of this set, we can perform a binary search to identify the number of measurables in the optimal solution, and the solution itself. For a problem with n possible measurables the process starts with $\frac{n}{2}$ measurables and calculates the ambiguity functional for all $\binom{n}{\frac{n}{2}}$ possible abstractions. If at least one yields 0, the process continues, recurring for $\frac{n}{4}$ measurables. Otherwise, all abstractions with $\frac{3n}{4}$ are verified, and so on. This search process is presented in Fig. 4.7. Each tree node represents a decision point, in which all abstraction with number of measurables indicated with the number inside the node are being evaluated. If any of them yields the value of the ambiguity functional equal to 0 the left sub-node is chosen for the next step. Otherwise, if all abstractions have ambiguity functional greater than 0 the path to the right is taken.

Similarly to the reasoning from Sec. 4.5 let us try to estimate the computational complexity of such process. As before, assume that the cost of evaluating an abstraction mapping with k

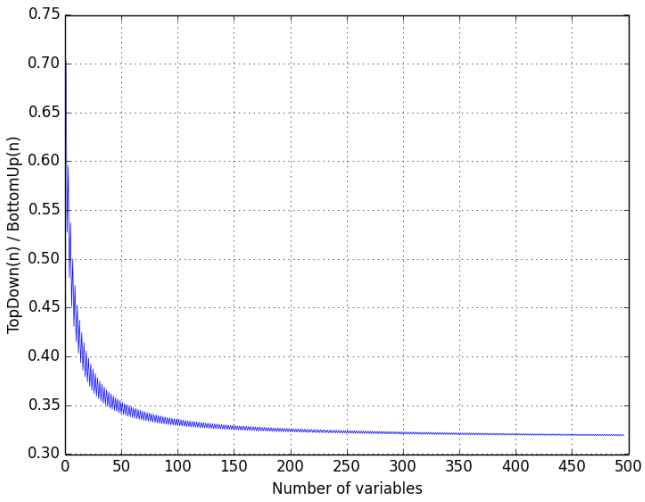

Figure 4.6: Comparison of the estimated time complexity of bottom-up and top-down incremental approaches to the state abstraction problem under the assumption that the solution has no less than $\frac{n}{2}$ measurables. In this case, as opposed to the previous comparisons presented in Figs. 4.4, 4.5, the top-down approach is better in terms of computational complexity.

measurables is $O(e^k)$ and that all solutions are equally probable. On each level i of the search tree the number of possible nodes is 2^{i-1}. The number of measurables in each subsequent node, looking from left to right on each level is: $\frac{(2j-1)n}{2^i}$, where j is the ordinal number of the node, $1 \leq j \leq 2^{i-1}$. For k measurables, there are $\binom{n}{k}$ possible variable selection state abstractions, each with evaluation cost of order e^k. For convenience denote by $k(i,j)$ the number of measurables in j-th node on i-th level, i.e. $k(i,j) = \frac{(2j-1)n}{2^i}$. Thus, we can estimate the average complexity of evaluating abstractions on level i:

$$\frac{\sum_{j=1}^{2^{i-1}} \binom{n}{k(i,j)} \exp(k(i,j))}{2^{i-1}} \tag{4.23}$$

Because the binary search tree for n elements has height of $log_2(n)$ the equation for average complexity of the whole process is straightforward:

$$Bin(n) = \sum_{i=1}^{log_2(n)} \frac{\sum_{j=1}^{2^{i-1}} \binom{n}{k(i,j)} \exp(k(i,j))}{2^{i-1}} \tag{4.24}$$

We compare numerically this paradigm with the bottom-up paradigm evaluated in Sec. 4.5. To do this, we plot the following function, for the number of measurables $n = 1, \ldots, 500$:

$$f(n) = \frac{BottomUp(n)}{Bin(n)} \tag{4.25}$$

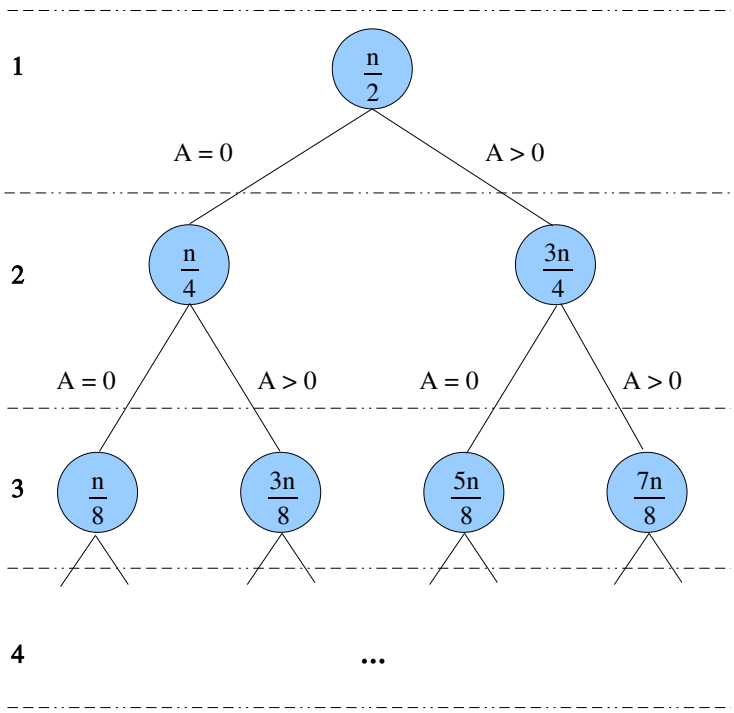

Figure 4.7: A binary search tree, illustrating the binary-search approach to the incremental abstraction problem, for a n-variable system. Numbers in nodes denote the number of measurables in the evaluated abstractions. $A > 0$ figuratively denotes that all abstractions have the value of the ambiguity functional greater than 0. $A = 0$ indicates the scenario where at least one abstraction is correct, i.e. its ambiguity functional is 0.

where the $BottomUp(n)$ function was defined in Sec. 4.5. The result of this evaluation is presented in Fig. 4.8. In can be seen that for large number of measurables the binary-search approach should on average be faster. The Fig. 4.9 presents the same graph focused on the area with fewer measurables. It can be seen that when there are no more than 20 measurables the approaches are comparable. In Fig. 4.10 and the Fig. 4.11 we present graphs under the assumption that the solution contains at least $\frac{n}{2}$ measurables and at most $\frac{n}{2}$ measurables resp. To obtain the formula for the first of these cases, we need to omit the first level from the outer sum, and include only the right part of the search tree in the inner sum. The formula for the average complexity in the first case is thus as follows:

$$Bin(n) = \sum_{i=2}^{log_2(n)} \frac{\sum_{j=2^{i-2}+1}^{2^{i-1}} \binom{n}{k(i,j)} \exp(k(i,j))}{2^{i-2}} \tag{4.26}$$

Similarly, for the second case. We omit the first level of the tree, and sum only the nodes from

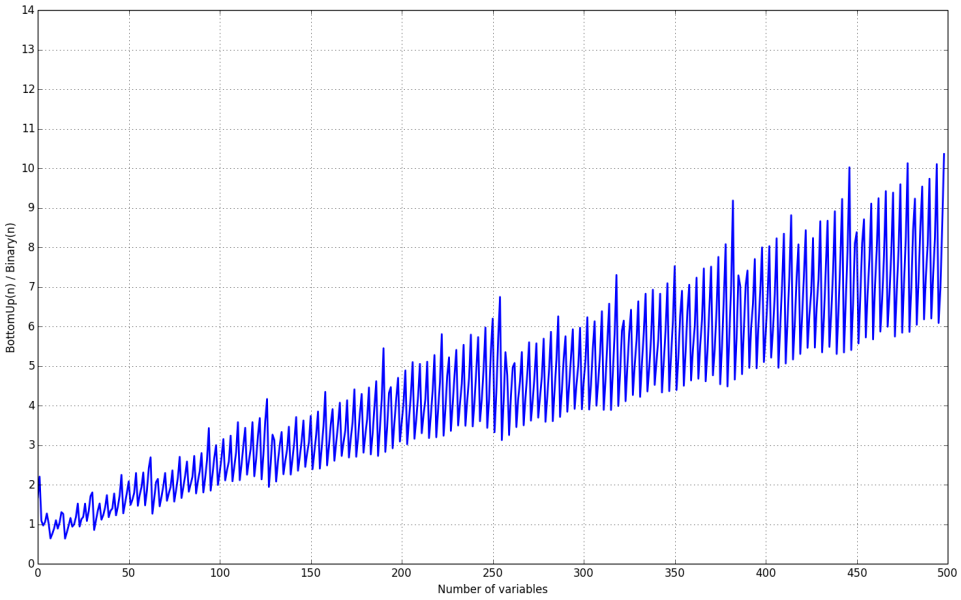

Figure 4.8: Comparison of the estimated time complexity of bottom-up and binary-search approaches to the state abstraction problem. In general the binary-search approach has lower complexity. A more detailed view for the part of the graph when the comparison is not clear is presented in Fig. 4.9.

the left sub-tree:

$$Bin(n) = \sum_{i=2}^{log_2(n)} \frac{\sum_{j=1}^{2^{i-2}} \binom{n}{k(i,j)} \exp(k(i,j))}{2^{i-2}} \tag{4.27}$$

In both cases the binary-search approach is better. Because we do not assume any additional knowledge about the solution, and deal with problems with the number of measurables less than 20, we focus on the bottom-up approach. Also, as explained in the following sections, the one-step incremental approaches give possibility to make mistakes when estimating the value of the ambiguity functional. In case of the binary-search approach, exact values must be known at each step.

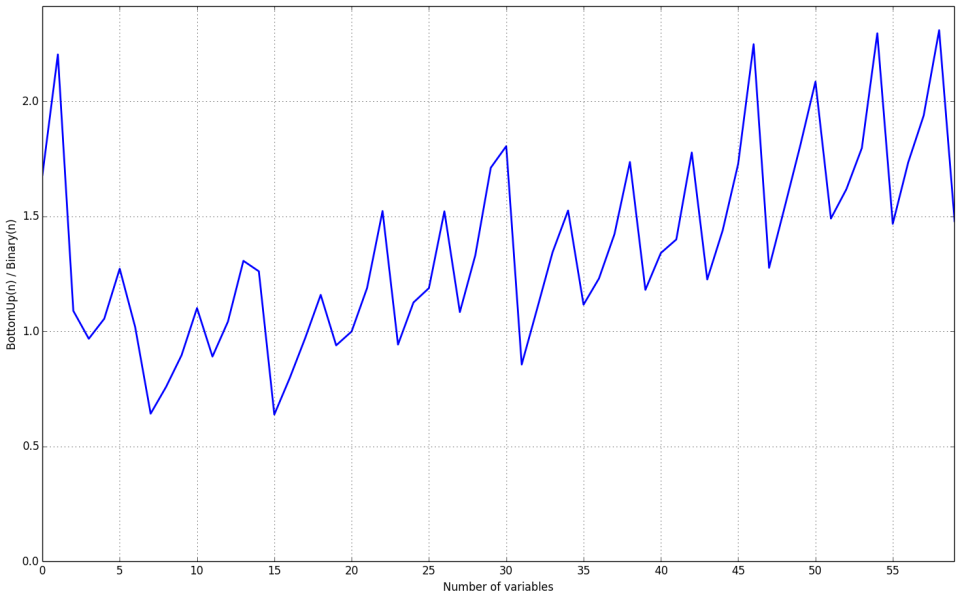

Figure 4.9: Comparison of the estimated time complexity of bottom-up and binary-search approaches to the state abstraction problem for no more than 60 measurables. For single problem instances the bottom-up approach yields lower complexity, but most of the time the binary-search approach is better.

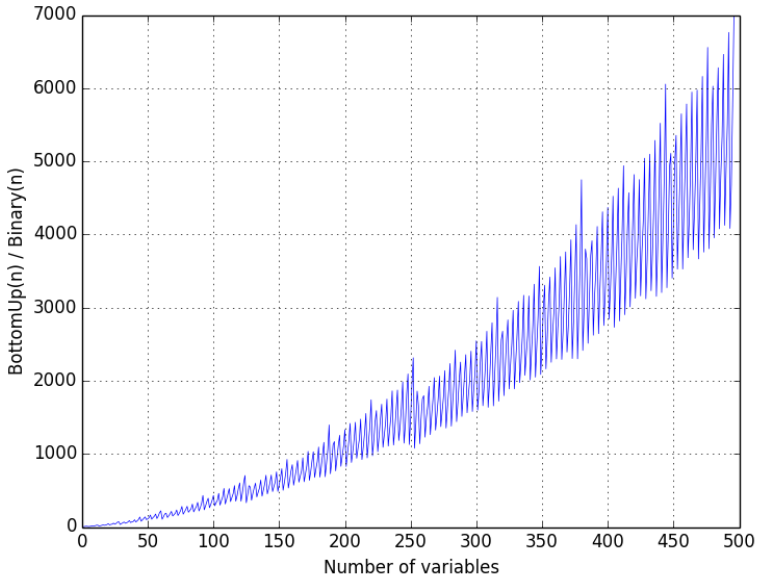

Figure 4.10: Comparison of the estimated time complexity of bottom-up and binary-search approaches to the state abstraction problem under the assumption that the solution has no less than $\frac{n}{2}$ measurables. As in previous cases, the binary-search approach has better complexity.

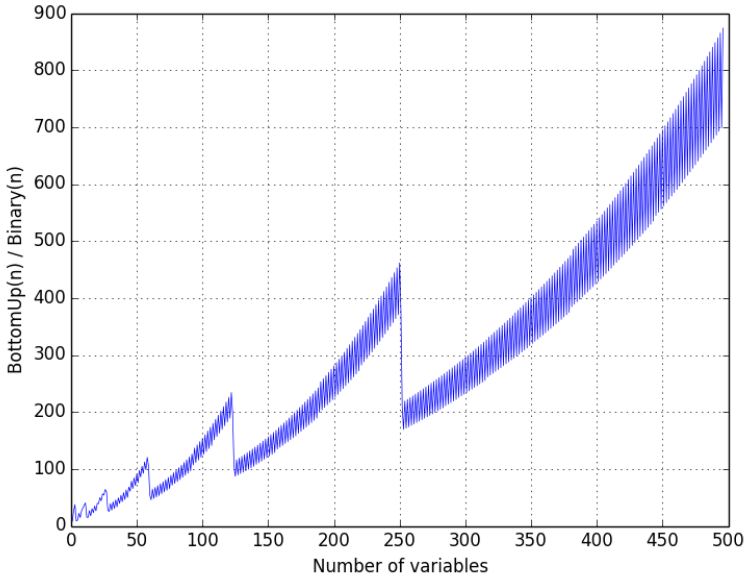

Figure 4.11: Comparison of the estimated time complexity of bottom-up and binary-search approaches to the state abstraction problem under the assumption that the solution has no more than $\frac{n}{2}$ measurables. As in previous cases, the binary-search approach has better complexity.

Chapter 5

Ambiguity resolving approach for discrete domains

In this chapter we propose a concrete definition of the ambiguity functional for discrete transition and reinforcement functional, and present a state abstraction algorithm that is based on this notion. Analysis of the proposed approach in the context of discrete state domains helps in better understanding of the underlying problems. It also provides a good starting point for applying the algorithm to continuous state domains, which is presented in the following chapter, namely Chapt. 6.

5.1 Ambiguity functional for discrete transition and reinforcement functions

The idea presented in this chapter is also described by Papis and Pacut in [92]. In the discrete case evaluating a set of abstract states into a number is quite easy. It is sufficient to count the number of different abstract state-reinforcement pairs in the given set, Λ function's argument. Because they represent the number of possible outcomes from an evaluated abstract state, the more of them there is, the more ambiguous the abstract state is. The ambiguity function Λ, which determines the means of calculating the ambiguity functional for the discrete case can be defined as follows::

$$\Lambda(\hat{\mathfrak{X}}_{t+1}) = \|\hat{\mathfrak{X}}_{t+1}\| - 1 \tag{5.1}$$

The function's argument is denoted as $\hat{\mathfrak{X}}_{t+1}$ figuratively: $\hat{\mathfrak{X}}$ indicates that the set contains elements from the abstract space (and reinforcement values), and $t+1$ subscript indicates that these elements are a result of the transition and reinforcement functions. The last term, 1, is subtracted from the result so that an unambiguous measurement will yield 0, and thus correct model-preserving abstractions that have no ambiguous measurements will yield ambiguity functional's value of 0. This definition satisfies properties (2). Abstract states that are more ambiguous will make transition function and reinforcement functions yield more different elements, which will be contained in $\hat{\mathfrak{X}}_{t+1}$. On the other hand, unambiguous abstract states will yield only one element and this is the lowest possible number of successor abstract states.

Because of properties (1) and (2) minimizing functional A with the inner Λ function defined as above is equivalent to transforming the index set in the same way as the H function described in Sec. 4.2 and reaching the fixed-point of this transformation. As presented in Example 4.2.1, these properties do not guarantee an optimal solution to the non-linear problem presented in (4.2). In the following sections we present some possible heuristics that can alleviate this sub-optimality.

Thought Experiment 5.1.1. To illustrate introduced notions with a simple example, we will now calculate the values of the discrete ambiguity functional for some abstraction functions. Consider the Discrete Labyrinth environment. Recall the complete set of measurables: $x, y, x+y, |x-y|, xy$ and consider the following abstractions:

1. $\phi^1 = \phi_{\{1,4\}}$ (selected measurables: $y, |x-y|$)
2. $\phi^2 = \phi_{\{1,3,4\}}$ (selected measurables: $y, x+y, |x-y|$)
3. $\phi^3 = \phi_{\{1,3\}}$ (selected measurables: $y, x+y$)

For the first abstraction function, ϕ^1, the maximum possible value of the ambiguity functional for one abstract state (see (5.1)) is 2, because in this abstraction at most two measurements can be represented by a single abstract state (at most two possible (x,y) solutions for the pair of equations $y = c^1, |x-y| = c^2$, where c^1, c^2 are some arbitrary constants). Indeed, this value can be reached given the pair of states considered in the previous Example 4.2.1, namely: $\mathbf{x}^1 = (x = 2, y = 2, \ldots)$ and $\mathbf{x}^2 = (x = 3, y = 1, \ldots)$ for action $u = \text{LEFT}$. Because in the first case the agent will successfully move to the left ($(x = 1, y = 2, \ldots)$), and in the second case will hit the wall, the set $\hat{\mathfrak{X}}_{t+1}$ will contain two different state-reinforcement pairs: $(\mathbf{x}^1_{t+1}, r^1_{t+1}) = ((x =$

$1, y = 2, \ldots), -1)$ and $(\mathbf{x}^2_{t+1}, r^2_{t+1}) = ((x = 3, y = 1), -5)$, which form two different abstract state-reinforcement pairs: $(\hat{\mathbf{x}}^1, r^1_{t+1}) = ((y = 2, |x-y| = 1), -1)$ and $(\hat{\mathbf{x}}^2, r^2) = ((y = 1, |x-y| = 2), -5)$. Thus the Eq. (5.1) will yield 2. Because this is the worst case scenario in terms of the ambiguity, $A(\phi^1) = 2 - 1 = 1$ (according to (5.1)).

Similar reasoning to the one in Example 4.1.1 can be presented for abstraction ϕ^2. There exist a bijection between abstract vectors created by this abstraction and environment's underlying state (x, y), namely:

$$g((y, x+y, |x-y|)) = (x+y-y, y)$$

Other possible way to look at it: for any nonnegative constants c^1, c^2, c^3 the set of equations:

$$\begin{aligned} y &= c^1 \\ x+y &= c^2 \\ |x-y| &= c^3 \end{aligned}$$

is overdetermined. Thus there will be always only one abstract state in the set $\hat{\mathfrak{X}}_{t+1}$, and therefore $A(\phi^2) = 1 - 1 = 0$ (according to (5.1)).

Consider now the last abstraction: ϕ^3. It can be noted again that a bijection exists that can map between abstract states and measurements. Thus, by analogy to the previous reasoning $A(\phi^3) = 1 - 1 = 0$.

□

5.2 The proposed algorithm for discrete state domains

With the concrete definition of the ambiguity functional presented in Eqs. (4.10), (5.1) we have means of comparing the abstraction functions. Because the proposed definition satisfies the properties described in lists (1) and (2) we can employ it in the incremental process presented in Sec. 4.2. The proposed algorithm is presented in Listing 1. In the following section we evaluate this algorithm using the knowledge about the transition and reinforcement functions of the environment to be able to calculate the exact values of the ambiguity functional. Having shown that the algorithm works, in Sec. 5.4 we propose a method of estimating the values of the ambiguity functional from a random set of samples. While the knowledge about transition and reinforcement functions is still used to estimate the values of the ambiguity functional faster,

Algorithm 1 Incremental state abstraction algorithm

1: $\phi \leftarrow \phi_\emptyset$
2: Calculate $A(\phi)$
3: **for** ϕ^i in all possible abstractions with one extra measurable **do**
4: Calculate $A(\phi^i)$
5: $\phi^{\text{ext}} \leftarrow arg \min_{a \in \{\phi^{\text{ext}}, \phi^i\}} A(a)$
6: **end for**
7: **if** $A(\phi^{\text{ext}}) < A(\phi)$ **then**
8: $phi \leftarrow \phi^{\text{ext}}$
9: GOTO 2
10: **else**
11: STOP.
12: **end if**

it is not directly used to calculate the results. Chapter 7 contains experiments that evaluate the algorithm on a purely random set of samples, i.e. without the simulator assumption, described in Sec. 3.2. The experiments in Sec. 5.4 show that it is not sufficient to only include measurables in the abstraction step by step. Therefore, in the next section, namely Sec. 5.5, we present additional mechanism that also removes measurables from the abstraction. This in turn leads to evaluation of too many possible abstractions. Thus, we propose to keep track of the number of measurables in the smallest model-preserving abstraction found, and do not evaluate any larger abstractions until it is needed. This significantly reduces the number of evaluated abstractions. The final algorithm is presented in Listing 3.

5.3 Evaluation of the algorithm with exact values of the ambiguity functional

Computer Experiment 5.3.1. To verify correctness of the described process let us evaluate its simple implementation on a simple problem. Consider again the Discrete Labyrinth domain with 100 measurables. First seven of them are: $x, y, x+y, |x-y|, xy, q_x = \lfloor \sqrt{x} \rfloor, q_y = \lfloor \sqrt{y} \rfloor$ and the rest consists of 93 randomly generated functions. For better readability let us define the

following function for convenience:

$$\mathrm{NZ}(x) = \begin{cases} x & x \neq 0 \\ 1 & x = 0 \end{cases}$$

For the sake of brevity, we list only a subset of the generated functions, relevant to the solutions presented below: $F_9 = \left\lfloor y + \frac{\sin y}{\mathrm{NZ}(x)} \right\rfloor$, $F_{31} = \lfloor y + y^2 \rfloor$, $F_{42} = \lfloor |x| - x - \cos y \rfloor$, $F_{46} = \lfloor xy - |x| \rfloor$, $F_{55} = \left\lfloor x^3 + \frac{|x|}{\mathrm{NZ}(y)} \right\rfloor$, $F_{58} = \lfloor \cos x \rfloor$, $F_{62} = \left\lfloor \frac{\sin y}{\mathrm{NZ}(y)} + \sin x \right\rfloor$, $F_{66} = \lfloor x \sin x \rfloor$, $F_{67} = \left\lfloor \frac{y}{\sqrt{\mathrm{NZ}(y)}} \right\rfloor$, $F_{88} = \left\lfloor \frac{x^6 + 2y}{\mathrm{NZ}(y^2)} \right\rfloor$, $F_{93} = \left\lfloor \cos x + x - x^3 + \sqrt{\mathrm{NZ}(y)} \right\rfloor$, $F_{97} = \left\lfloor \frac{x^2}{\mathrm{NZ}(\sin x)} \right\rfloor$.

The results from 10 runs of this algorithm are presented in Tab. 5.1. The "Decision sequence" column presents the order in which measurables were being included to the abstraction. Algorithm's decisions regarding including or removing a measurable are denoted with the "+" and "−" signs, resp. The second column, "Result" presents the measurables in the final abstraction with the exact value of its ambiguity functional presented in the third column, "Ambiguity". "Correctness" column indicates whether the result is correct, i.e. is an abstraction with minimal number of measurables, sufficient to describe environment's behaviour. All results were verified numerically: each abstraction yields a set of abstract states that has a $1-1$ relationship with a set of all possible (x,y) pairs. If the index set would be extended with the

Table 5.1: Results for the 100-dimensional Discrete Labyrinth problem with exact calculation of the ambiguity functional.

Run #	Decision sequence	Result	Ambiguity	Correctness
1	$+F_{93}$, $+F_{42}$	(F_{93}, F_{42})	0	✔
2	$+F_{93}$, $+F_9$	(F_9, F_{93})	0	✔
3	$+F_{46}$, $+F_{88}$	(F_{46}, F_{88})	0	✔
4	$+F_{66}$, $+F_{93}$	(F_{66}, F_{93})	0	✔
5	$+F_{66}$, $+F_{62}$	(F_{62}, F_{66})	0	✔
6	$+F_{93}$, $+F_{97}$	(F_{93}, F_{97})	0	✔
7	$+y$, $+F_{93}$	(y, F_{93})	0	✔
8	$+F_{31}$, $+F_{46}$	(F_{31}, F_{46})	0	✔
9	$+F_{46}$, $+F_{58}$	(F_{46}, F_{58})	0	✔
10	$+F_{46}$, $+F_{67}$	(F_{46}, F_{67})	0	✔

following measurable: $o = 10y + x$, which maps every possible plant's underlying state to an

ordinal number, then the algorithm always chooses it in the first step, and the result in every run consists of an abstraction with one measurable: (o), with ambiguity 0, as presented in Tab. 5.2.

Table 5.2: Results for the 100-dimensional Discrete Labyrinth problem with exact calculation of the ambiguity functional, with one universal measurable added.

Run #	Decision sequence	Result	Ambiguity	Correctness
1	$+o$	(o)	0	✔
2	$+o$	(o)	0	✔
3	$+o$	(o)	0	✔
4	$+o$	(o)	0	✔
5	$+o$	(o)	0	✔
6	$+o$	(o)	0	✔
7	$+o$	(o)	0	✔
8	$+o$	(o)	0	✔
9	$+o$	(o)	0	✔
10	$+o$	(o)	0	✔

□

In general, to calculate the exact value of the ambiguity functional for a given abstraction, as in the above experiment, one needs to know $\phi^{-1}(\hat{\mathbf{x}})$, and about the transition and the reinforcement functions. In practice, this is impossible. The following section presents the proposed solution to this problem.

5.4 Estimating the ambiguity functional

To partially overcome the problem of unknown transition and reinforcement functions, and thus contents of the sets $\phi^{-1}(\hat{\mathbf{x}})$ we can somehow estimate the value of the ambiguity functional. In a typical to RL manner we start by gathering samples through an interaction with the plant. To estimate $\phi_{\mathcal{I}}^{-1}(\hat{\mathbf{x}})$, one must group measurements $\mathbf{x}(\mathbf{t})$ by the values of the vectors $\mathbf{x}(\hat{\mathbf{t}})$. Abstract states are being grouped either by their exact values in the discrete case or by the proximity criterion based on the estimated average value of the measurables. Thus, to cover both cases

uniformly, the proximity parameter can be estimated as follows, introducing the resolution parameter ρ:

$$\varepsilon = \mathscr{C}|\mathbf{x}(\mathbf{t}) - \mathbf{x}(\mathbf{t}-\mathbf{1})|/\rho \tag{5.2}$$

where $\rho > 0$. Vector ε defines a maximum possible distances between the measurables to be treated as representing the same measurement. This is a common approach used in Process Control, where such parameters are called *deadbands* [55]. For discrete environments this vector is a zero vector.

In the context of an abstraction function $\phi_{\mathcal{I}}$ and its index set $\mathcal{I}$, a group of measurements belonging to one abstract state is defined as follows:

$$\mathcal{S}^{\hat{\mathbf{x}}}_{\phi_{\mathcal{I}}} = \left\{\mathbf{x} : |\phi_{\mathcal{I}}(\mathbf{x}) - \phi_{\mathcal{I}}(\mathbf{x}^0)| < \phi_{\mathcal{I}}(\varepsilon)\right\} \tag{5.3}$$

where $\mathbf{x^0} \in \mathcal{S}^{\hat{\mathbf{x}}}_{\phi_{\mathcal{I}}}$ is the first measurement added to the group, acting as the arbitrarily distinguished measurement.

Each group of measurements will contain some elements of the inverse image of $\phi_{\mathcal{I}}$, for each abstract state. The terms $\phi^{-1}_{\mathcal{I}}(\hat{\mathbf{x}})$ for all abstract states being evaluated are substituted by their approximations $\mathcal{S}^{\hat{\mathbf{x}}}_{\phi_{\mathcal{I}}}$.

Psychological context The idea that a recurring event (here: values of a subset of measurables) fosters stimulus discrimination ability is also investigated in psychology [50]. If an event does not appear enough times many measurables (discriminators) appear to be directly related with the event's consequences [102]. □

With the estimated contents of $\phi^{-1}_{\mathcal{I}}(\hat{\mathbf{x}})$ and the use of the simulator (the *simulator assumption* described in Chapt. 3), the values of $\bar{\bar{\phi}}_{\mathcal{I}}(f_x(\mathbf{x},u), f_r(\mathbf{x}))$ for all $u \in \mathcal{U}$ can be easily determined, and used to estimate the ambiguity functional.

Computer Experiment 5.4.1. Let us now evaluate the idea presented above with an experiment. We will run the same process as we did in Example 5.1.1 above, but this time the ambiguity functional will be estimated from a random sample of states. Consider again the Discrete Labyrinth task with only seven measurables: $x, y, s = x+y, d = |x-y|, m = xy, q_x = \lfloor\sqrt{x}\rfloor, q_y = \lfloor\sqrt{y}\rfloor$. Because the values of the ambiguity functional are now being estimated, we reiterate the algorithm if for some estimation it does not insert any measurables. We assume that the process is finished if it fails to insert any measurables for MaxFailureCount number of times in a row. The results from 10 runs of this algorithm are presented in Tab. 5.3. The ambiguity functional

for each abstraction was estimated with 25 random states. Meaning of the columns is the same as described in Experiment 5.3.1. All results were verified numerically: each abstraction yields a set of abstract states that has a $1-1$ relationship with a set of all possible (x,y) pairs. This demonstrates the problem described in Example 4.2.1 in practice: there are possible scenarios that can lead to a non-optimal solution in terms of number of measurables, as can be seen in the 8th run of this experiment. The proposed solution to this problem is presented in the next section.

Table 5.3: Results for the 7-dimensional Discrete Labyrinth problem with estimation of the ambiguity functional. The MaxFailureCount parameter was set to 5.

Run #	Decision sequence	Result	Ambiguity	Correctness
1	+x, +y	(x,y)	0	✔
2	+y, +m, +x	(x,y,m)	0	✖
3	+s, +y	(y,s)	0	✔
4	+s, +y	(y,s)	0	✔
5	+s, +x	(x,s)	0	✔
6	+s, +y	(y,s)	0	✔
7	+s, +x	(x,s)	0	✔
8	+m, +d, +y	(y,d,m)	0	✖
9	+x, +y	(x,y)	0	✔
10	+s, +y	(y,s)	0	✔

□

5.5 Inserting and removing measurables

The incremental process presented in Sec. 4.2 includes the measurables one by one, thus lowering the estimated value of the ambiguity functional. When the value of the ambiguity functional is estimated from a finite set of samples, the algorithm is prone to choosing an abstraction that looks better than the others for some set of samples, but in fact its ambiguity functional is larger for another set of samples. Also, the process of constraining the abstraction presented in Sec. 4.2 can lead to suboptimal solution, depending on the order of inserting measurables. Because of this it is necessary to constantly verify that previously included measurables are still needed.

This is done by removing them, if this would not increase the value of the ambiguity functional. This approach allows the extension algorithm to make mistakes including some incorrect measurables that only look promising. After the correct measurables were found (and the value of the ambiguity functional is low), it is be possible to determine if the previously included measurables are redundant and can be removed. This is directly related to the problem in the psychological experiment described in [102]. Let us recall again the incremental process of forming subsequent abstractions: $\phi^1, \phi^2, \ldots$. In each step of the proposed approach first we try to *extend* the abstraction function by inserting a measurable do the *index set*, and thus creating a new abstraction function. If none of the possible extensions lowers the ambiguity functional, we try to *reduce* the abstraction by removing one of measurables from its *index set* if it will not make the values of the ambiguity functional larger. We call the sequence of subsequent abstractions resulting from these operations an *abstraction path*.

Thought Experiment 5.5.1. As an example, Figure 5.1 presents two possible abstraction paths for a space with four measurables. The abstraction path marked with the red line at first performs three consecutive extensions inserting measurables $\mathbf{x}_3$, $\mathbf{x}_4$ and $\mathbf{x}_2$ and then a reduction removing the measurable $\mathbf{x}_4$. This corresponds to the following sequence of abstraction functions along with their index sets:

$$\phi_{\emptyset} \xrightarrow{+3} \phi_{\{3\}} \xrightarrow{+4} \phi_{\{3,4\}} \xrightarrow{+2} \phi_{\{2,3,4\}} \xrightarrow{-4} \phi_{\{2,3\}} \tag{5.4}$$

Similarly, the abstraction path denoted with the blue line presents the following sequence of extensions and reductions:

$$\phi_{\emptyset} \xrightarrow{+2} \phi_{\{2\}} \xrightarrow{+1} \phi_{\{1,2\}} \xrightarrow{+3} \phi_{\{1,2,3\}} \xrightarrow{+4} \phi_{\{1,2,3,4\}} \xrightarrow{-2} \phi_{\{1,3,4\}} \xrightarrow{-3} \phi_{\{1,4\}} \xrightarrow{-3} \phi_{\{1,4\}} \tag{5.5}$$

□

A question that naturally comes to mind is: what if an abstraction path will intersect itself? Fortunately, theoretically, this is impossible. Consider the following reasoning.

Theorem 5.5.1. The proposed incremental process of constructing an abstraction function will not evaluate the same abstraction more than once.

Proof. Assume an arbitrary point on an abstraction path, corresponding to an index set $\mathcal{I}$. We say that a measurable (or an index) disambiguates an abstract state $\hat{\mathbf{x}}$ if inserting it to some abstraction ϕ will lower the ambiguity functional calculated for that state. Looking again at

Fig. 5.1, it is convenient to think about extensions as *going up* the abstraction path diagram, and reductions as *going down*. Abstractions on the same level of the diagram, are the ones with equal sizes of their index sets, and we call them *siblings*. The following proof shows that it it impossible to reach an already visited point on an abstraction path via extensions and reductions with the approach described above.

Each element i in the index set $\mathcal{I}$ (the starting point) was added because it disambiguates some abstract state $\hat{\mathbf{x}}^i$. The next operation, according to the approach described above, can be either an extension or a reduction (if none of them is possible, the process ends and the statement under the question holds). To reach the starting point again the subsequent extension/reduction operations must create a cycle. The point before the last point in the cycle (which would be the starting point again) must be either on the higher level of the abstraction path diagram (from which it will reach the starting point via a reduction) or on the lower level (from which it will reach the starting point via an extension). Reaching the starting point from a sibling point is impossible, because that would involve substitution of measurables, which is not among possible operations in our approach.

1. Consider the first possibility: is it possible to reduce from a point $\mathcal{K}$, being one level above the starting point, to $\mathcal{I}$, if the process has already visited $\mathcal{I}$? Let j be the one measurable present in $\mathcal{K}$ and not present in $\mathcal{I}$, i.e. $j = \mathcal{K} \setminus \mathcal{I}$. If at some point of the supposed cycle j was added to the set, it must have been because there exists an abstract state $\hat{\mathbf{x}}^j$ disambiguated by j with measurables from $\mathcal{I}$ already present in the abstraction. This means that it is impossible to remove j from the index set, because that would make the ambiguity functional larger.

2. Consider the second possibility: is it possible to extend from a point $\mathcal{H}$, being one level below the starting point, to $\mathcal{I}$, if the process has already visited $\mathcal{I}$? Let h be the one measurable present in $\mathcal{I}$ and not present in $\mathcal{H}$, i.e. $h = \mathcal{I} \setminus \mathcal{H}$. If at some point of the supposed cycle h got removed, it means there is no abstract state $\hat{\mathbf{x}}^h$ that determined the value of the ambiguity functional (because of the maximization, one abstract state can do that) and was disambiguated by the measurable h. This means that it is impossible to insert h to the index set, because that would not make the ambiguity functional lower.

Thus, if values of the ambiguity functional can be calculated accurately, cycles in abstraction paths are impossible. □

However, in practice (as it is presented in the following sections), the ambiguity functional is estimated from a finite sample of states. Because of that, when performing an extension with measurable i the abstract state disambiguated by i, $\hat{\mathbf{x}}^i$ is stored in an entity called *extension memory*. *Extension memory* holds the states that justified inclusion of a particular measurable. Each reduction operation is additionally verified with the states from the *extension memory*, so it is also impossible to form a cycle in a practical setting.

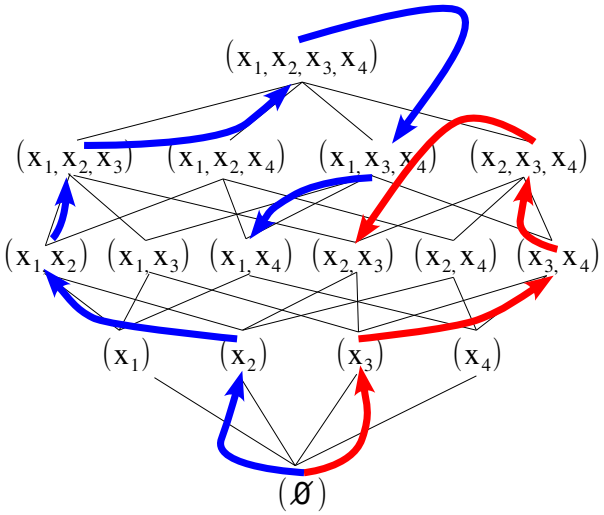

Figure 5.1: All possible subsets of measurables for a four-dimensional space with measurables: $\{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4\}$. Black lines indicate a possible transition between abstract state spaces by inserting (when going up the diagram) or removing (when going down the diagram) measurables. Red and blue lines present two exemplary abstraction paths.

Computer Experiment 5.5.1. To evaluate the method proposed above, we will run the same process as we did in Example 5.4.1 above, but this time the algorithm is modified: it attempts to remove redundant measurables. Again, the test problem is the Discrete Labyrinth environment with seven measurables: $x, y, s = x + y, d = |x - y|, m = xy, q_x = \lfloor\sqrt{x}\rfloor, q_y = \lfloor\sqrt{y}\rfloor$. To speed up the process, if some extensions improve the ambiguity functional equally, they are all considered as possible solutions. Thus, the algorithm evaluates many possible abstraction paths simultaneously. The procedure for one abstraction path is presented in Listing 2. To hold the property of not self-intersecting abstraction paths described in Sec. 5.5 a collection called *extension memory* is introduced. It holds all samples used for estimating the ambiguity functional, if as a result of this estimation the current abstraction was extended. The results from 10 runs of this algorithm are presented in Tab. 5.4. Columns have similar meaning to the ones described in Experiment 5.3.1. The "Example decision sequence" column presents the order, in which

measurables were added to the abstraction, to reach one of possible solutions. This sequence is exemplary, because the algorithm usually reaches the same abstraction through different paths (although it never falls back to an already evaluated abstraction, to avoid cycles). For example, all possible steps in the first run of the algorithm that led to reaching the (x, s) abstraction are presented in Fig. 5.2. For readability of the results, we always present only one possibility as an example. The second column, "Best results" presents all solutions reached by the algorithm. Solutions are abstractions with the least number of measurables among the abstractions with the lowest value of the ambiguity functional. This value is presented in the third column, "Ambiguity". The fifth column, "Correctness" indicates whether the solution is correct. The last column, "Coverage" presents the coverage of the solution space - i.e. how many of possible abstractions were evaluated by the algorithm in relation to the all possible number of abstractions.

These results show that the measurable removal mechanism seemingly solves the problem of redundant measurables, present in the previous Experiment 5.4.1. The most complex example of the behaviour this mechanism exhibits, can be seen in the last run, when two incorrect measurables, namely q_y and d, were added to the abstraction. First, a random sample of states was generated that proved these two measurables (together with x) to be redundant, and in turn q_y was removed. After the correct measurables, s was added to the abstraction, d was also no longer needed and the result again was rectified to contain only non-redundant measurables. Similar, but simpler scenario occurred in the 5th run. The main problem with this approach is large solution space coverage, which renders this solution to be still impractical. We deal with this problem in the following section.

□

5.6 Estimating the order of the system

A natural solution to the problem presented in the previous section, in Experiment 5.5.1 is to inhibit evaluation of abstractions of higher order, if there are still abstractions with lower order that look promising. Specifically, if at least one abstraction with k measurables, was not extended then, there is no point in evaluating abstractions with more than k measurables. The algorithm employing this idea is presented in Listing 3.

Psychological context The psychological intuition behind this idea is that people prefer simple explanations over more complicated ones (e.g. [95]). Similarly to this principle we assume

Table 5.4: Results for the 7-dimensional Discrete Labyrinth problem with estimation of the ambiguity functional.

Run #	Example decision sequence	Best results	Ambiguity	Correctness	Coverage
1	$+x,+s$	$(x,s),(x,y),(y,s)$	0	✔	49%
2	$+x,+y$	$(x,s),(x,y),(y,s)$	0	✔	50%
3	$+s,+x$	$(x,s),(x,y),(y,s)$	0	✔	50%
4	$+x,+s$	$(x,s),(x,y),(y,s)$	0	✔	50%
5	$+y,+m,+s,-m$	$(x,s),(x,y),(y,s)$	0	✔	50%
6	$+x,+y$	$(x,s),(x,y),(y,s)$	0	✔	50%
7	$+x,+s$	$(x,s),(x,y),(y,s)$	0	✔	50%
8	$+x,+s$	$(x,s),(x,y),(y,s)$	0	✔	49%
9	$+y,+s$	$(x,s),(x,y),(y,s)$	0	✔	49%
10	$+q_y,+x,+d,-q_y,+s,-d$	$(x,s),(x,y),(y,s)$	0	✔	50%

that until all abstractions of lower order (i.e. that preserve less measurables) are not proven to be incorrect, we do not want to evaluate ones of higher order (i.e. that preserve more measurables).
□

Computer Experiment 5.6.1. To verify correctness of the order estimation idea, we run this algorithm on the Discrete Labyrinth environment. The results are presented in Tab. 5.5. Meaning of the columns is the same as described in Experiment 5.5.1. It can be seen that the idea proposed in this section works as expected, and the coverage of the solution space has been significantly reduced.

□

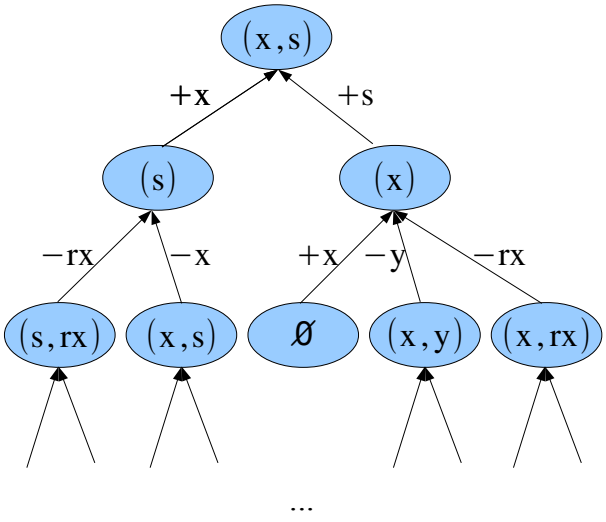

Figure 5.2: All possible steps leading to the (x,s) abstraction in the first run of the algorithm, presented in Tab. 5.4.

Table 5.5: Results for the 7-dimensional Discrete Labyrinth problem with estimation of the ambiguity functional and with order estimation.

Run #	Example decision sequence	Best results	Ambiguity	Correctness	Coverage
1	$+q_x,+x,-q_x,+s$	$(x,s),(x,y),(y,s)$	0	✔	22%
2	$+y,+s$	$(x,s),(x,y),(y,s)$	0	✔	23%
3	$+x,+y$	$(x,s),(x,y),(y,s)$	0	✔	22%
4	$+q_x,+x,-q_x,+y,-x,+s$	$(x,s),(x,y),(y,s)$	0	✔	22%
5	$+s,+x$	$(x,s),(x,y),(y,s)$	0	✔	20%
6	$+s,+x,-s,+y$	$(x,s),(x,y),(y,s)$	0	✔	23%
7	$+q_x,+x,-q_x,+y,-x,+s$	$(x,s),(x,y),(y,s)$	0	✔	21%
8	$+q_y,+y,-q_y,+x$	$(x,s),(x,y),(y,s)$	0	✔	20%
9	$+m,+x,-m,+y$	$(x,s),(x,y),(y,s)$	0	✔	20%
10	$+q_y,+y,-q_y,-y,+x,+s$	$(x,s),(x,y),(y,s)$	0	✔	22%

Algorithm 2 Algorithm with measurable removal and ambiguity estimation

1: Start with the null abstraction $\phi \leftarrow \phi_\emptyset$
2: Estimate the value of the ambiguity functional for ϕ and for all possible abstractions with one extra measurable
3: **if** An abstraction with extended set of measurables ϕ^{ext}, for which $A(\phi^{\text{ext}}) < A(\phi)$ exists **then**
4: Remember the current sample in *extension memory*
5: Reset the *failure counter*
6: $\phi \leftarrow \phi^{\text{ext}}$ and start over from step 2
7: **else**
8: Estimate the value of the ambiguity functional for ϕ and for all possible abstraction with one measurable removed
9: Choose randomly an abstraction ϕ^{red} for which $A(\phi^{\text{red}}) \leq A(\phi)$, with the value of the ambiguity functional estimated with random samples and with samples from *extension memory*
10: **if** For any ϕ^{red} $A(\phi^{\text{red}}) \leq A(\phi)$ **then**
11: $\phi \leftarrow \phi^{\text{red}}$
12: Reset the *failure counter*
13: Start over from step 2
14: **else**
15: Increment the *failure counter*
16: **end if**
17: **end if**
18: **if** The value of the *failure counter* $\geq$ MaxFailureCount **then**
19: STOP
20: **else**
21: Start over from step 2
22: **end if**

Algorithm 3 Algorithm with measurable removal and ambiguity estimation

1: Start with the null abstraction $\phi \leftarrow \phi_\emptyset$
2: Set $order \leftarrow 0$
3: Estimate the value of the ambiguity functional for ϕ and for all possible abstractions with one extra measurable
4: **if** An abstraction with extended set of measurables ϕ^{ext}, for which $A(\phi^{\text{ext}}) < A(\phi)$ exists **then**
5: Remember the current sample in *extension memory*
6: Reset the *failure counter*
7: **if** Any abstraction with k measurables was evaluated and not extended **then**
8: add ϕ^{ext} to the queue, but do not evaluate it
9: **else**
10: $order \leftarrow order + 1$
11: $\phi \leftarrow \phi^{\text{ext}}$ and start over from step 2
12: **end if**
13: **else**
14: Estimate the value of the ambiguity functional for ϕ and for all possible abstraction with one measurable removed
15: Choose randomly an abstraction ϕ^{red} for which $A(\phi^{\text{red}}) \leq A(\phi)$, with the value of the ambiguity functional estimated with random samples and with samples from *extension memory*
16: **if** For any ϕ^{red} $A(\phi^{\text{red}}) \leq A(\phi)$ **then**
17: $\phi \leftarrow \phi^{\text{red}}$
18: Reset the *failure counter*
19: Start over from step 2
20: **else**
21: Increment the *failure counter*
22: **end if**
23: **end if**
24: **if** The value of the *failure counter* ≥ 10 **then**
25: STOP
26: **else**
27: Start over from step 2
28: **end if**

Chapter 6

Ambiguity resolving approach for continuous domains

This chapter develops the ideas presented in the previous chapter to be applicable to the continuous state context. We provide means of estimating the ambiguity functional for continuous transition and reinforcement functions and propose a continuous version of state abstraction algorithm.

In this chapter we present the first, known to the author, numerical simulation of a state abstraction algorithm in a continuous state domain (starting from Experiment 6.5.1). In the first series of experiments, we use generated samples (*simulator assumption*, described in Sec. 3.2). This allows us to generate samples close to the centers of sample groups (see Sec. 5.4), needed to estimate the value of the ambiguity functional. Usually, in practice, this is not possible, because given a measurement the knowledge about the plant's underlying state is essential in order to generate another valid measurement in a desired proximity. However, this allows us to analyze all possible abstraction paths and verify that the algorithm behaves correctly regardless of the abstraction path determined by a particular order of samples. Assuming that all measurements have non-zero probability of occurring, we would observe them *eventually* - it just may take a very long time. In Chapt. 7 we show how the algorithm can be used in a more practical scenario that does not require a model of the plant.

6.1 Ambiguity functional for continuous transition and reinforcement functions

For continuous transition functions, it is not possible to count the number of different abstract states in the given set, as suggested by (5.1), because it is hard to define what is the meaning of "different" in the continuous case. However, if we introduce a distance threshold parameter, we can count the number of abstract state pairs with a distance above that threshold. This step is similar to the way the discrete problem statement in (4.2) was extended to the continuous form in (4.1). To decide whether a particular distance between the measurements makes them equivalent, we use the threshold parameters introduced in Sec. 4.2, namely a vector $\theta^{(x)}$ and a scalar $\theta^{(r)}$. These parameters contain values of distance thresholds for each measurable, and reinforcement values separately. Conditions defined in (4.1) verify whether the values are contained inside a hyper-sphere, because of the threshold parameter being a scalar and the Euclidean distance between the measurements is being calculated. Here we propose to verify whether they are contained within a box with its sides length defined by the *threshold vector*. This is different than the assumption in (4.1), but it is consistent with the *deadband* approach mentioned in Sec. 5.4. Formally, this condition is defined as the following predicate, which verifies whether two abstract state-reinforcement pairs are different, in the context of a particular abstraction $\phi_{\mathcal{F}}$:

$$\text{Different}((\mathbf{x}^1, r^1), (\mathbf{x}^2, r^2)) = |\phi_{\mathcal{F}}(\mathbf{x}^1) - \phi_{\mathcal{F}}(\mathbf{x}^2)| > \phi_{\mathcal{F}}(\theta^{(x)}) \vee |r^1 - r^2| > \theta^{(r)} \tag{6.1}$$

Using this definition of equivalence the number of different abstract state-reinforcement pairs in the Λ function's argument, set $\hat{\mathfrak{X}}_{t+1}$, can be counted. To account for the worst case, each of the abstract states is chosen as a *central abstract state* and the abstract states outside of the box centered at the *central abstract state* are counted. The result indicates the overall ambiguity:

$$\Lambda(\hat{\mathfrak{X}}_{t+1}) = \max_{(\mathbf{x}^0, r^0) \in \hat{\mathfrak{X}}_{t+1}} \| \{ (\mathbf{x}, r) \in \hat{\mathfrak{X}}_{t+1} : \text{Different}((\mathbf{x}^0, r^0), (\mathbf{x}, r)) \} \| \tag{6.2}$$

Similarly to the discrete version (5.1), the continuous version satisfies the properties (2). Abstract states that are more ambiguous will yield more different, in the sense defined in 6.1, elements in Λ's argument. For unambiguous abstract states, either different outcomes of the transition and reinforcement functions will be indistinguishable in the context of the difference predicate (6.1) or there will be only one element in the set, which will yield 0. Thus, analogously to the discrete version presented in the previous section, minimizing this functional is

equivalent to transforming the index set by the H function as described in Sec. 4.2 and leads to a suboptimal solution of the non-linear problem presented in (4.1). In further sections we present some heuristics that can alleviate this suboptimality. The following section presents the algorithm used for gathering sample groups, described in Sec. 5.4 for the continuous case.

6.2 The proposed algorithm for continuous state domains

With the definition of the ambiguity functional presented in Eq. 4.10 in Sec. 4.4 and Eq. 6.2 in Sec. 6.1 above, we can apply the state abstraction algorithm from the previous chapter 5 to the continuous state domains. However, we need a method of creating sample groups, described in Sec. 5.4, for measurements from $\mathcal{R}^n$- the following section 6.3 examines possible algorithm that deal with grouping such vectors, and the next section 6.4 presents a modification, in relation to the method from Sec. 5.4, of creating samples groups for the continuous state case.

Recall from Sec. 4.1 that in the continuous state case two additional threshold parameters need to be determined. Section 6.5 presents a solution to this issue. Experiments show that in the continuous case it is important to estimate the values of the ambiguity functional for different abstractions to compare them, to have equal number of samples for each estimation. Section 6.6 presents a simple modification of the sample group creation mechanism to deal with this issue.

Finally, in Sec. 6.7 we introduce a modification to the method of calculating the threshold parameters to compensate for transition functions that are expansive mappings. This leads us to the final form of the algorithm, presented in Listing 9 in Sec. 6.8.

All modifications introduced in this section do not invalidate the algorithm in the discrete context, as we show by evaluating the final form of the algorithm, without the simulator assumption, on the Discrete Labyrinth in the following Chapter 7.

6.3 Estimating the ambiguity functional in the continuous case

Grouping the abstract states by the values of measurables is in fact an instance of the *fixed-radius nearest neighbour problem* [12]. Additionally, the queries about neighbouring abstract states are issued in the context of different abstractions, and thus - different projections of the

space of measurables. This problem is addressed by Papis and Pacut in [91]. In this section we reiterate the most relevant, in the context of this work, parts of this paper.

6.3.1 Exact solution

The main purpose of our data structure is to answer the nearest neighbour queries in a context of different projections. We adapt the solution presented in [37]. The presented idea is also conceptually close to Random Projection techniques [19]. The algorithm is presented in Listings 4 and 5. We maintain all of the points sorted according to values of all measurables (line 10 of Algorithm 4). For each abstract state we also store their one-dimensional neighbourhoods, which are updated during each insert (line 8 of Algorithm 4). Update of one axis has the complexity of $O(\log k)$ for performing a binary-search with k abstract states, and $O(m)$ for updating neighbourhoods, with m neighbours. So, overall the update operation is $O(n(\log k + m))$. Space complexity is $O(nkm)$ - for n axes we store m identifiers for each of k input points. Then, a fixed-range query for one axis can be solved in $O(\log k + m)$ time (line 3 of Algorithm 5). The results need to be aggregated for all axes, so that only the input points within the given radius on all axes remain. Let us assume $O(m)$ complexity for calculating the intersection of two hash tables containing only point identifiers, i.e. integer numbers (line 4 of Algorithm 5). Then for the Chebyshev metric for a particular projection related to a given abstraction, the fixed-radius query has the complexity $O(nm)$. The main drawback of this version is a constant radius query parameter, however, for the purpose of abstract state aggregation this is not an important problem, as we calculate the ε parameter beforehand (see (5.2)). We call this version of the solution the Log version.

6.3.2 Storage-less solution

Space complexity can be traded for query time complexity. If we will not store the neighbourhoods and will not update them on each insert then: the update time complexity becomes $O(D \log n)$ (D runs of Algorithm 6), space complexity becomes $O(Dn)$, but the query time complexity (Algorithm 7) becomes $O(D(\log n + m))$ (or $O(D^2(\log n + m))$ for the Euclidean distance). Additional important merit, relating the the memory expensive version presented in previous Section, is that the radius query parameter is no long constant. This version will be further referred to as the StoragelessLog version.

Algorithm 4 Update operation for one axis. Log version of the proposed nearest neighbour algorithm.

Require: *point* - new point to add to the structure

Require: *pointId* - identifier of the new point

Require: *di* - index of the dimension for this axis

1: $lowerBound \leftarrow point_{di} - \varepsilon$

2: $upperBound \leftarrow point_{di} + \varepsilon$

3: $lowerBoundIndex \leftarrow BinarySearch(sortedPoints, lowerBound)$

4: $upperBoundIndex \leftarrow BinarySearch(sortedPoints, upperBound)$

5: $neighboursForNewPoint \leftarrow EmptyList$

6: **for** $i \leftarrow lowerBoundIndex$ to $upperBoundIndex$ **do**

7: $\quad neighboursForNewPoint \leftarrow neighboursForNewPoint \cup sortedPoints_i$

8: $\quad storedNeighbourhoods_i \leftarrow storedNeighbourhoods_i \cup pointId$

9: **end for**

10: add *point* to *sortedPoints* at appropriate index

11: add a new neighbourhood for *pointId* with neighbours *neighboursForNewPoint*

Ensure: *point* with *pointId* identifier is added to the structure with calculated neighbourhood and neighbourhoods of relevant points are updated

6.3.3 Approximate solution

For the purpose of the discussed bisimulation-based variable selection it is important to aggregate similar abstract states, i.e. at most n-dimensional vectors that are close enough to each other. From this point of view, if we know that a vector **y** is a neighbour of **x**, the information that **x** is also a neighbour of **y** is obviously redundant. What is more, for a small radius the neighbourhood of vector **y** will be very similar to the neighbourhood of vector **x**. Thus, we do not loose much information by ignoring the neighbourhood of vector **y**. This situation is presented in Fig. 6.1. The approximate version of the algorithm creates neighbourhood groups only for input points that are not within range of some other input point. We call such input points *core points* (analogy to the notion of *core* in the DBSCAN algorithm [27]). Then, the space complexity reduces to $O(k)$ for one axis, because each abstract state can be at most in two overlapping neighbourhoods. However, to correctly return neighbourhoods for more than one measurable, if a given point becomes a *core point* on one of the axes, it must be also added

Algorithm 5 Query operation. Log version of the proposed nearest neighbour algorithm.

Require: *pointId* - identifier of a query point

Require: $\mathcal{J}$ - index set of a given abstraction

1: $result \leftarrow$ neighbourhood for *pointId* from the first axis from $\mathcal{J}$

2: **for all** $i \in \mathcal{J}$ except the first element **do**

3: $partialResult \leftarrow$ neighbourhood for *pointId* from the ith axis

4: $result \leftarrow$ hashtable intersection of *result* and $partialResult$

5: **end for**

Ensure: *result* contains the result of the fixed-radius query for a distance metric determined by $\mathcal{J}$

Algorithm 6 Update operation for one axis. StoragelessLog version of the proposed nearest neighbour algorithm.

Require: *point* - new point to add to the structure

Require: *pointId* - identifier of the new point

Require: *di* - index of the dimension for this axis

1: $insertionIndex \leftarrow BinarySearch(sortedPoints, point_{di})$

2: add *point* to $sortedPoints$ at index *insertionIndex*

Ensure: *point* with *pointId* identifier is added to the structure

as a *core point* on other axes. Because of this the overall space complexity for the whole algorithm is slightly worse than $O(nk)$. The approximate version is presented in Listing 8. It is very similar to the exact version (algorithm 4) with the difference in line 6 - if the condition is true, then the added point falls within a range of some existing *core point* and will not be added to $sortedPoints$ collections. Otherwise, the added point forms its own, new neighbourhood (line 10). We call this version of the proposed solution the ApproximateLog version.

6.3.4 Experimental comparison of nearest neighbour algorithms

The proposed method is compared with state-of-the-art nearest neighbour algorithms. Experiments were carried out for the Chebyshev distance, using random samples from the Cart-Pole Swing-Up environment. In the presented graphs, the algorithms are called as follows:

- the *M-Tree* algorithm: MTree [14]
- the *k-d tree* algorithm: KdTree [36]

Algorithm 7 Query operation for one axis. To aggregate the result for a particular index set $\mathcal{I}$ algorithm 5 is used. StoragelessLog version of the proposed nearest neighbour algorithm.

Require: *pointId* - identifier of a query point

1: $lowerBound \leftarrow point_{di} - \varepsilon$

2: $upperBound \leftarrow point_{di} + \varepsilon$

3: $lowerBoundIndex \leftarrow BinarySearch(sortedPoints, lowerBound)$

4: $upperBoundIndex \leftarrow BinarySearch(sortedPoints, upperBound)$

5: *result* $\leftarrow$ all points with indices in range $[lowerBoundIndex, upperBoundIndex)$

Ensure: *result* contains the result of the fixed-radius query for this axis

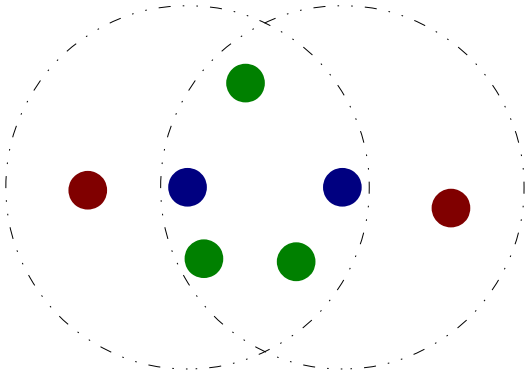

Figure 6.1: Two neighbourhoods of two points (in blue) that are neighbours to each other. One of them can be omitted at the expense of loosing information about one of non-common neighbours (in red).

- a basic version of the proposed solution: Log
- a version with lowered memory usage: StoragelessLog
- an approximate version: ApproximateLog

The first four algorithms yield exact results, whereas the last one, ApproximateLog, can sometimes return incomplete neighbourhoods, as described in Sec. 6.3.3.

Figure 6.2 presents measurements of the query operation processing time, depending on the number of abstract states processed so far. For 12 dimensions only the approximate solution can compete with *M-Tree* and *k-d tree*. Figure 6.3 presents processing time of query operations for 10000 samples, depending on the number of queried dimensions. As in the previous scenario, only the approximate solution yields similar results to *M-Tree* and *k-d tree* for 12 dimensions.

Algorithm 8 Update operation for one axis. ApproximateLog version of the proposed nearest neighbour algorithm.

Require: *point* - new point to add to the structure

Require: *pointId* - identifier of the new point

Require: *di* - index of the dimension for this axis

1: $lowerBound \leftarrow point_{di} - \varepsilon$

2: $upperBound \leftarrow point_{di} + \varepsilon$

3: $lowerBoundIndex \leftarrow BinarySearch(sortedPoints, lowerBound)$

4: $upperBoundIndex \leftarrow BinarySearch(sortedPoints, upperBound)$

5: $neighboursForNewPoint \leftarrow EmptyList$

6: **if** $lowerBoundIndex < upperBoundIndex$ **then**

7: **for** $i \leftarrow lowerBoundIndex$ to $upperBoundIndex$ **do**

8: $storedNeighbourhoods_i \leftarrow storedNeighbourhoods_i \cup pointId$

9: **end for**

10: **else**

11: add *point* to $sortedPoints$

12: add a new neighbourhood for *pointId* with neighbours *neighboursForNewPoint*

13: **end if**

Ensure: *point* with *pointId* identifier is added to the structure with calculated neighbourhood or neighbourhoods of relevant points are updated

When considering queries for different combinations of input dimensions, the ordinary methods need to maintain one instance of their whole structure per one subset of input dimensions, which in turn results in high update query costs. This drawback is not present in StoragelessLog and ApproximateLog versions of the proposed solution, and thus their average update times are the smallest. This is presented in Fig. 6.4. Note that in these experiments not all of possible subsets of dimensions were considered - it would be impractical for the MTree and KdTree methods, as the number of possible subsets of input dimensions is 2^n. On the other hand, the methods proposed in this work need only n simple structures to maintain and can handle any of 2^n possible dimension subsets. This figure shows also a problem with Log version - update processing time grows substantially when there is too much neighbourhoods to handle. ApproximateLog version is not affected by this issue.

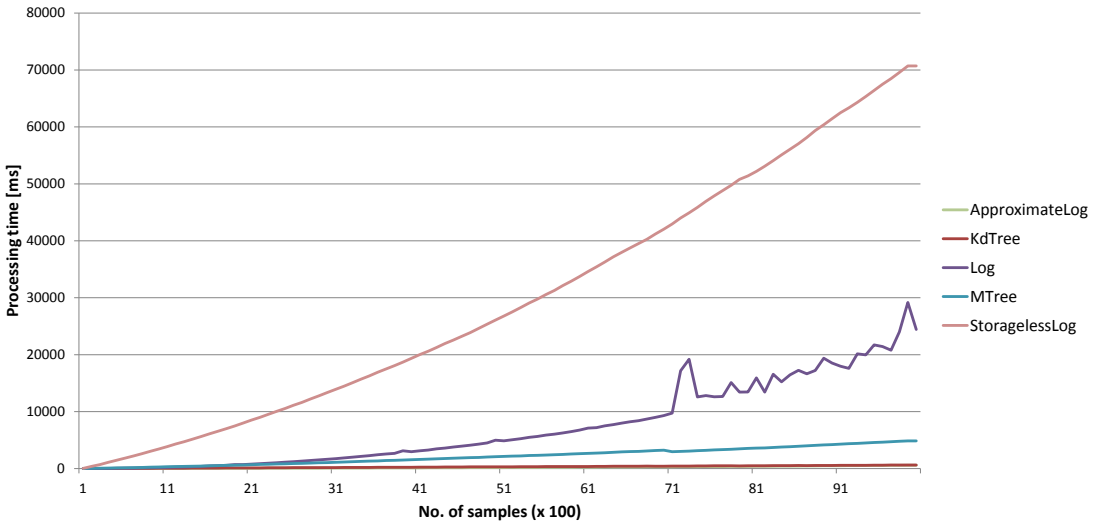

Figure 6.2: Processing time of fixed-radius query for 12 dimensions for number of input points processed so far ($\times 100$). In this case only ApproximateLog version of the proposed method is comparable with MTree and KdTree solutions.

6.3.5 Preliminary evaluation of the abstraction algorithm in the continuous case

In this section we present a simple analysis of the abstraction algorithm's behaviour when using the continuous definition (6.2) of the ambiguity functional.

Thought Experiment 6.3.1. Let us examine the consequences of using the continuous definition of the ambiguity functional. We will show that a simple, straightforward application of this definition in the incremental process presented in Sec. 4.2 is not enough for the algorithm to work. We will use a small set of measurements for the Continuous Labyrinth environment, so it will be possible to calculate the values of the ambiguity functional by hand. Consider the following measurables: x, y, v_x, v_y and the following abstractions: $\phi_{\{x\}}$ and $\phi_{\{x,y\}}$. Assume the following epsilon vector values: $\varepsilon = (0.001, 0.001, 0.001, 0.001)$ and the following threshold vector values: $\theta^{(x)} = (0.001, 0.001)$ and $\theta^{(r)} = 1$. Assume the following measurements would

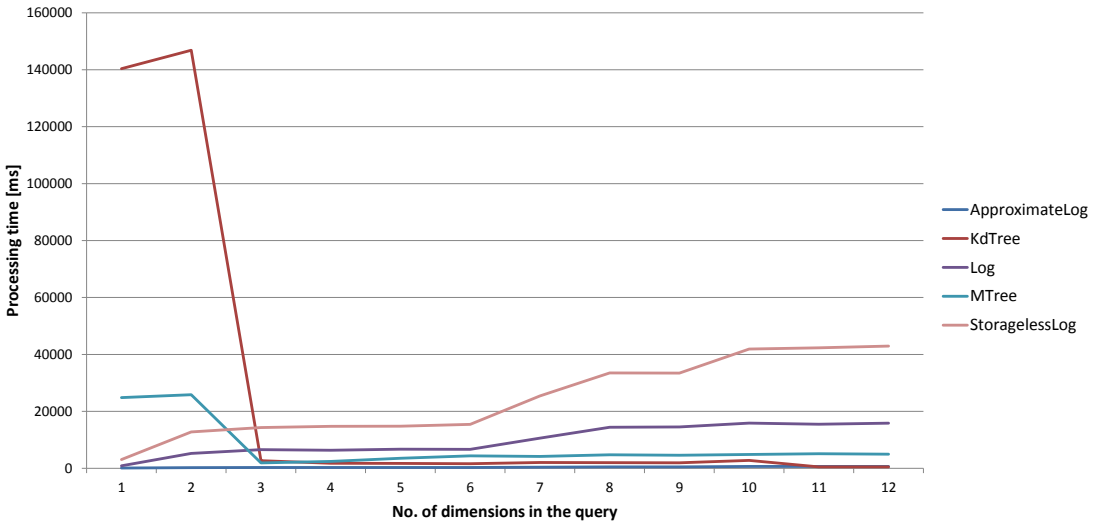

Figure 6.3: Processing time of fixed-radius query after 10000 input points for the given number of dimensions, for the *Cart-Pole* generator. Similarly to the case with fixed number of dimensions, presented in Fig. 6.2, MTree and KdTree and ApproximateLog version of the proposed method seem to be the best in terms of processing time.

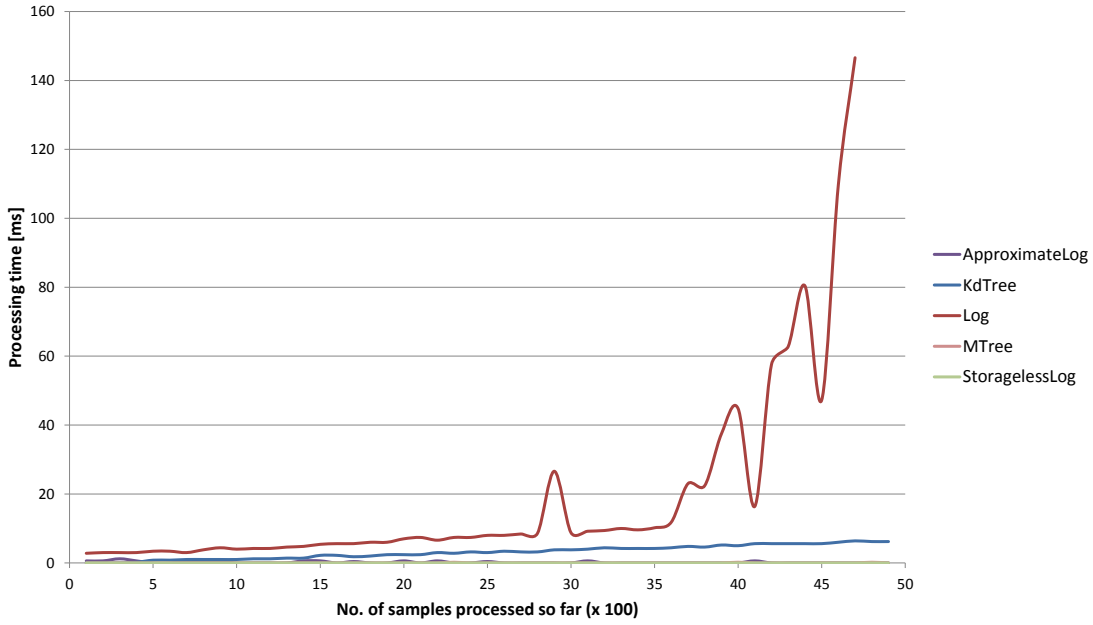

Figure 6.4: Average processing time of 1000 update operations against number of input points so far, up to 5000 samples. Two of the proposed methods, namely StoragelessLog and ApproximateLog versions, along with the MTree structure, outperform the KdTree solution. Processing time of Log version becomes unacceptable for large number of samples.

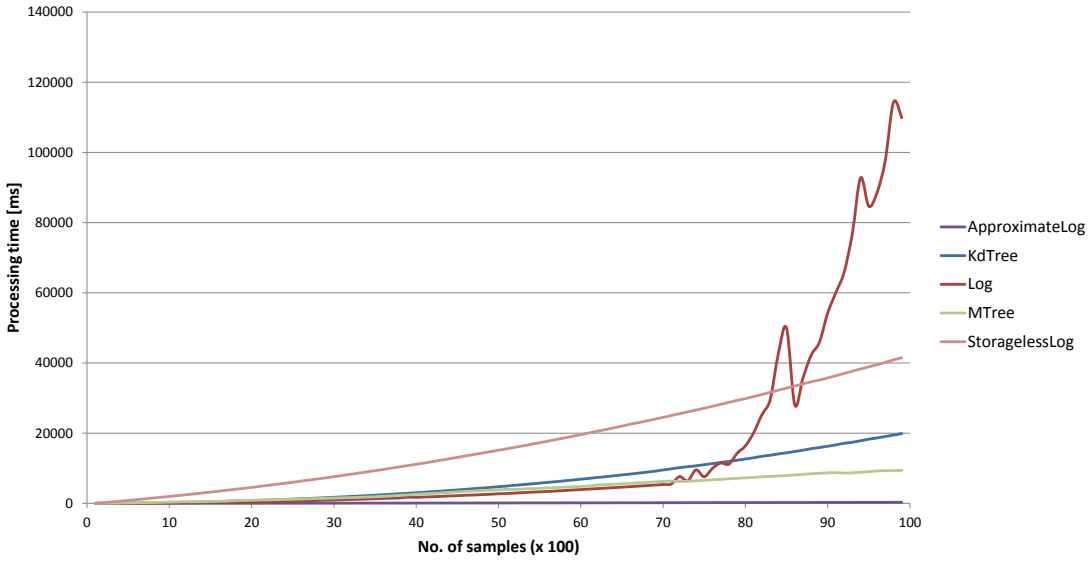

Figure 6.5: Average processing time of 1000 subsequent queries, for random number of dimensions, against number of samples processed so far. ApproximateLog solution yields the lowest processing time.

have been generated for the purpose of estimating the ambiguity functional:

$$\mathbf{x}^1 = (x = 2.9, y = 9, v_x = 0, v_y = 0)$$
$$\mathbf{x}^2 = (x = 2.9, y = 7, v_x = 0, v_y = 0)$$
$$\mathbf{x}^3 = (x = 2.9, y = 0.5, v_x = 0, v_y = 0)$$
$$\mathbf{x}^4 = (x = 2.9, y = 0.5, v_x = 0, v_y = 1)$$

(refer to Fig. 2.1). The discussed scenario is illustrated in Fig. 6.6. Consider the action $u =$ RIGHT, and calculate $A(\phi_{\{x\}})$, referring to the transition and reinforcement function definitions in Sec. 2.3. In the context of $\phi_{\{x\}}$ one abstract state represents all of the given measurements:

$$\hat{\mathbf{x}}^1 = \phi_{\{x\}}(\mathbf{x}^1) = \ldots = \phi_{\{x\}}(\mathbf{x}^4) = (x = 2.9)$$

Applying the transition and reinforcement functions to measurements represented by this state yields the following $\hat{\mathfrak{X}}_{t+1}$ set of next-time abstract state-reinforcement pairs:

$$\hat{\mathfrak{X}}^1_{t+1} = \{((x = 3), -1), ((x = 2.9), -5)\}$$

The abstract state $\hat{\mathbf{x}}^1$ represents four states near the vertical wall inside the labyrinth (refer to Fig. 2.1). The first and the last two states allow the agent to execute the RIGHT action, and in

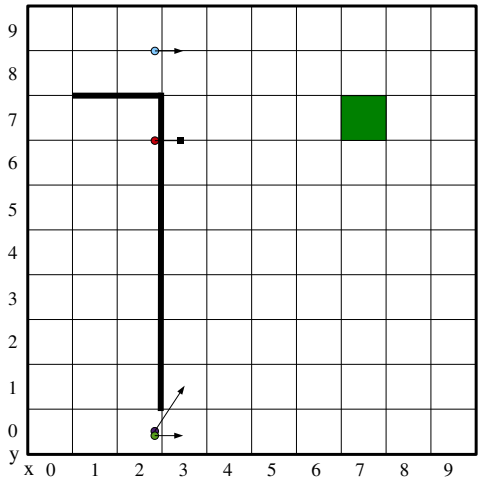

Figure 6.6: Four samples near the vertical wall in the Continuous Labyrinth. The arrows indicate the outcome of the RIGHT action. The first sample from the top will move to the right. The second one will hit the wall, and thus the agent will bounce back to the initial position. The third sample has a positive vertical velocity and thus will move to right and up. The last sample will move to the right, like the first one.

turn move to the right, changing its position to $x = 3$ and producing reinforcement equal to -1. The second state has the vertical wall to its right so an attempt to move to the right results in hitting the wall, and thus being bounced back to the initial position and producing reinforcement -5. This determines that $\hat{\mathfrak{X}}^1_{t+1}$ contains two different abstract state-reinforcement pairs. Thus, according to (6.2), $\Lambda(\hat{\mathfrak{X}}^1_{t+1}) = 1$. This implies, according to (4.10) that $A(\phi_{\{x\}}) = 1$ (other actions do not yield larger values in this case).

Consider now the second abstraction, $\phi_{\{x,y\}}$, and calculate $A(\phi_{\{x,y\}})$. The abstract states representing the given measurements are as follows:

$$\hat{\mathbf{x}}^1 = \phi_{\{x,y\}}(\mathbf{x}^1) = (x = 2.9, y = 9)$$
$$\hat{\mathbf{x}}^2 = \phi_{\{x,y\}}(\mathbf{x}^2) = (x = 2.9, y = 7)$$
$$\hat{\mathbf{x}}^3 = \phi_{\{x,y\}}(\mathbf{x}^3) = \phi_{\{x,y\}}(\mathbf{x}^4) = (x = 2.9, y = 0.5)$$

Consider again action $u =$ RIGHT. Applying the transition and reinforcement functions to these

measurements yields the following $\hat{\mathfrak{X}}_{t+1}$ sets of next-time abstract state-reinforcement pairs:

$$\hat{\mathfrak{X}}^1_{t+1} = \{((x=3, y=9), -1)\}$$
$$\hat{\mathfrak{X}}^2_{t+1} = \{((x=2.9, y=7), -5)\}$$
$$\hat{\mathfrak{X}}^3_{t+1} = \{((x=3, y=0.5), -1), ((x=3, y=1.5), -1)\}$$

Analogously to the reasoning for $\phi_{\{x\}}$ this implies that $A(\phi_{\{x,y\}}) = 1$, because of two different abstract state-reinforcement pairs in $\hat{\mathfrak{X}}^3_{t+1}$. Thus the algorithm would not extend the abstraction $\phi_{\{x\}}$ with the measurable y even though, by intuition, it should - because it resolves the ambiguity in the proximity of states $\mathbf{x}^1$ and $\mathbf{x}^2$.

□

The above example shows that the incremental process presented in 4.2 would not extend the abstraction $\phi_{\{x\}}$, by inserting the y measurable, even though it partially resolves the ambiguity for $\phi_{\{x\}}$ abstraction ($\Lambda(\hat{\mathfrak{X}}^1_{t+1}) = \Lambda(\hat{\mathfrak{X}}^2_{t+1}) = 0$, so without $\hat{\mathfrak{X}}^3_{t+1}$ the value of the ambiguity functional for $\phi_{\{x,y\}}$ is $A(\phi_{\{x,y\}}) = 0$). The set of samples in this example is as simple as possible to illustrate the issue - the same phenomenon occurs for large groups of samples.

Another important conclusion from this example is that the abstraction functions are in general incomparable when considering the whole space of measurables. Different measurables might be important in different parts of the space of measurables. That is why in the final version of the algorithm, after the incremental process has finished, the final result is evaluated using an average over a specifically chosen group of sample states. The following section presents a possible remedy to this problem.

6.4 Ambiguity sample groups

Recall from Sec. 5.4 that we estimate the inverse image ϕ^{-1} for some given abstraction ϕ by gathering groups of measurements using a proximity criterion determined by vector ε (5.2). Consider two abstraction functions $\phi_{\mathcal{J}}$ and $\phi_{\mathcal{K}}$ and assume that their index sets differ by one variable: $\|\mathcal{K} \setminus \mathcal{J}\| = 1$. Denote any abstract state yielded by $\phi_{\mathcal{J}}$ and $\phi_{\mathcal{K}}$ by $\hat{\mathbf{x}}^{(1)}$ and $\hat{\mathbf{x}}^{(2)}$ resp. As described in Sec. 5.4, for these abstractions, the groups of measurements that will be collected by our algorithm are denoted by $\mathcal{S}^{\hat{\mathbf{x}}}_{\phi_{\mathcal{J}}}$ and $\mathcal{S}^{\hat{\mathbf{y}}}_{\phi_{\mathcal{K}}}$. We want to compare values of the ambiguity functional for abstractions $\phi_{\mathcal{J}}$ and $\phi_{\mathcal{K}}$ locally, i.e. in the proximity of a particular abstract state yielded by the finer abstraction. To be able to do this, we match the groups of samples for both

abstractions by values of the common measurables and form pairs of groups, which we call *ambiguity samples* and define them as follows. Denote with $\mathbf{x}$ an arbitrary vector in $[\hat{\mathbf{x}}]$ and with $\mathbf{y}$ an arbitrary vector in $[\hat{\mathbf{y}}]$. Then:

$$\mathcal{A}^{\hat{\mathbf{x}}}_{\phi_{\mathcal{I}} \to \phi_{\mathcal{K}}} = \left\{ (\mathcal{S}^{\hat{\mathbf{x}}}_{\phi_{\mathcal{I}}}, \mathcal{S}^{\hat{\mathbf{y}}}_{\phi_{\mathcal{K}}}) : \underset{i \in \mathcal{I}}{\forall} |\mathbf{x}_i - \mathbf{y}_i| < \varepsilon_i \right\} \tag{6.3}$$

Note that the condition on the RHS is based only on measurables that are preserved by both abstractions, i.e. measurables from the smaller index set $\mathcal{I}$. There is one extra element in the index set $\mathcal{K}$ but it is ignored, and not used for vector $\mathbf{y}$. The purpose of comparing values of the ambiguity functional estimated from such ambiguity samples is to determine whether this extra measurable helps in making the abstraction less ambiguous.

The notation has the following meaning: the superscript $\hat{\mathbf{x}}$ denotes that this is an ambiguity sample group for the vector $\hat{\mathbf{x}}$ and the subscript $\phi_{\mathcal{I}} \to \phi_{\mathcal{K}}$ denotes that this contains pairs of groups gathered for $\phi_{\mathcal{I}}$ and $\phi_{\mathcal{K}}$ matched by values of their common measurables (i.e. ignoring only one extra measurable from $\mathcal{K}$).

This way the incremental abstraction process can verify if an extended abstraction ϕ^{ext} is better than the current abstraction ϕ in some particular point of ϕ's abstract state space, by comparing values of the ambiguity functional for sample group pairs separately.

Thought Experiment 6.4.1. To verify the idea presented in this section, we will now reevaluate the same scenario as in Example 6.3.1, i.e. the Continuous Labyrinth domain with the following measurables: x, y, v_x, v_y. Consider abstractions $\phi_{\{x\}}$ and $\phi_{\{x,y\}}$ again. The sample groups analyzed in this example are illustrated in Fig. 6.7. Assume the same proximity vector values: $\varepsilon = (0.001, 0.001)$ and the same threshold vector values: $\theta^{(x)} = (0.001, 0.001)$ and $\theta^{(r)} = 1$ and the following measurements that would have been generated for the purpose of estimating the ambiguity functional:

$$\mathbf{x}^1 = (x = 2.9, y = 9, v_x = 0, v_y = 0)$$
$$\mathbf{x}^2 = (x = 2.9, y = 7, v_x = 0, v_y = 0)$$
$$\mathbf{x}^3 = (x = 2.9, y = 0.5, v_x = 0, v_y = 0)$$
$$\mathbf{x}^4 = (x = 2.9, y = 0.5, v_x = 0, v_y = 1)$$

(refer to Fig. 2.1). Consider action $u = \text{RIGHT}$, and let us calculate $A(\phi^1)$. One abstract state representing all of the given measurements is as follows:

$$\hat{\mathbf{x}}^1 = \phi_{\{x\}}(\mathbf{x}^1) = \ldots = \phi_{\{x\}}(\mathbf{x}^4) = (x = 2.9)$$

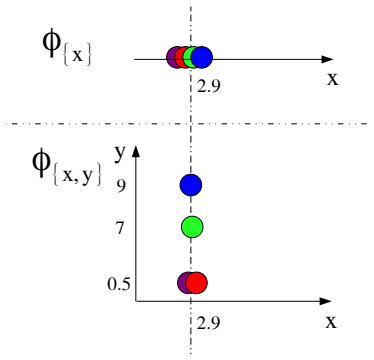

Figure 6.7: Four sample groups: one for $\phi_{\{x\}}$ abstraction (for the abstract state $\hat{\mathbf{x}}^1$), and three abstraction $\phi_{\{x,y\}}$. The measurements are drawn bigger than usual for clarity. Because values of abstract states for all measurables contained in the coarser abstraction are equal ($x = 2.9$) these groups form together an ambiguity sample $\mathcal{A}^{\hat{\mathbf{x}}^1}_{\phi_{\{x\}}\to\phi_{\{x,y\}}}$, containing all three pairs of the sample group for the first abstraction with three sample groups for the second abstraction. Points were moved slightly from their accurate positions not to obscure each other.

For the purpose of estimating the inverse image of $\phi_{\{x\}}$ the following sample group is created:

$$\mathcal{S}^{\hat{\mathbf{x}}^1}_{\phi_{\{x\}}} = \left\{\mathbf{x}^1, \mathbf{x}^2, \mathbf{x}^3, \mathbf{x}^4\right\}$$

Applying the transition and reinforcement functions to these measurements yields the following $\hat{\mathfrak{X}}_{t+1}$ set:

$$\hat{\mathfrak{X}}^1_{t+1} = \{((x = 3), -1), ((x = 2.9), -5)\}$$

As in the previous Example 6.3.1, the abstract state $\hat{\mathbf{x}}^1$ represents four states near the vertical wall inside the labyrinth (refer to Fig. 2.1). The first and the last two states allow the agent to execute the RIGHT action, and in turn move to the right, changing its position to $x = 3$ and producing reinforcement equal to -1. The second state has the vertical wall to its right so an attempt to move to the right results in hitting the wall, and thus being bounced back to the initial position and producing reinforcement -5. This determines that $\hat{\mathfrak{X}}^1_{t+1}$ contains two different abstract state-reinforcement pairs. Thus, according to (6.2), $\Lambda(\hat{\mathfrak{X}}^1_{t+1}) = 1$. This implies, according to (4.10) that $A(\phi_{\{x\}}) = 1$ (other actions do not yield larger values in this case).

Consider now the second abstraction, $\phi_{\{x,y\}}$ and calculate $A(\phi_{\{x,y\}})$. Abstract states repre-

senting the given measurements are as follows:

$$\hat{\mathbf{x}}^1 = \phi_{\{x,y\}}(\mathbf{x}^1) = (x = 2.9, y = 9)$$
$$\hat{\mathbf{x}}^2 = \phi_{\{x,y\}}(\mathbf{x}^2) = (x = 2.9, y = 7)$$
$$\hat{\mathbf{x}}^3 = \phi_{\{x,y\}}(\mathbf{x}^3) = \phi_{\{x,y\}}(\mathbf{x}^4) = (x = 2.9, y = 0.5)$$

Three sample groups are being created, each for one abstract state:

$$\mathcal{S}^{\hat{\mathbf{x}}^1}_{\phi_{\{x,y\}}} = \{\mathbf{x}^1\}$$
$$\mathcal{S}^{\hat{\mathbf{x}}^2}_{\phi_{\{x,y\}}} = \{\mathbf{x}^2\}$$
$$\mathcal{S}^{\hat{\mathbf{x}}^3}_{\phi_{\{x,y\}}} = \{\mathbf{x}^3, \mathbf{x}^4\}$$

Consider again action $u =$ RIGHT. Applying the transition and reinforcement functions to these measurements yields the following $\hat{\mathfrak{X}}_{t+1}$ sets of next-time abstract state-reinforcement pairs:

$$\hat{\mathfrak{X}}^1_{t+1} = \{((x = 3, y = 9), -1)\}$$
$$\hat{\mathfrak{X}}^2_{t+1} = \{((x = 2.9, y = 7), -5)\}$$
$$\hat{\mathfrak{X}}^3_{t+1} = \{((x = 3, y = 0.5), -1), ((x = 3, y = 1.5), -1)\}$$

Sample group for $\phi_{\{x\}}$ is matched with the sample groups for $\phi_{\{x,y\}}$, and create the following ambiguity samples for the single abstract state $\hat{\mathbf{x}}^1$ from abstraction $\phi_{\{x\}}$:

$$\mathcal{A}^{\hat{\mathbf{x}}^1}_{\phi_{\{x\}} \to \phi_{\{x,y\}}} = \left\{ (\mathcal{S}^{\hat{\mathbf{x}}^1}_{\phi_{\{x\}}}, \mathcal{S}^{\hat{\mathbf{x}}^1}_{\phi_{\{x,y\}}}), (\mathcal{S}^{\hat{\mathbf{x}}^1}_{\phi_{\{x\}}}, \mathcal{S}^{\hat{\mathbf{x}}^2}_{\phi_{\{x,y\}}}), (\mathcal{S}^{\hat{\mathbf{x}}^1}_{\phi_{\{x\}}}, \mathcal{S}^{\hat{\mathbf{x}}^3}_{\phi_{\{x,y\}}}) \right\}$$

Now, the values of the ambiguity functional are being compared separately for all three sample group pairs. In this case, $\Lambda(\hat{\mathfrak{X}}^1_{t+1})$ for $\phi_{\{x\}}$ calculated using the $\mathcal{S}^{\hat{\mathbf{x}}^1}_{\phi_{\{x\}}}$ sample group yields $A(\phi_{\{x\}}) = 1$. This is compared to $\Lambda(\hat{\mathfrak{X}}^1_{t+1})$ and $\Lambda(\hat{\mathfrak{X}}^2_{t+1})$ for $\phi_{\{x,y\}}$, calculated using sample groups $\mathcal{S}^{\hat{\mathbf{x}}^1}_{\phi_{\{x,y\}}}$ and $\mathcal{S}^{\hat{\mathbf{x}}^2}_{\phi_{\{x,y\}}}$ respectively. The result for both of them is $A(\phi_{\{x,y\}}) = 0$, and the last sample group for $\hat{\mathfrak{X}}^3_{t+1}$) yields $A(\phi_{\{x,y\}}) = 1$. The comparison for the the first two pairs justifies extending the abstraction from $\phi_{\{x\}}$ to $\phi_{\{x,y\}}$, successfully including the y measurable in the result.

□

This experiment, along with Experiment 5.3.1 supports the third second-thesis, namely that:

the proposed notion of *ambiguity* correctly reflects the quality of a solution of a variable selection task.

Before the algorithm can be used, we must find a way of estimating the threshold parameters $\theta^{(x)}$ and $\theta^{(r)}$. The following section addresses this issue.

6.5 Determining the threshold parameters

Similarly to parameter ε (5.2), we can again estimate the average distances between consecutive measurements, and use them as values for the threshold parameters:

$$\theta^x = \mathfrak{E}|\mathbf{x}(\mathbf{t}) - \mathbf{x}(\mathbf{t}-\mathbf{1})|/\rho \tag{6.4}$$
$$\theta^r = \mathfrak{E}|r(t) - r(t-1)|/\rho$$

where $\rho > 0$ is the resolution parameter, introduced in (5.2).

Computer Experiment 6.5.1. In this experiment we test the complete algorithm presented in Listing 9 on the Continuous Labyrinth problem with the following set of measurables: x, y, v_x, v_y, $x+y$, $|x-y|$, xy, $q_x = \sqrt{x}, q_y = \sqrt{y}$.

First, we gather 100000 samples to calculate vector ε as in (5.2), and the threshold values $\theta^{(x)}$ and $\theta^{(r)}$ as in (6.9). Parameter ρ was set to 3. We start with the null abstraction: $\phi = \phi_\emptyset$. Then we collect samples and every 500 steps try to reduce or extend the current abstraction function ϕ. The results are presented in Tab. 6.1. Because all runs have failed and abstraction paths were very long, reductions were omitted for readability.

The reason for these failures is as follows: consider two sample groups, one for ϕ and one for ϕ', where ϕ' has is a result of including an additional measurable in the index set of ϕ. Samples aggregated for the purpose of evaluating ϕ' have more dimensions (measurables) than these collected for ϕ. The more dimension, the smaller probability of measurements to reoccur. Thus, there are always more samples within the sample group with fewer measurables. Also, if a pair of vectors is equal with respect to the finer abstraction, it implies that they are also equal with respect to the coarser abstraction, but not the other way around. This leads to the situation where two abstractions are being compared, but values of the ambiguity functional are estimated for them with different number of measurements. This can be compensated if the groups of vectors are very large. However, for the continuous case, in practice, this is unfeasible - waiting for duplicate vectors is very time consuming. Another possibility is to collect samples in two phases, and ensuring that they all have equal cardinality. The following section presents that idea.

□

Table 6.1: Results of the algorithm with threshold parameter estimation, for the 9-dimensional Continuous Labyrinth problem.

Run #	Example decision sequence	Result	Ambiguity	Correctness
1	$+x, +v_x, +s, +q_y, +m, +q_x, \ldots$	not finished	not calculated	✖
2	$+x, +y, +v_x, +v_y, +s, +d, \ldots$	not finished	not calculated	✖
3	$+x, +v_x, +m, +q_x, +d, +q_y, +s, \ldots$	not finished	not calculated	✖

6.6 Two-phase ambiguity sample group collection

In Sec. 6.4 we proposed to accumulate similar measurements into *ambiguity samples*. In the context of evaluating whether an abstraction $\phi_{\mathcal{I}}$ should be extended to $\phi_{\mathcal{K}}$ (where set $\mathcal{I} \subset \mathcal{K}$, and $\mathcal{K}$ has exactly one additional element), we consider two sample groups: one for each abstraction. We denote this pair of sample groups, for one particular abstract state $\hat{\mathbf{x}}$ by $\mathcal{A}^{\hat{\mathbf{x}}}_{\phi_{\mathcal{I}} \to \phi_{\mathcal{K}}}$ (see (6.3)). It is a set of tuples, each consisting of two groups of similar measurements, measurements $\mathcal{S}_{\phi_{\mathcal{I}}}$, similar in the context of abstraction $\phi_{\mathcal{I}}$, and measurements $\mathcal{S}_{\phi_{\mathcal{K}}}$, similar in the context of abstraction $\phi_{\mathcal{K}}$ (see (5.3)). The problem encountered at the end of Sec. 6.4 stems from the fact that the number of measurements in these groups is usually different. To deal with this issue, we propose to first collect groups $\mathcal{S}_{\phi_{\mathcal{I}}}$ for the abstraction with smaller number of measurables, until we reach some predefined, desired number of groups (GroupCount parameter), and number of measurements in these groups (NeighbourCount parameter). We call this phase the *initial collection phase*. After this phase is finished, we reuse all of the observed vectors from groups $\mathcal{S}_{\phi_{\mathcal{I}}}$ and use them to form an initial set of groups $\mathcal{S}_{\phi_{\mathcal{K}}}$. We then start the *secondary collection phase*, to continue collecting samples in the same manner as in the *initial collection phase*, until the same number of groups and neighbours in both groups is reached. Then the value of the ambiguity functional can be estimated using groups with equal number of measurements.

Computer Experiment 6.6.1. With this modification, the algorithm presented in the Listing 3 yields the correct results, presented in Tab. 6.2. The columns are the same as described in Experiment 5.5.1, with the difference that not all of the resulting abstractions with ambiguity functional equal to 0 are presented in the second column, for brevity. Taking all runs into account, all possible solutions were found, and they were as follows:

$$(x,y,v_x,v_y),(x,v_x,v_y,s),(x,v_x,v_y,d),(x,v_x,v_y,m),(y,v_x,v_y,s),(y,v_x,v_y,d),$$
$$(y,v_x,v_y,m),(x,v_x,v_y,q_y),(y,v_x,v_y,q_x),(v_x,v_y,s,q_x),(v_x,v_y,s,q_y),$$
$$(v_x,v_y,d,q_x),(v_x,v_y,d,q_y),(v_x,v_y,m,q_x),(v_x,v_y,m,q_y),(v_x,v_y,q_x,q_y)$$

In theory, combinations of measurables involving one coordinate (x or y) and d or m (being defined as: $d = |x-y|$ and $m = xy$) do not give full information on agent's position. However, situations in which values of such combinations of measurables are ambiguous are almost impossible to occur. This is because probability of a value of a continuous random falling into a measure-zero set is zero. For example, the (x,m) abstraction correctly represents agent's position in the labyrinth, except when agent is on the line $x = 0$. Similarly, for (d,q_y): for each q_y, y is fixed and equal to $y = q_y^2$, and ambiguous measurements lie on the line: $f(x) = 2y - x$. Because the correct solution conditions are violated only for vectors from a measure-zero set, we regard these solutions as correct.

Table 6.2: Results of the algorithm with two-phase ambiguity sample group collection for the 9-dimensional Continuous Labyrinth problem. Parameters were set for the following values: GroupCount $= 3$, NeighbourCount $= 5, \rho = 3$.

Run #	Examples of the results	Ambiguity	Correctness	Coverage
1	$(x,y,v_x,v_y),(x,v_x,v_y,s),(x,v_x,v_y,m),(v_x,v_y,s,q_y)$	0	✔	32%
2	$(x,v_x,v_y,m),(x,v_x,v_y,s),(y,v_x,v_y,s),(v_x,v_y,d,q_y)$	0	✔	38%
3	$(v_x,v_y,d,q_x),(v_x,v_y,m,q_y),(x,v_x,v_y,m),(x,v_x,v_y,s)$	0	✔	36%
4	$(x,v_x,v_y,d),(y,v_x,v_y,d),(v_x,v_y,d,q_y),(x,v_x,v_y,s)$	0	✔	35%
5	$(v_x,v_y,m,q_x),(v_x,v_y,m,q_y),(x,v_x,v_y,m),(x,v_x,v_y,d)$	0	✔	33%
6	$(x,v_x,v_y,s),(y,v_x,v_y,s),(x,y,v_x,v_y),(v_x,v_y,d,q_y)$	0	✔	32%
7	$(v_x,v_y,s,q_y),(v_x,v_y,s,q_x),(x,v_x,v_y,q_y),(x,v_x,v_y,m)$	0	✔	33%
8	$(y,v_x,v_y,q_x),(v_x,v_y,q_x,q_y),(v_x,v_y,m,q_y),(v_x,v_y,d,q_y)$	0	✔	33%
9	$(v_x,v_y,s,q_y),(v_x,v_y,m,q_x),(x,v_x,v_y,m),(x,v_x,v_y,s)$	0	✔	36%
10	$(y,v_x,v_y,q_x),(v_x,v_y,s,q_y),(x,v_x,v_y,m),(x,v_x,v_y,q_y)$	0	✔	34%

□

Computer Experiment 6.6.2. We also evaluate the *two-phase* collection version of the algorithm in the Cart-Pole Swing-Up task. The results presented in Tab. 6.3. The columns are the

same as described in Experiment 5.5.1. This time, for the Cart-Pole Swing-Up environment, the algorithm fails. The problem is very subtle, and occurred only in only one of 10 runs. It is caused by the fact that the transition function has a large Lipschitz constant, and even for correct abstraction the outcomes of the transition function for similar measurements can be very different. There is a very low probability for this situation to occur, as it happens only for angles close to 0, for example, when the sin function changes its value faster than in other part of its domain. In such case, even when having complete information about the pole's angle from sin and cos measurables, the tan measurable is added. The following section presents a possible solutions to this problem.

Table 6.3: Results of the algorithm with two-phase ambiguity sample group collection for the 10-dimensional Cart-Pole Swing-Up with a simple reinforcement function problem. Parameters were set for the following values: $\text{GroupCount} = 3, \text{NeighbourCount} = 5, \rho = 3$.

Run #	Examples of the results	Ambiguity	Correctness	Coverage
1	(s,a,j)	0	✔	37%
2	(c,a,j)	0	✔	37%
3	(s,c,a)	0	✔	38%
4	$(c,a,j),(s,a,j)$	0	✔	37%
5	(s,c,a)	0	✔	37%
6	(c,a,j)	0	✔	37%
7	(s,c,a)	0	✔	38%
8	(s,a,j)	0	✔	37%
9	$(v,s,c,a),(v,c,a,j),(s,c,a,k),(s,a,j,k)$	0	✖	62%
10	(s,a,j)	0	✔	37%

□

6.7 Estimating the threshold parameters with variance

In Sec. 6.5 we presented a simple way of estimating the threshold parameters: thresholds are equal to the average distances between the subsequent values of measurables. When the transition function is a contraction, or at least its Lipschitz constant is not too large this is sufficient, as was presented in the previous section with the results for the Continuous Labyrinth

environment (Tab. 6.2). To account for the case when the transition function is not a contraction, we propose to also estimate the standard deviation σ of these distances. Therefore, denoting with θ any of the threshold parameters, the resulting threshold values will be calculated as: $\theta + \kappa\sigma$, where κ is a parameter. Also, for different measurables the transition function may introduce different expansion, so this estimation is done separately for each measurable.

The interval $(\theta - \kappa\sigma, \theta + \kappa\sigma)$ does not always make sense, as its left boundary may be negative and distance should be non-negative. Thus we propose to estimate average logarithms of distances instead, and calculate the resulting value of the threshold vector using the exp function. The equations for calculating mean and variance components for $\theta^{(x)}$ and $\theta^{(r)}$ are as follows:

$$E_{\ln \delta \mathbf{x}} = \mathscr{E} \ln \frac{|\mathbf{x}(\mathbf{t}) - \mathbf{x}(\mathbf{t}-\mathbf{1})|}{\rho} \tag{6.5}$$

$$E_{\ln \delta r} = \mathscr{E} \ln \frac{|r(t) - r(t-1)|}{\rho} \tag{6.6}$$

$$V_{\ln \delta \mathbf{x}} = \mathscr{V} \ln \frac{|\mathbf{x}(\mathbf{t}) - \mathbf{x}(\mathbf{t}-\mathbf{1})|}{\rho} \tag{6.7}$$

$$V_{\ln \delta r} = \mathscr{V} \ln \frac{|r(t) - r(t-1)|}{\rho} \tag{6.8}$$

where $\rho > 0$ is the resolution parameter, introduced in (5.2). The threshold values for each measurable and for the reinforcement can be thus then defined as:

$$\theta = \left(\exp\left(E_{\ln \delta \mathbf{x}} - \kappa \sqrt{V_{\ln \delta \mathbf{x}}} \right), \exp\left(E_{\ln \delta r} + \kappa \sqrt{V_{\ln \delta r}} \right) \right) \tag{6.9}$$

where for a vector $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$, $\exp(\mathbf{x})$ denotes the vector $(\exp(\mathbf{x}_1), \ldots, \exp(\mathbf{x}_n))$.

Computer Experiment 6.7.1. In this experiment, we evaluate the algorithm with added parametrized estimation of the threshold parameters. In Tab. 6.4 we present all best results for all of 10 runs of the algorithm, for different values of the κ parameter. Large values of this parameter introduce large tolerance for divergence between measurements and reinforcements resulting from the transition and reinforcement functions. The smaller the parameter value, the smaller the threshold, and more measurables are needed to maintain determinateness of the model.

□

There are two remaining issues with this approach. The first one is a high coverage of the solution space (the term coverage is defined in Experiment 5.5.1). The idea of order estimation, presented in Sec. 5.6 remains valid, but it does not help with the algorithm evaluating more

Table 6.4: Results of the algorithm with two-phase ambiguity sample group collection for the 10-dimensional Cart-Pole Swing-Up with a simple reinforcement function problem. Parameters were set for the following values: GroupCount $= 3$, NeighbourCount $= 5, \rho = 3$.

Run #	κ	Best results	Ambiguity	Correctness	Coverage
1-10	6	$()$	0	✖	0%
1-10	4,5	$(c),(j)$	0	✖	5%
1-10	3	$(c),(v,c),(c,h)$	0	✖	17%
1-10	$[0.4,2)$	$(c,a,j),(s,a,j),(s,c,a),(a,j,k),(s,a,k)$	0	✔	37%
1-10	$[0,0.4)$	$(s,c,a,k),(s,a,j,k),\ldots$	0	✖	37%

than a half of possible 3-dimensional combinations. The second, is long processing time: three hours on the average. To cover a smaller part of the solution space, we run the experiments with larger values of the NeighbourCount and GroupCount parameters. To significantly reduce the processing time, we evaluate only a random subset of sample groups in the *secondary collection phase*. Reasoning behind this is as follows: Assume for simplicity that if a measurable *shatters* (Sec. 4.3) a sample group it produces only two sample groups. Then, if we have GroupCount sample groups accumulated in the *initial collection phase*, then after inserting each measurable we get 2GroupCount sample groups. This will produce $2^k \times$ GroupCount sample groups for k measurables, which tends to be expensive, especially at the beginning of the process, where many measurables are added accidentally. We propose to process only $2 \times$ GroupCount samples at maximum in the *secondary collection phase*. Additionally, we do not consider all possible extensions, but only those with the highest improvement in the value of the ambiguity functional.

Nearest Neighbour problem performance note When using large values of the NeighbourCount and GroupCount the performance degrades significantly. The solution is to maintain separate data storage for each evaluated abstraction, and solve the nearest neighbours problem separately for each of them.

Computer Experiment 6.7.2. We evaluate the version of the the algorithm with the aforementioned improvements, with larger values of the NeighbourCount and GroupCount parameters on both versions of the Cart-Pole Swing-Up environment. The results are presented in Tab. 6.5 and 6.6. As expected, higher values of the NeighbourCount and GroupCount parameters

result in very low solution space coverage for both solved tasks (the term coverage is defined in Experiment 5.5.1). Lower solution space coverage results in smaller algorithm running time. However, at the same time, values of the NeighbourCount and GroupCount parameters can not be too large, because gathering more sample groups with more neighbours increases the running time. This is especially important in the context of experiments without the use of the simulator assumption (Sec. 3.2) that will be presented in Chapt. 7, because waiting for the same values of measurements to reoccur is very time consuming. A simple, intuitive, way to find the satisfactory values of the NeighbourCount and GroupCount parameters is to increase their values and observe how long does it take for the algorithm to reach two-dimensional or three-dimensional abstractions. The relation between the number of measurables and the time needed to gather samples to calculate the ambiguity functional is exponential. Thus, if the time to reach two-dimensional abstractions is considered long, then waiting for larger abstractions will surely be unsatisfactory, and the values of the NeighbourCount and GroupCount parameters need to be smaller.

Table 6.5: Results of the algorithm with performance improvements for the 10-dimensional Cart-Pole Swing-Up with a simple reinforcement function problem. Parameters were set for the following values: $\text{GroupCount} = 20, \text{NeighbourCount} = 20, \rho = 3, \kappa = 1$.

Run #	Results	Ambiguity	Correctness	Coverage
1	$(s,c,a),(c,a,j),(c,a,k)$	0	✔	1.1%
2	$(s,c,a),(c,a,j),(c,a,k)$	0	✔	1.1%
3	$(s,c,a),(c,a,j)$	0	✔	1.1%
4	$(s,c,a),(c,a,j),(c,a,k)$	0	✔	1.1%
5	$(s,c,a),(c,a,j),(c,a,k)$	0	✔	1.1%
6	$(s,c,a),(c,a,j),(c,a,k)$	0	✔	1.1%
7	$(s,c,a),(c,a,j),(c,a,k)$	0	✔	1.1%
8	$(s,c,a),(c,a,j),(c,a,k)$	0	✔	1.1%
9	$(s,c,a),(c,a,j)$	0	✔	1.1%
10	$(s,c,a),(c,a,j),(c,a,k)$	0	✔	1.1%

□

Table 6.6: Results of the algorithm with performance improvements for the 10-dimensional Cart-Pole Swing-Up with a complex reinforcement function problem. Parameters were set for the following values: GroupCount $= 20$, NeighbourCount $= 20$, $\rho = 3$, $\kappa = 1$. All results were correct, but for brevity we present only the first three from the output.

Run #	First three results	Ambiguity	Correctness	Coverage
1	$(x,c,a,h,k),(v,c,a,h,k),(c,a,h,k,l)$	0	✔	2.9%
2	$(x,v,c,a,j),(x,c,a,h,j),(v,c,a,h,j)$	0	✔	4.2%
3	$(v,c,a,k,l),(c,a,h,k,l),(x,c,a,h,j)$	0	✔	2.9%
4	$(v,c,a,j,l),(c,a,h,j,l)$	0	✔	3.5%
5	$(x,v,s,c,a),(x,s,c,a,h),(v,s,c,a,h)$	0	✔	4.5%
6	$(x,s,c,h,l),(x,c,a,h,l),(x,c,h,j,l)$	0	✔	4.8%
7	(v,c,a,j,l)	0	✔	3.6%
8	$(x,c,a,h,k),(v,c,a,h,k),(c,a,h,k,l)$	0	✔	2.8%
9	$(x,v,c,a,l),(x,c,a,h,l)$	0	✔	3.9%
10	$(x,c,a,h,k),(v,c,a,h,k),(c,a,h,k,l)$	0	✔	2.9%

6.8 Summary

The proposed algorithm, presented in Listing 9, searches through the space of all possible subsets of all measurables. It aims to narrow down the search using incremental *extensions* and *reductions*. To additionally speed up the calculations, for each set of measurables we evaluate all possible extensions and reductions simultaneously (i.e., a set of abstraction paths is processed in parallel, see Sec. 5.5).

The algorithm depends on the following parameters:

1. resolution parameter ρ (deadband), used for calculating ε, introduced in Sec. 5.4 and values of vector $\theta^{(x)}$ and scalar $\theta^{(r)}$, introduced in Sec. 4.2.

2. threshold parameter κ, introduced in Sec. 6.7

3. GroupCount parameter - the number of collected sample groups (defined in (5.3)), introduced in Sec. 6.6

4. NeighbourCount parameter - the number of measurements in each sample group, introduced in Sec. 6.6

5. MaxFailureCount parameter to determine that when algorithm should stop, introduced in Sec. 5.4

We employ the following intuition for finding satisfactory values of these parameters. Values of the NeighbourCount and GroupCount parameters are found in a way described in Experiment 6.7.2: we increase their values step by step for as long, as the algorithm reaches two-dimensional abstractions in "short" time. We stop, when this time becomes significantly prolonged. For parameter ρ, we start with a small value (e.g. 0.1) and increase it if the solution is wrong (i.e., typically, algorithm includes all or almost all measurables). We use small values of the MaxFailureCount parameter (e.g. 5) if different reinforcement values are easy to be observed by the algorithm (e.g. in the Cart-Pole Swing-Up environment every movement of the pole changes the value of the reinforcement) and large (e.g. 20 or 50) when different reinforcements are rare (e.g. in the Mountain Car environment, only the target position of the car induces different reinforcement than the single value observed most of the time). Large values of the NeighbourCount parameter make possible to use smaller values of the MaxFailureCount parameter, because waiting for more neighbour samples increases the chance of rare events to occur.

The experimental results presented in this Chapter support the third sub-thesis, namely that:

the proposed approach, based on the notion of ambiguity correctly solves variable selection tasks for RL, including tasks with continuous transition and reinforcement functions.

With all sub-theses advocated, we conclude that the main thesis of this work holds:

our ambiguity-based, bottom-up approach is a valid solution of the Reinforcement Learning variable selection problem

Algorithm 9 NADA: Neighborhood Ambiguity Driven Abstraction

indexSets $\leftarrow \emptyset$
result $\leftarrow \emptyset$
while $|$indexSets$|$ 0 **do**
  interact with the plant using a random policy
  collect samples: $\langle \mathbf{x}(\mathbf{t}-\mathbf{1}), u(t-1), \mathbf{x}(\mathbf{t}), r(t) \rangle$
  accumulate samples into groups (Sections 5.4 and 6.6)
  if all there are GroupCount groups, with NeighbourCount number of states each **then**
    for all $\mathcal{I} \in$ indexSets **do**
      estimate $A(\phi_{\mathcal{I}})$ (see (5.1) or (6.2))
      if $A(\phi_{\mathcal{I}}) > A(\phi_{\mathcal{I} \cup i})$ for some measurable i **then**
        indexSets $\leftarrow$ indexSets $\cup (\mathcal{I} \cup i)$
        add samples, which maximize A functional for $\mathcal{I}$ to the *extension memory* collection
        indexSets $\leftarrow$ indexSets $\setminus (\mathcal{I})$
      end if
      if $A(\phi_{\mathcal{I}}) = A(\phi_{\mathcal{I} \setminus i})$ for some measurable i **then**
        if the condition holds also for samples from *extension memory* collection **then**
          indexSets $\leftarrow$ indexSets $\cup (\mathcal{I} \setminus i)$
          indexSets $\leftarrow$ indexSets $\setminus (\mathcal{I})$
        end if
      end if
      if $\mathcal{I}$ was not extended for MaxFailureCount number of attempts **then**
        indexSets $\leftarrow$ indexSets $\setminus (\mathcal{I})$
        result $\leftarrow$ result $\cup \mathcal{I}$
      end if
    end for
  end if
end while

Chapter 7

Evaluation of the abstraction algorithm without the simulator assumption

Up to this chapter we have been using a generator whenever some measurements were missing, to fill a sample group with required number of neighbour points. Having shown that the approach works when given all required information, in this section we run the algorithm on a purely random set of transitions. To maintain reasonable running time of the algorithm, we introduce the following modifications:

1. We ignore the fact that some measurements are missing from a sample group if they did not occur for large number of N_{max} steps, and calculate the values of the ambiguity functionals using only the information that is currently available.

2. Only the abstractions that could not be extended in MaxFailureCount attempts are considered a result, regardless of the value of the ambiguity functional.

The first modification is the most important in terms of algorithm's time complexity: time needed for similar measurements to occur grows exponentially with the number of dimensions. Given that also some measurements occur very rarely (e.g. reaching the target spot in Continuous Labyrinth), it is impractical to wait for all samples when estimating the values of the ambiguity functional for more than 2 dimensions.

Second modification makes the result dependent on the fact that no measurements were encountered in which the abstraction can be improved (in terms of transition function or reinforcement functions determinateness) instead of the value of the ambiguity functional. Because of the first modification, estimated values of the ambiguity functional are less reliable in terms

of comparing the abstractions in context of the whole space of measurables.

We set the parameters according to the intuition presented in Sec. 6.8, but use larger values of the MaxFailureCount parameter due to the first modification presented above, which decreases the accuracy of estimation of the ambiguity functional.

7.1 Results for a deterministic, discrete domain

Computer Experiment 7.1.1. First, we test the proposed algorithm on the enhanced version of the Discrete Labyrinth. The enhancement consists of making the labyrinth larger, namely we consider a grid with 500×500 cells, which gives a domain with 250000 measurements, instead of 100. The results are presented in Tab. 7.1. The GroupCount parameter was set to 100 so it was possible to gather samples with situations in which the agent hits the wall, or collects the reward. Alternatively, large value of the MaxFailureCount could have been used. Each run took about two hours, on average.

Table 7.1: Results of the approximate algorithm on the 7-dimensional, 500×500 Discrete Labyrinth problem. Parameters were set for the following values: GroupCount $=$ $100, \text{NeighbourCount} = 10, \rho = 0.1, \text{MaxFailureCount} = 10$, $N_{max} = 1000$.

Run #	Example decision sequence	Results	Ambiguity	Correctness	Coverage
1	$+y,+s$	$(y,s),(x,s)$	0	✔	3.7%
2	$+x,+s$	$(x,s),(x,y)$	0	✔	2.8%
3	$+s,+x$	$(x,s),(x,y)$	0	✔	3.8%
4	$+x,+s$	$(x,s),(x,y)$	0	✔	2.2%
5	$+x,+s$	$(x,s),(x,y)$	0	✔	2.9%
6	$+x,+s$	$(x,s),(x,y)$	0	✔	2.9%
7	$+x,+s$	$(x,s),(x,y)$	0	✔	2.2%
8	$+x,+s$	$(x,s),(x,y)$	0	✔	2.5%
9	$+x,+s$	$(x,s),(x,y)$	0	✔	2.5%
10	$+x,+s$	$(x,s),(x,y)$	0	✔	2.4%

□

7.2 Results for stochastic, discrete domain

In this section we additionally evaluate the proposed algorithm on a stochastic domain, namely the Coffee Task environment. The main difficulty in this task stems from the fact that the transition function is not deterministic. The results are presented in Tab. 7.2. The values of the GroupCount and NeighbourCount were significantly increased to take into account many possible outcomes of the transition and reinforcement functions, even for a correct model. In case of such an environment the ambiguity functional will never be 0, however it reaches the lowest possible value only for correct solutions. That is why all final abstractions have ambiguity equal to 3. Each measurement in the best possible abstraction can have 4 possible outcomes - regardless of the chosen action it may fail or not, giving two possible values of some measurable and it may start raining or not, regardless of the chosen action or any other measurable, which gives also two possible values of another measurable. Two possible values of two binary measurables give 4 possible combinations, and taking into account the -1 term from (5.1) we get the minimal ambiguity functional equal to 3. The reason why the algorithm is still able to find the correct abstractions is that for some points in the space of measurables some measurables are still relevant on the average. For example, without the umbrella the robot sometimes gets wet, and sometimes not. But if the robot takes the umbrella, it will never become wet, regardless if the action *GO* will succeed or fail, and regardless of if it is raining or not. The knowledge about having the umbrella makes the transition function less *ambiguous*.

7.3 Results for continuous domains

Computer Experiment 7.3.1. The algorithm presented in Listing 9, along with modifications presented at the beginning of this chapter, was evaluated for continuous domains, namely the Continuous Labyrinth domain, both versions of the Cart-Pole Swing-Up domain, both versions of the Mountain Car domain and the Pinball domain. The results are presented in Tables 7.3, 7.4, 7.5, 7.6 and 7.8 respectively.

The results for the Continuous Labyrinth are presented in Tab. 7.3. The simplicity of this domain allows to reach correct results even for the resolution parameter ρ as small as 0.1. When $\rho < 1$ chances of measurements reoccurring are greatly improved, which significantly reduces the running time. Always the same, correct solution had the lowest value of the ambiguity functional. The decision sequence was always that first the coordinates (two of measurables

Table 7.2: Results for the Coffee Task problem. Parameter were set to the following values: GroupCount $= 50$, NeighbourCount $= 30$, $N_{max} = 1000$.

Run #	Decision sequence	Result	Ambiguity	Correctness	Coverage
1	+SW, +SL, +SC, +SU, +SR	SC,SR,SU,SL,SW	3	✔	22%
2	+SW, +SL, +SU, +SC, +SR	SC,SR,SU,SL,SW	3	✔	13%
3	+SL, +SW, +SU, +SC, +SR	SC,SR,SU,SL,SW	3	✔	22%
4	+SL, +SW, +SC, +S9, +SR	SC,SR,SL,SW,S9	3	✔	19%
5	+SL, +SW, +SU, +SC, +SR	SC,SR,SU,SL,SW	3	✔	20%
6	+SL, +SW, +SC, +SU, +SR	SC,SR,SU,SL,SW	3	✔	17%
7	+SL, +SW, +S9, +SC, +SR	SC,SR,SL,SW,S9	3	✔	16%
8	+SL, +SC, +SL, +S9, +SR	SC,SR,SL,SW,S9	3	✔	14%
9	+SL, +SC, +SW, +SU, +SR	SC,SR,SU,SL,SW	3	✔	17%
10	+SL, +SW, +SC, +S9, +SR	SC,SR,SL,SW,S9	3	✔	10%

x, y, s, d and m, q_x, q_y) were added - to correctly predict reaching the rewarded square, or hitting the wall. Then, the algorithm added the velocity coordinates, to improve prediction of the values of coordinates. In the lists with final abstractions, some abstractions with only one velocity measurable were present in most runs - however they always had significantly larger values of the ambiguity functional. The average computation time for each run was under one minute.

The results for the Cart-Pole Swing-Up with a simple reinforcement function, shown in Tab. 7.4, also demonstrate the correct behaviour of the algorithm. Setting ρ as small as in the case of the Continuous Labyrinth yields incorrect results - because such resolution makes the effect of the angular velocity unobservable. As a compromise between accuracy and running time, we chose $\rho = 1$. The results are presented in the Tab. 7.4. In all runs, the insertion of the angular velocity measurable a is always preceded by one or two angular measurables (s, c j, or k). This is because a is not related with the reinforcement values. After inserting the angular measurables, the angular velocity is then required to correctly predict the behaviour of the pole's angle, hence the algorithm correctly inserts this measurable. The simulations took about 8 hours, on average, for each run.

Table 7.5 presents the results for the Cart-Pole Swing-Up with a complex reinforcement function. In case of this domain, the value of the ρ parameter was set to 0.2. It is much

lower than in the experiment with a simple reinforcement function, even though this domain is more complex. The reason for this is that the ambiguous measurements are more likely to be observed, because of the nonlinear behaviour near the cart's boundaries. The penalty for hitting the bound is received by the agent when the cart or the pole hit the boundary and this makes the measurables responsible for cart's position and pole's angle easy to be identified as nonredunant. After the position and pole measurables were added, the algorithm correctly includes the velocity and the angular velocity, respectively. The average processing time for each run was about 20 hours.

In Tabs. 7.6, 7.7 we present results for the simple Mountain Car and the stochastic Mountain Car environments, respectively. In both cases, in each run, a correct set of observables was determined. Because this domain is simple in terms of the number of underlying state variables (two: car's position and its velocity), we test the algorithm against the largest number of given observables, namely twenty. The main difficulty in this environment is that the agent receives constant reinforcement most of the time and the only event that makes a difference is reaching the goal, which does not happen often. Also, in the stochastic version the transition function is not deterministic. Because of that the number of neighbours in sample groups was increased to 100 and the MaxFailureCount parameter was set to 20 and 30 for the simple and stochastic versions, respectively. With these settings the algorithm had the chance to observe different values of the reinforcement function before concluding that its constant and no observations are needed, and the stochastic effects of the transition functions could be averaged. Note that the algorithm consistently avoids ambiguous variables like $F_9 = x^2$ or $F_{10} = v^2$ (Sec. 2.5). The simulations for each run took about twenty minutes for the simple version, and about eight hours for the stochastic version, on average.

The results for the Pinball environment are shown in Tab. 7.8. As in other experiments, in all runs the algorithm returned a correct result, that contains all necessary information about the agent, i.e. its two-dimensional position and two-dimensional velocity. Note that the algorithm always starts with inserting position-related observables related, as the reinforcement functions directly depends only on agent's position. Only after determining that the position is needed to make reinforcement values deterministic, velocity-related observables are inserted. Simulation time was two hours for each run, on average.

□

Table 7.3: Results of the approximate algorithm on the 9-dimensional Continuous Labyrinth problem. Parameters were set for the following values: GroupCount $= 10$, NeighbourCount $= 5$, $\rho = 0.1$, MaxFailureCount $= 5$, $N_{max} = 200000$.

Run #	Decision sequence	Result	Correctness	Coverage
1	$+x,+y,+v_x,+v_y$	(x,y,v_x,v_y)	✔	12%
2	$+y,+q_x,+x,-q_x,+v_y,+v_x$	(x,y,v_x,v_y)	✔	15%
3	$+y,+x,+v_x,+v_y$	(x,y,v_x,v_y)	✔	14%
4	$+x,+y,+v_x,+q_x,+v_y,-q_x$	(x,y,v_x,v_y)	✔	12%
5	$+q_x,+y,+x,-q_x,+v_x,+v_y$	(x,y,v_x,v_y)	✔	13%
6	$+x,+y,+v_x,+v_y$	(x,y,v_x,v_y)	✔	11%
7	$+x,+q_y,+v_x,+y,-q_y,+v_y$	(x,y,v_x,v_y)	✔	13%
8	$+y,+x,+v_y,+v_x$	(x,y,v_x,v_y)	✔	11%
9	$+y,+x,+v_x,+v_y$	(x,y,v_x,v_y)	✔	13%
10	$+x,+y,+v_x,+v_y$	(x,y,v_x,v_y)	✔	12%

7.3.1 Results for an alternative version of the Λ function

In this section we present an alternative method of calculating the ambiguity functional for continuous transition and reinforcement functions.

7.4 Distance-based ambiguity functional for continuous transition and reinforcement functions

Instead of saying that abstract state's ambiguity depends on the number of next abstract states reached with the same action (as in Sec. 6.1), we measure how far from each other these next abstract states are. To additionally take into account the expansion factor of the transition function, we do not want to interpret small divergence between next states and reinforcements as an ambiguity. Thus, for every measurable, every distance smaller than the appropriate threshold vector value is reset to 0. For simplicity, let us denote a concatenation of the threshold vector $\theta^{(x)}$ with the single element $\theta^{(r)}$ with just θ:

$$\theta = (\theta^{(x)}, \theta^{(r)})$$

Table 7.4: Results of the approximate algorithm on the 10-dimensional Cart-Pole Swing-Up problem with a simple reinforcement function. Parameters were set for the following values: GroupCount $= 5$, NeighbourCount $= 10$, $\rho = 1$, $\kappa = 0.3$, MaxFailureCount $= 20$, $N_{max} =$ 200000.

Run #	Decision sequence	Results	Correctness	Coverage
1	+s, +c, +a	(s,c,a)	✔	1.0%
2	+j, +c, +a	(c,a,j)	✔	1.1%
3	+s, +c, +a	(s,c,a)	✔	1.8%
4	+j, +a, +c	(c,a,j)	✔	2.7%
5	+c, +k, +i, -k, -i, +s, +a	(s,c,a)	✔	2.8%
6	+j, +a, +c	(c,a,j)	✔	1.7%
7	+c, +a, +i, -a, +s, -i, +a	(s,c,a)	✔	1.1%
8	+s, +c, +a	(s,c,a)	✔	2.7%
9	+v, +c, -v, +s, +a	(s,c,a)	✔	1.1%
10	+k, +c, +s, -k, +a	(s,c,a)	✔	1.7%

Then we can apply the threshold to the distance calculation using the *indicator function*:

$$\Delta_\theta((\hat{\mathbf{x}}^1, r^1), (\hat{\mathbf{x}}^2, r^2))_i = \mathbf{1}_{[0,\theta_i]}(|(\hat{\mathbf{x}}^1, r^1) - (\hat{\mathbf{x}}^2, r^2)|_i) \tag{7.1}$$

where $\mathbf{1}_{[0,\theta_i]}$ is the indicator function of the given interval.

Evaluation of this vector into a single quantity is possible with numerous ways, Euclidean distance being the simplest. However, as in the discrete version (see (5.1)) we want to account for the worst case in terms of ambiguity. Therefore we propose to use the Chebyshev distance (L_∞ norm, denoted here by $\|\cdot\|_{max}$):

$$\Lambda(\hat{\mathfrak{X}}_{t+1}) = \max_{(\hat{\mathbf{x}}^1, r^1), (\hat{\mathbf{x}}^2, r^2) \in \hat{\mathfrak{X}}_{t+1}} \|\Delta_\theta((\hat{\mathbf{x}}^1, r^1), (\hat{\mathbf{x}}^2, r^2))\|_{max} \tag{7.2}$$

Similarly to the previous version of the continuous ambiguity functional, defined in (6.2), this version shares the properties defined in (2), if we interpret an abstract state to be the more ambiguous the further apart its outcomes are. Unambiguous abstract state will yield only one element in the $\hat{\mathfrak{X}}_{t+1}$ set, and thus will yield 0, the lowest possible value of Λ.

Evaluation of this idea is presented for the Cart-Pole Swing-Up task, for both versions of the reinforcement function. The results are presented in Tab. 7.9 and the Tab. 7.10.

Table 7.5: Results of the approximate algorithm on the 10-dimensional Cart-Pole Swing-Up with a complex reinforcement function. Parameters were set for the following values: GroupCount $= 50$, NeighbourCount $= 50$, $\rho = 0.2$, $\kappa = 0.3$, MaxFailureCount $= 5$, $N_{max} = 200000$.

Run #	Decision sequence	Result	Correctness	Coverage
1	$+s,+i,+k,+l,+h,+x,-i$	(x,s,h,k,l)	✔	2.0%
2	$+s,+c,+h,+l,+x$	(x,s,c,h,l)	✔	2.5%
3	$+k,+s,+h,+x,+a$	(x,s,a,h,k)	✔	4.7%
4	$+k,+s,+i,-k,+l,+x,+h,-i,-s,+s,+c$	(x,s,c,h,l)	✔	7.0%
5	$+k,+s,+j,-k,+x,+h,+l$	(x,s,h,j,l)	✔	7.9%
6	$+s,+j,+i,-s,-j,+k,+l,+h,-i,+x,+j$	(x,h,j,k,l)	✔	4.5%
7	$+s,+j,+k,+h,+l,-s,+x$	(x,h,j,k,l)	✔	1.3%
8	$+k,+s,+j,-k,+x,+h,+l$	(x,s,h,j,l)	✔	7.9%
9	$+s,+c,+h,+x,+a$	(x,s,c,a,h)	✔	2.5%
10	$+k,+s,+h,-k,+l,+x,+j$	(x,s,h,j,l)	✔	4.7%

Table 7.6: Results of the approximate algorithm on the 20-dimensional simple Mountain Car problem. Parameters were set for the following values: GroupCount $= 10$, NeighbourCount $= 100$, $\rho = 3$, $\kappa = 3$, MaxFailureCount $= 20$, $N_{max} = 1000$.

Run #	Decision sequence	Result	Correctness	Coverage
1	$+F_{16},+F_{17}$	(F_{16},F_{17})	✔	0.01%
2	$+F_{7},+F_{20}$	(F_{7},F_{20})	✔	0.01%
3	$+F_{20},+F_{16}$	(F_{20},F_{16})	✔	0.01%
4	$+F_{20},+F_{17}$	(F_{17},F_{20})	✔	0.01%
5	$+F_{16},+F_{8}$	(F_{8},F_{16})	✔	0.01%
6	$+F_{16},+F_{7}$	(F_{7},F_{16})	✔	0.01%
7	$+F_{15},+F_{20}$	(F_{15},F_{20})	✔	0.01%
8	$+F_{16},+F_{15}$	(F_{15},F_{16})	✔	0.01%
9	$+F_{14},+F_{17}$	(F_{14},F_{17})	✔	0.01%
10	$+F_{20},+F_{7}$	(F_{7},F_{20})	✔	0.01%

Table 7.7: Results of the approximate algorithm on the 20-dimensional stochastic Mountain Car problem. Parameters were set for the following values: GroupCount $= 10$, NeighbourCount $= 100$, $\rho = 3$, $\kappa = 3$, MaxFailureCount $= 30$, $N_{max} = 1000$.

Run #	Decision sequence	Result	Correctness	Coverage
1	$+F_{15}, +F_{16}$	(F_{15}, F_{16})	✔	0.01%
2	$+F_{17}, +F_{16}$	(F_{16}, F_{17})	✔	0.01%
3	$+F_{15}, +F_{16}$	(F_{15}, F_{16})	✔	0.01%
4	$+F_{15}, +F_{14}$	(F_{14}, F_{15})	✔	0.01%
5	$+F_{15}, +F_{16}$	(F_{15}, F_{16})	✔	0.01%
6	$+F_{17}, +F_{20}$	(F_{17}, F_{20})	✔	0.01%
7	$+F_{15}, +F_{14}$	(F_{14}, F_{15})	✔	0.01%
8	$+F_{15}, +F_{16}$	(F_{15}, F_{16})	✔	0.01%
9	$+F_{16}, +F_{15}$	(F_{15}, F_{16})	✔	0.01%
10	$+F_{16}, +F_{15}$	(F_{15}, F_{16})	✔	0.01%

Table 7.8: Results of the approximate algorithm on the 10-dimensional Pinball problem. Parameters were set for the following values: GroupCount $= 10$, NeighbourCount $= 100$, $\rho = 3$, $\kappa = 4.5$, MaxFailureCount $= 10$, $N_{max} = 1000$.

Run #	Decision sequence	Result	Correctness	Coverage
1	$+y, +x, +v_y, +F_9$	(x, y, v_y, F_9)	✔	2.5%
2	$+x, +y, +F_{10}, +v_x$	(x, y, v_x, F_{10})	✔	3.9%
3	$+y, +F_5, +F_{10}, +x, -F_5, +v_x$	(x, y, v_x, F_{10})	✔	4.2%
4	$+F_5, +x, +F_6, -F_5, +y, -F_6, +v_x, -v_x, +v_y, -v_y, +F_{10}, +v_x$	(x, y, v_x, F_{10})	✔	4.3%
5	$+F_5, +y, +v_x, +x, -y, +y, -F_5, -v_x, +F_8, -F_8, +v_y, +F_8$	(x, y, v_x, F_8)	✔	3.6%
6	$+F_6, +v_y, +x, +v_x$	(x, v_x, v_y, F_6)	✔	6.2%
7	$+x, +y, F_8, +v_x$	(x, y, v_x, F_8)	✔	6.8%
8	$+F_6, +v_x, +F_{10}, +x$	(x, v_x, F_6, F_{10})	✔	4.5%
9	$+F_6, +x, +F_9, +F_{10}, -F_9, +v_x, -x, +x$	(x, v_x, F_6, F_{10})	✔	3.2%
10	$+F_5, +x, +y, +v_y, -F_5, +F_5, -v_y, +v_y$	(x, y, v_y, F_5)	✔	2.2%

Table 7.9: Results for the Cart-Pole Swing-Up problem with simple reinforcement function, with distance-based ambiguity calculation. Parameter were set to the following values: GroupCount $= 5$, NeighbourCount $= 3$, $N_{max} = 1000$.

Run #	Decision sequence	Result	Ambiguity	Correctness	Coverage
1	+j, +a, +c	c, a, j	0	✔	25%
2	+s, +j, +a	s, a, j	0	✔	29%
3	+s, +c, +a	s, c, a	0	✔	26%
4	+j, +c, +a	c, a, j	0	✔	29%
5	+a, +j, +s	s, a, j	0	✔	27%
6	+s, +j, +a	s, a, j	0	✔	27%
7	+c, +j, +v, +a, -v	c, a, j	0	✔	25%
8	+c, +j, +a	c, a, j	0	✔	26%
9	+s, +a, +c	s, c, a	0	✔	28%
10	+c, +a, +j	c, a, j	0	✔	25%

Table 7.10: Results for the Cart-Pole Swing-Up problem a complex reinforcement function, with distance-based ambiguity calculation. Parameter were set to the following values: GroupCount $= 5$, NeighbourCount $= 3$, $N_{max} = 1000$.

Run #	Decision sequence	Result	Ambiguity	Correctness	Coverage
1	+s, +j, +a, -j, +h, +j, +v	v, s, a, h, j	0.00783	✔	33%
2	+a, +s, +h, +v, +j	v, s, a, h, j	0.00399	✔	26%
3	+j, +l, +s, -j, +x, +h, +j	x, s, h, j, l	0.00000	✔	25%
4	+c, +j, +a, -c, +x, +h, +s	x, s, a, h, j	0.01009	✔	25%
5	+a, +j, +s, +x, +v	x, v, s, a, j	0.01618	✔	33%
6	+j, +c, +l, +x, +v	x, v, c, j, l	0.00398	✔	31%
7	+h, +v, +j, +l, +c	v, c, h, j, l	0.00820	✔	26%
8	+h, +v, +s, +j, +l	v, s, h, j, l	0.00554	✔	25%
9	+c, +a, +j, +x, +h	x, c, a, h, j	0.00000	✔	25%
10	+l, +s, +h, +j, -s, +v, +c	v, c, h, j, l	0.00414	✔	27%

Chapter 8

Conclusion

8.1 Summary of thesis

In this work we analyzed the state abstraction by variable selection problem in the context of Reinforcement Learning (RL). After the introduction in Chapt. 1, in Chapt. 2 we presented testing environments, used throughout this thesis in examples and experiments, namely the Discrete Labyrinth, Coffee Task, Continuous Labyrinth, Cart-Pole Swing-Up (with simple and complex reinforcement functions), Mountain Car (simple and stochastic versions) and Pinball. In Chapt. 3 we put forward basic notions and definitions, on which the main idea of this work is founded. In Sec. 3.7 we summarized the existing work on state abstraction in RL.

In Sec. 4.3 of Chapt. 4 we showed proof of the first sub-thesis of this work formulated in Chapt. 1, namely that state abstraction by variable selection problem is NP-hard and inapproximable. This fact legitimizes devising a heuristic algorithm.

In the following Sec. 4.4 we combined the results of the analysis of the existing work on *bisimulation* in RL (Sec. 4.2) with the introduced notion of *ambiguity*, inspired by behavioral psychology, to propose the *ambiguity functional* as a way of assessing and comparing quality of state abstraction functions. Next, in Sec. 4.5, two general incremental paradigms, namely the *bottom-up* approach and the *top-down* approach were compared theoretically to show that without having any knowledge about the problem being solved the *bottom-up* approach is better on average. We argue that this result advocates focusing on the *bottom-up* approach in our algorithm, even though occasionally it makes *top-down* steps.

After analyzing our algorithm in the context of discrete environments in Chapt. 5, we extended the proposed framework to the continuous case in Chapt. 6. Estimating the values

of the ambiguity functional in the continuous case involves solving a new type of the *nearest neighbour problem*, which we call the *multispace nearest neighbour* problem. In Sec. 6.3 we analyzed this problem and presented a solution, which improves our state abstraction algorithm's computational complexity. This led to the formulation of our algorithm, namely the NADA (Neighborhood Ambiguity Driven Abstraction) algorithm in Sec. 6.8.

Successive examples and experiments presented in subsequent sections of Chapters 4, 5 and 6 support the second sub-thesis introduced in Chapt. 1, which claims that the introduced ambiguity functional correctly reflects validity and quality of the abstraction function. These results were obtained in the context of the *simulator assumption* (Sec. 3.2), where data samples can be generated on demand, using the knowledge about the internals of the considered testing environment.

Knowing that our algorithm works when given all required data, we examined the proposed approach in a practical scenario, when the algorithm receives only a random set of trajectories. Practical issues with using the ambiguity functional in this scenario were analyzed and a set of heuristics to address them were proposed. The evaluation of our algorithm without the simulator assumption in Chapt. 7 shows that the algorithm works reliably for all of the testing environments. This advocates the last sub-thesis, namely that the proposed approach is feasible for solving variable selection tasks in RL. Experimental results demonstrate improvement, regarding other works in state abstraction in RL, in terms of number of states (in case of discrete domains) and number of measurables (in case of continuous domains) in the context of state-of-the-art research regarding the state abstraction problem. The comparison with other works was presented in Sec. 3.7. Experimental results for the continuous environments are the first known the the author.

The three sub-theses formulated in Chapt. 1, through the presented derivations and experimental results, support the main thesis of this work, namely that our ambiguity-based, bottom-up approach is a valid solution of the Reinforcement Learning variable selection problem.

8.2 Critical overview

The main problem of the presented algorithm stems from the fact that to estimate the inverse images of abstraction functions the algorithm needs to wait for reoccurring events (values of the vector of measurables). For abstractions with more than 3 measurables, in case of continuous

tasks, this can take a long time (e.g. 20 hours in case of the Cart-Pole Swing-Up with complex reinforcement function, Sec. 7.3.1). To make the running time shorter, we estimate the value of the ambiguity functional with incomplete samples groups, as presented in Chapt. 7. The results for such an approximate solution look promising, however they were obtained only for problems with up to twenty measurables, with five-dimensional solution at most. On the other hand, interestingly, this is somehow related to the psychological mechanisms that inspired this work. Some experiments related to humans' ability to make classification "judgments" based on "multidimensional stimuli" show that the accuracy of such judgments significantly drops even for two dimensions [81], [26]. Such psychological experiments were carried out for no more than six dimensions. While this is not directly related to the *stimulus discrimination* problem [102], the inability to make accurate classification assessments for more than two stimuli suggests that deciding which of these stimuli is more important than others would also be hard.

The conclusion of the reasoning in Sec. 4.5 in Chapt. 4 was that the *bottom-up* approach on average is better, and thus the designed state abstraction approach should work according to this paradigm. However, further investigations (namely, Experiment 5.4.1 in Sec. 5.4 in Chapt. 5) have shown that to reach the optimal solution, sometimes some measurables need to be removed from the abstraction and removing measurables from the solution is a *top-down* step. This makes the proposed solution, in fact, a mixed paradigm solution, i.e. one that is partially *bottom-up* and partially *top-down*. This in turn puts to question whether it would not be better to investigate mixed paradigms further, not attaching too much importance on the superiority of the *bottom-up* paradigm.

Another issue with the presented analysis is that the parameters of our algorithm were either known from using transitions generated on demand (the *simulator assumption*, Sec. 3.2) or found using the knowledge of what is the correct solution (e.g. smallest ρ parameters values that have yielded results known to be correct). In a real-work scenario we do not know the solution beforehand, and even though for some problems numerical simulators exist [120], this is not always the case. Thus, the real-world scenario would involve guessing the values of the parameters, finding an abstraction with our algorithm, and then running a RL algorithm to determine quality of a learned policy. This would significantly prolong computation time, which now can take even 20 hours, for complex environments (see results for Cart-Pole Swing-Up with complex reinforcement function in Sec. 7.3.1).

8.3 Future work

The most important drawback of the algorithm - waiting for reoccurring events, is the first opportunity for the follow-up research. Reliable estimator for the ambiguity functional that could work for a given set of independent trajectories, without the need to wait for some particular observations to occur, would greatly improve performance of the algorithm.

Another interesting direction for the future research is investigating an application of this algorithm for calculating other types of abstractions, most notably the ϕ_{Q^*} type. Abstractions of this type aggregate more measurements than ϕ_{model} abstractions, thus simplify the task more, potentially improving RL algorithm performance significantly. This type of abstraction is also more popular in the RL community (as can be seen in related work lists in Sec. 3.7). The main idea behind this approach is to treat Bellman's equation as a model that has to be preserved by a ϕ_{model} abstraction, with values of the Q-function estimator treated as states. This way, an abstraction algorithm working to determine a ϕ_{model} abstraction for an "imaginary" plant would in fact discover the ϕ_{Q^*} abstraction for the underlying domain.

Additionally, the presented approach should be easy to extend for selecting sufficient and non-redundant action vector's elements for the continuous action space case ($\mathcal{U} \in \mathcal{R}^p$ for some p).

Determining the values of the algorithm's parameters also requires further investigation. For example, the algorithm could be automatically run many times, starting with low accuracy (large values of parameter ρ) and increasing it if needed, similarly to multi-resolution approaches like the one presented in [87]. The value of parameter κ could also be determined automatically. Sufficiently large value of this parameter always makes the algorithm to yield a *null abstraction*. Decreasing value of κ automatically, for subsequent runs of the same experiment would lead to a non-null abstraction eventually.

Appendix A

Technical details

The solution was mainly implemented on Microsoft Windows 7 system, within the .NET Framework 4.5, using the C# language and the *dotRL* platform [93]. The sample data was stored simultaneously using a MySQL 5.6 database [118] in MyISAM tables and in RAM. The database was used for solving the Nearest Neighbour problem using the approach presented in Sec. 6.3. For performance reasons, the results from the database were retrieved only as sample identifiers and appropriate samples were read from the redundant RAM storage. Smaller experiments and examples were calculated/generated using scripts written in Python 2.7 [98]. Most machines used for calculations where equipped with Intel® Core™ i5 Processors (with two examples running on Intel® Core™ i7), at least 4GB of RAM memory (with two examples equipped with 16GB) and at least 0.5GB of additional disk space for sample database storage.

The main idea of this work is represented by the `AmbiguityCalculator` interface with three implementations, presented in Fig. A.1, namely: `DiscreteAmbiguityCalculator` - implementing the method of calculating the discrete version of the ambiguity functional, according to (5.1), `NeighbourhoodContinuousAmbiguityCalculator` - implementing the method of calculating the continuous version of the ambiguity functional, according to (6.2) and `DistanceContinuousAmbiguityCalculator` - implementing the alternative method of calculating the continuous version of the ambiguity functional, according to (7.2). They directly relate to the three presented ways of calculating the *ambiguity functional* in Chapters 5.1, 6.1 and 7 resp. Values calculated by these classes are in turn used by the `AbstractionEvaluator`. It uses two classes: `ExtensionCalculator` and `ReductionCalculator`, which execute extension and reduction steps resp. as described in Sec. 6.8. The `ReductionCalculator` additionally uses the `DimensionGuard` class, which represents the *extension memory* concept and is

responsible for preventing cycles in abstraction paths. This is done as described in Sec. 5.5.

Figure A.3 presents `VectorTransitionService` class, which orchestrates all operations related to observed transitions and reinforcements, like storage and calculating the resolution parameter ρ presented in Sec. 5.4 and threshold parameters $\theta^{(x)}$ and $\theta^{(r)}$ presented in Sec. 6.4 using classes `EpsilonEstimator` and `LogThresholdEstimator` resp. It realizes the top-level domain service class, according to the principles presented by Evans in [28].

The main notions are modelled by entities presented in Fig. A.2. The `VectorTransition` and `MarkovTransition` classes represent a pair of vectors $\mathbf{x}_t$, $\mathbf{x}_{t+1}$ and scalars u_t and r_{t+1} in a general, and RL context resp. Ambiguity samples and groups of samples are represented by `AmbiguitySample` and `AmbiguitySampleMatch` classes resp. Classes `Ambiguity`, `AmbiguityRatio` represent the core notions of this work: the *ambiguity* being the value of the *ambiguity functional*.

The Fig. A.4 presents additional helper classes that implement sets of measurables that constitute an abstraction function, and *finer*/*coarser* relations between abstractions presented in (3.6) and described in Sec. 3.3. Classes `ActionSubspace`, `CoreSubspace` and `ReinforcementSubspace` are aliases that give particular meaning for some sets of measurables - an *action subspace* is a group of indices in the transition vector that hold the action chosen by the control policy, the *core subspace* is the set of measurables in the current abstraction function and the *reinforcement subspace* is the index in the transition vector that contains reinforcement value. In the context of this work *action subspace* and *reinforcement subspace* are always 1-dimensional. The `AmbiguitySpace` class represents a pair of index sets:

- *source index set* is a set of indices from the transition vector used to group samples together (in this work it contains indices from the *action subspace* and *core subspace* - because state and action uniquely define an event in the system)

- *target index set* is a set of indices from the transition vector that represnt an outcome of RL policy step, and are used to calculate the ambiguity functional (in this work it contains indices from the *core subspace* and the *reinforcement subspace* - because the next state and received reinforcement define an outcome of a policy's step)

There are also numerous data structures for handling the storage of samples. This is due to the necessity of optimizing the nearest neighbour queries for different projections, as described in Sec. 6.3.

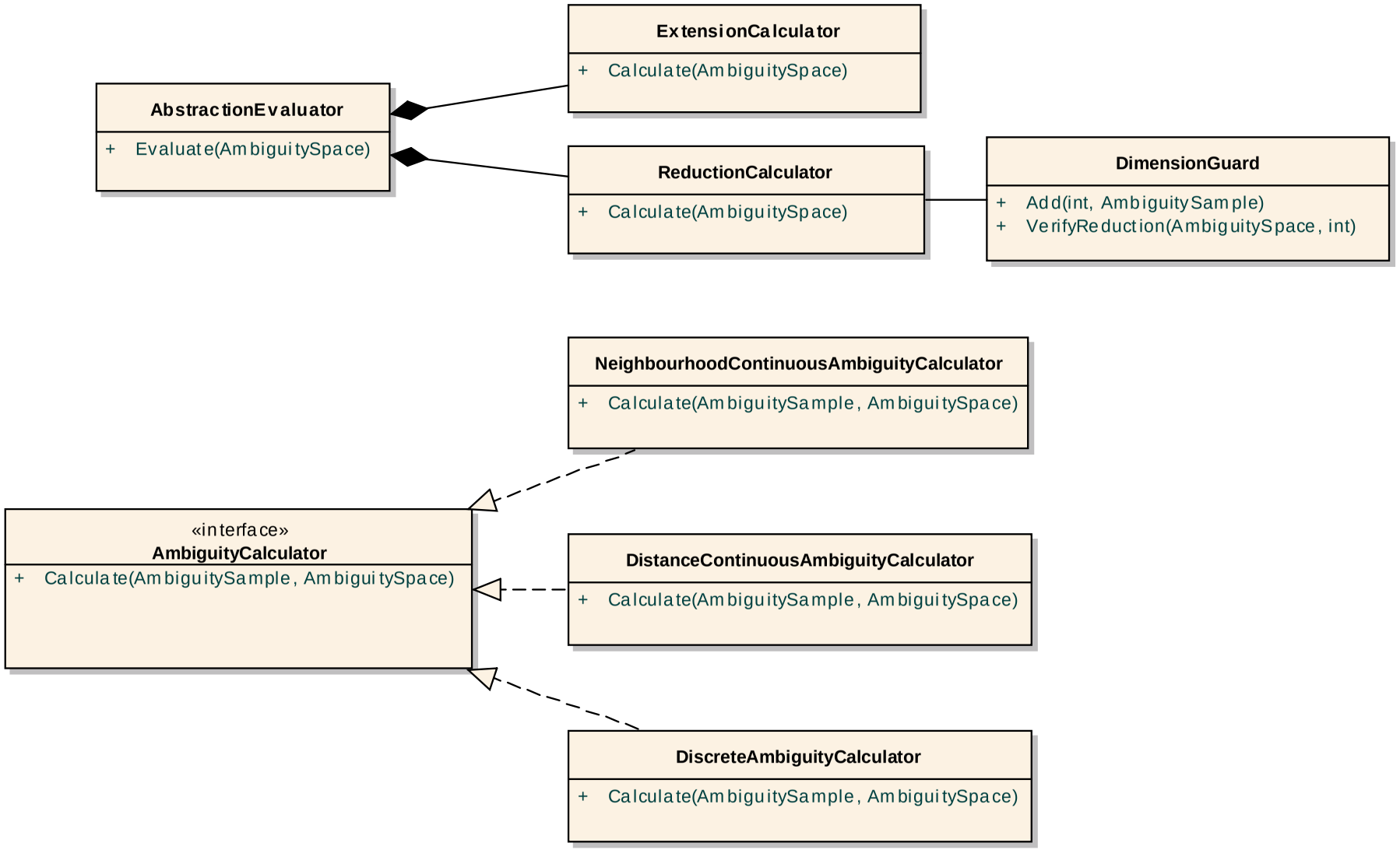

Figure A.1: Abstraction algorithm related service classes. `AbstractionEvaluator` class determines potential measurables to include or remove from current abstraction. The `DimensionGuard` class is used to prevent abstraction path cycles. Various implementations of the `AmbiguityCalculator` interface implement calculation of three versions of the ambiguity functional presented in this work: discrete, continuous and continuous with distance.

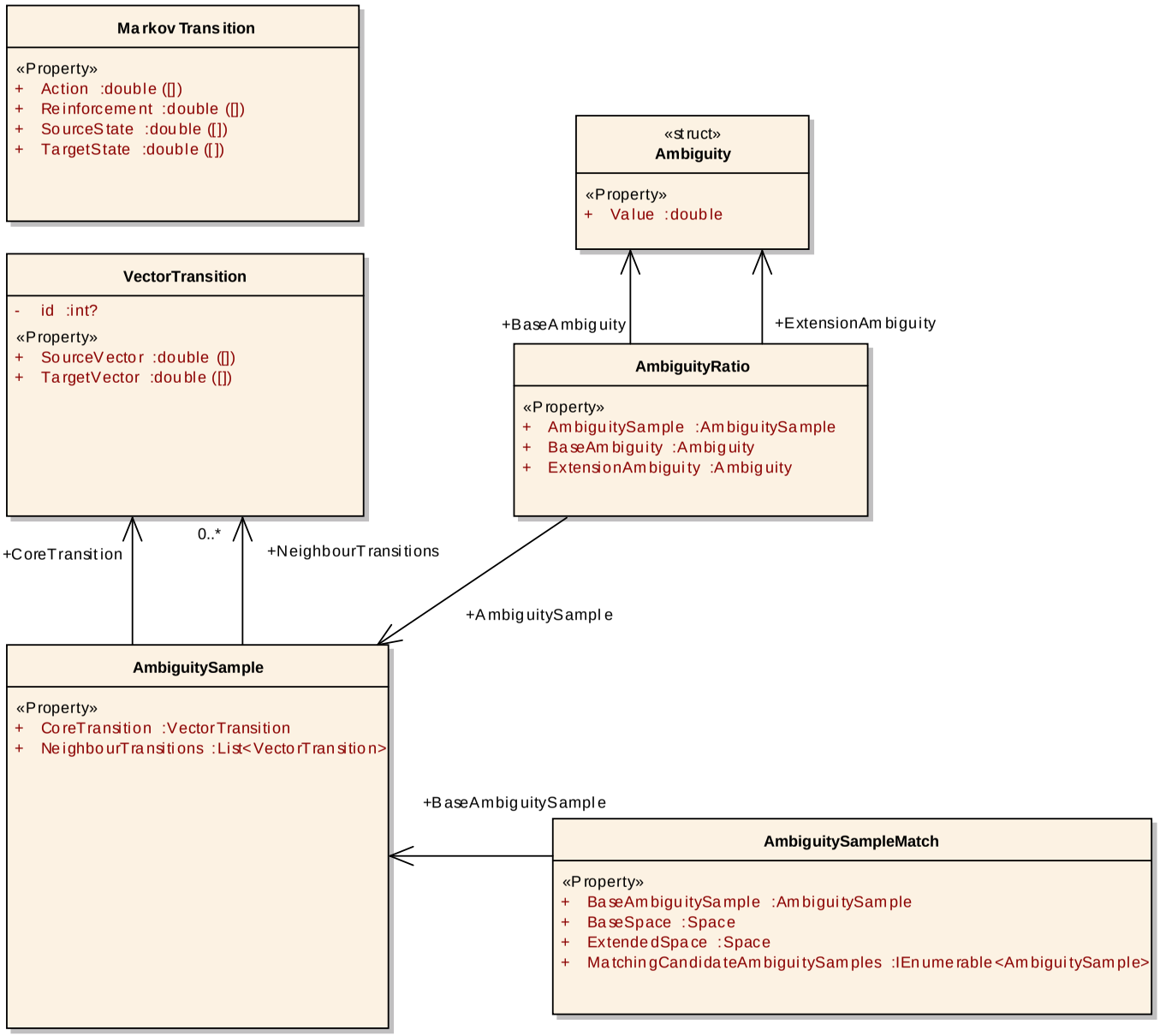

Figure A.2: Abstraction algorithm and ambiguity related entities. `MarkovTransition` and `VectorTransition` represent samples in MDP restricted sense and in general vector-vector transition sense, resp. `AmbiguitySample` and `AmbiguitySampleMatch` represent groups introduced in Secs. 5.4 and 6.4, resp.

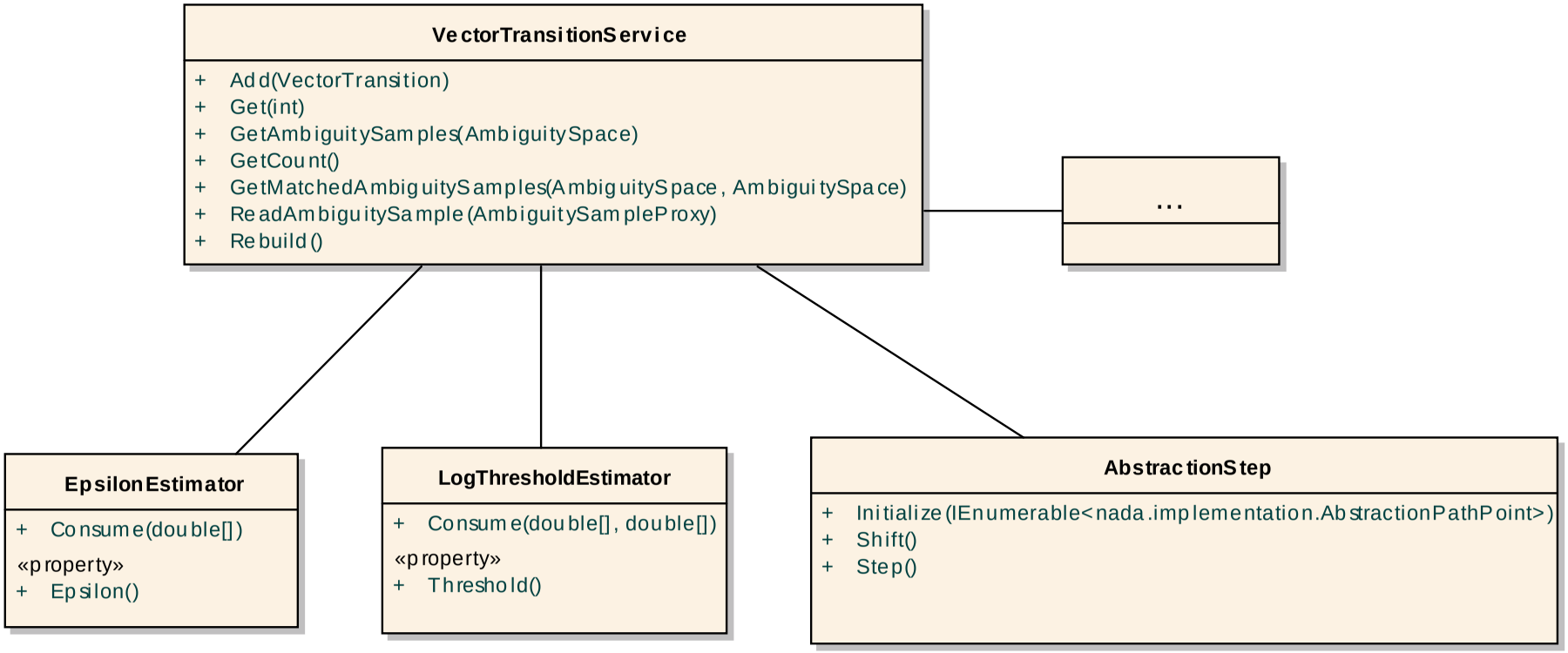

Figure A.3: Main class that processes samples from the observed environment with some utility classes: `EpsilonEstimator` and `LogThresholdEstimator` that calculate parameters defined in (5.2) and (6.4), and `AbstractionStep` that attempts to include or remove variables from current abstraction and reports if the algorithm has finished.

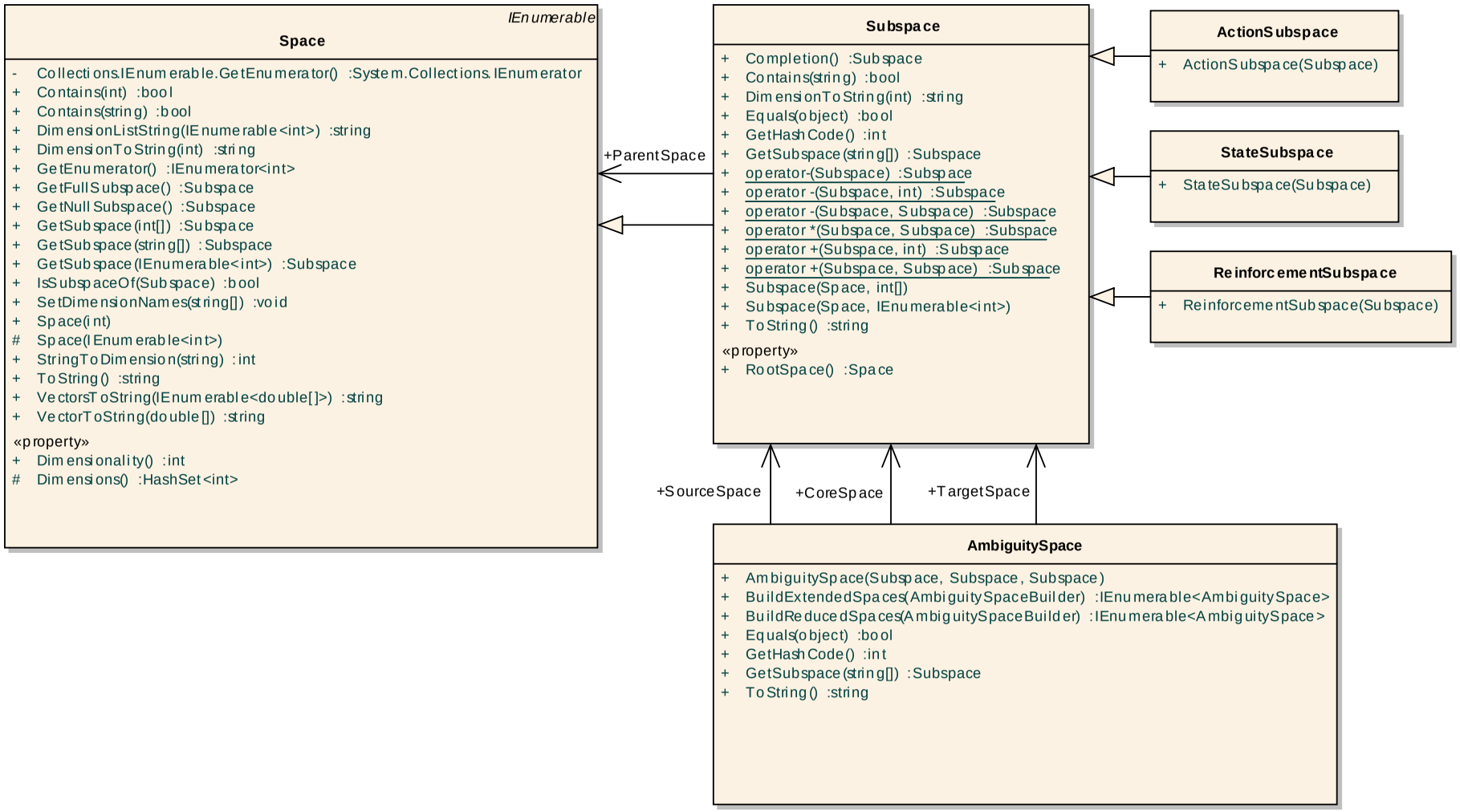

Figure A.4: State abstraction related classes. `Space` represents general concept of a space in factored form. `Subspace` models a projection of a given space, obtained by ignoring some of its coordinates. `AmbiguitySpace` represents three subspaces: *source*, *core*, and *target*. The source subspace represents variables used to aggregate vectors at time instant t, the target subspace represents variables used to evaluated outcomes of the transition and reinforcement functions at time instant $t+1$. The core subpsace is a common part of these subspaces. Agent's actions are variables that are always present in the source subspace, but never present in the target subspace. Reinforcement values are always included in the target space but never in the source space. The core space contains all other variables, that are result of the state abstraction algorithm.

Author's publications

[A1] Bartosz Papis. "Evaluation of colour recognition algorithms with a palette designed for applications in aiding people with visual impairment". In: *International Journal of Image, Graphics and Signal Processing (IJIGSP)* 6.12 (2014), pp. 1–7. ISSN: 2074-9074.

[A2] Bartosz Papis and Andrzej Pacut. "Multispace, Dynamic, Fixed-Radius, All Nearest Neighbours Problem". In: *16th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*. 2014, pp. 244–250.

[A3] Bartosz Papis and Andrzej Pacut. "Neighbourhood approach to bisimulation in state abstraction for quantized domains". In: *19th International Conference On Methods and Models in Automation and Robotics (MMAR)*. 2014, pp. 566–571.

[A4] Bartosz Papis and Paweł Wawrzyński. "dotRL: A platform for rapid Reinforcement Learning methods development and validation". In: *Federated Conference on Computer Science and Information Systems (FedCSIS)*. 2013, pp. 129–136.

[A5] Paweł Wawrzyński and Bartosz Papis. "Fixed point method for autonomous on-line neural network training". In: *Neurocomputing* 74.17 (2011), pp. 2893–2905.

Bibliography

[1] J. S. Albus. “A New Approach to Manipulator Control: the Cerebellar Model Articulation Controller (CMAC)”. In: *Journal of Dynamic Systems, Measurement, and Control* 97 (1975), pp. 220–227.

[2] Saul Amarel. “On Representations of Problems of Reasoning about Actions”. In: *Machine Intelligence 3*. Ed. by Donald Michie. American Elsevier Publisher, 1968, pp. 131–171.

[3] David Andre and Stuart J. Russell. “State Abstraction for Programmable Reinforcement Learning Agents”. In: *Proceedings of the Eighteenth National Conference on Artificial Intelligence*. AAAI Press, 2002, pp. 119–125.

[4] Andrew G. Barto and Sridhar Mahadevan. “Recent Advances in Hierarchical Reinforcement Learning”. In: *Discrete Event Dynamic Systems* 13.1-2 (Jan. 2003), pp. 41–77. ISSN: 0924-6703.

[5] Andrew G. Barto, Richard S. Sutton, and Charles W. Anderson. “Neuronlike adaptive elements that can solve difficult learning control problems”. In: *Artificial neural networks*. Piscataway, NJ, USA: IEEE Press, 1990, pp. 81–93. ISBN: 0-8186-2015-3.

[6] Richard Bellman. “A Markovian Decision Process”. In: *Indiana University Mathematics Journal* 6 (4 1957), pp. 679–684. ISSN: 0022-2518.

[7] Richard Ernest Bellman. *Dynamic Programming*. Dover Publications, Incorporated, 2003. ISBN: 0486428095.

[8] Sandjai Bhulai. *Markov Decision Processes: The Control of High-dimensional Systems*. Universal Press, Netherlands, 2002. ISBN: 9789090157917.

[9] Craig Boutilier, Richard Dearden, and Moisés Goldszmidt. “Exploiting structure in policy construction”. In: *IJCAI-95, pp.1104-1111*. 1995.

[10] Ronen I. Brafman, Moshe Tennenholtz, and Pack Kaelbling. *R-MAX - A General Polynomial Time Algorithm for Near-Optimal Reinforcement Learning*. 2001.

[11] J. D. Bransford, A. L. Brown, and R. R. Cocking. *How People Learn: Brain, Mind, Experience, and School: Expanded Edition*. National Academies Press, 2000. ISBN: 0309070368.

[12] Edgar Chávez et al. "Searching in Metric Spaces". In: *ACM Comput. Surv.* 33.3 (Sept. 2001), pp. 273–321. ISSN: 0360-0300. DOI: 10.1145/502807.502808.

[13] D. Christiansen, C.K. Alexander, and R.K. Jurgen. *Standard Handbook of Electronic Engineering, 5th Edition*. McGraw-Hill standard handbooks. McGraw-Hill, 2005. ISBN: 9780071500845.

[14] Paolo Ciaccia, Marco Patella, and Pavel Zezula. "M-tree: An Efficient Access Method for Similarity Search in Metric Spaces". In: *Proceedings of the 23rd International Conference on Very Large Data Bases*. VLDB '97. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1997, pp. 426–435. ISBN: 1-55860-470-7.

[15] Luis C. Cobo et al. "Abstraction from demonstration for efficient reinforcement learning in high-dimensional domains". In: *Artificial Intelligence* 216.0 (2014), pp. 103 –128. ISSN: 0004-3702.

[16] Gheorghe Comanici, Prakash Panangaden, and Doina Precup. "On-the-Fly Algorithms for Bisimulation Metrics". In: *Proceedings of the 2012 Ninth International Conference on Quantitative Evaluation of Systems*. QEST '12. IEEE Computer Society, 2012, pp. 94–103. ISBN: 978-0-7695-4781-7.

[17] Gheorghe Comanici and Doina Precup. "Basis Function Discovery Using Spectral Clustering and Bisimulation Metrics". In: *Adaptive and Learning Agents*. Vol. 7113. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012, pp. 85–99. ISBN: 978-3-642-28498-4.

[18] Pat Croskerry, Karen S. Cosby, and Stephen M. Schenkel. *Patient Safety in Emergency Medicine*. Lippincott Williams & Wilkins, 2008. ISBN: 0781777275.

[19] Sanjoy Dasgupta and Yoav Freund. "Random Projection Trees and Low Dimensional Manifolds". In: *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*. STOC '08. Victoria, British Columbia, and Canada: ACM, 2008, pp. 537–546. ISBN: 978-1-60558-047-0.

[20] Thomas Dean, Robert Givan, and Sonia Leach. “Model Reduction Techniques for Computing Approximately Optimal Solutions for Markov Decision Processes”. In: *Thirteenth Conference on Uncertainty in Artificial Intelligence*. Morgan Kaufmann, 1997, pp. 124–131.

[21] Thomas Degris and Olivier Sigaud. “Learning the structure of factored Markov decision processes in reinforcement learning problems”. In: *Proceedings of the 23rd international conference on Machine learning*. 2006, pp. 257–264.

[22] Thomas G. Dietterich. “State Abstraction in MAXQ Hierarchical Reinforcement Learning”. In: *Advances in Neural Information Processing Systems 12*. MIT Press, 2000, pp. 994–1000.

[23] Carlos Diuk, Lihong Li, and Bethany R. Leffler. “The adaptive k-meteorologists problem and its application to structure learning and feature selection in reinforcement learning”. In: *Proceedings of the 26th Annual International Conference on Machine Learning*. ICML ’09. 2009, pp. 249–256. ISBN: 978-1-60558-516-1.

[24] David L. Donoho. *Aide-Memoire. High-Dimensional Data Analysis: The Curses and Blessings of Dimensionality*. 2000.

[25] Grzegorz Dudek. “Tournament Searching Method to Feature Selection Problem”. English. In: *Artifical Intelligence and Soft Computing*. Ed. by Leszek Rutkowski et al. Vol. 6114. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2010, pp. 437–444. ISBN: 978-3-642-13231-5.

[26] Howard Egeth and Robert Pachella. “Multidimensional stimulus identification”. English. In: *Perception & Psychophysics* 5.6 (1969), pp. 341–346. ISSN: 0031-5117.

[27] Martin Ester et al. “A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise”. In: *Second International Conference on Knowledge Discovery and Data Mining*. Ed. by Evangelos Simoudis, Jiawei Han, and Usama Fayyad. Portland, Oregon: AAAI Press, 1996, pp. 226–231.

[28] Eric Evans. *Domain-driven design: tackling complexity in the heart of software*. Addison-Wesley, 2003. ISBN: 0-321-12521-5.

[29] Uriel Feige. “A Threshold of Ln N for Approximating Set Cover”. In: *Journal of the ACM* 45.4 (July 1998), pp. 634–652. ISSN: 0004-5411.

[30] Norm Ferns, Prakash Panangaden, and Doina Precup. “Bisimulation Metrics for Continuous Markov Decision Processes”. In: *SIAM J. Comput.* 40.6 (2011), pp. 1662–1714.

[31] Norm Ferns, Prakash Panangaden, and Doina Precup. “Metrics for finite Markov decision processes”. In: *Proceedings of the 20th conference on Uncertainty in artificial intelligence*. UAI ’04. Banff, Canada: AUAI Press, 2004, pp. 162–169. ISBN: 0-9749039-0-6.

[32] Razvan V Florian. “Correct equations for the dynamics of the cart-pole system”. In: *Center for Cognitive and Neural Studies (Coneural), Romania* (2007).

[33] Imola Fodor. *A Survey of Dimension Reduction Techniques*. 2002.

[34] T. Freijy, B. Mullan, and L. Sharpe. “Food-related attentional bias. Word versus pictorial stimuli and the importance of stimuli calorific value in the dot probe task”. In: *Appetite* 83 (2014), pp. 202–208.

[35] Bernard Friedland. *Control System Design: An Introduction to State-Space Methods*. Dover Books on Electrical Engineering. Dover Publications, 2012. ISBN: 9780486135113.

[36] Jerome H. Friedman, Jon Louis Bentley, and Raphael Ari Finkel. “An Algorithm for Finding Best Matches in Logarithmic Expected Time”. In: *ACM Trans. Math. Softw.* 3.3 (Sept. 1977), pp. 209–226. ISSN: 0098-3500.

[37] J.H. Friedman, F. Baskett, and L.J. Shustek. “An Algorithm for Finding Nearest Neighbors”. In: *IEEE Transactions on Computers* C-24.10 (1975), pp. 1000–1006. ISSN: 0018-9340.

[38] Bernd Fritzke. “A Growing Neural Gas Network Learns Topologies”. In: *Advances in Neural Information Processing Systems 7*. MIT Press, 1995, pp. 625–632.

[39] Adrian Furnham and Joseph Marks. “Tolerance of Ambiguity: A Review of the Recent Literature”. In: *Psychology* 4.9 (2013), pp. 717–728.

[40] Robert Givan, Thomas Dean, and Matthew Greig. “Equivalence notions and model minimization in Markov decision processes”. In: *Artif. Intell.* 147.1-2 (July 2003), pp. 163–223. ISSN: 0004-3702.

[41] Ebru Aydin Gol et al. “Finite bisimulations for switched linear systems”. In: *Proceedings of the 51th IEEE Conference on Decision and Control (CDC)*. 2012, pp. 7632–7637.

[42] S. Gugercin and A. C. Antoulas. “A survey of model reduction by balanced truncation and some new results”. In: *International Journal of Control* 77.8 (2004), 748–766.

[43] Isabelle Guyon and André Elisseeff. “An introduction to variable and feature selection”. In: *J. Mach. Learn. Res.* 3 (Mar. 2003), pp. 1157–1182. ISSN: 1532-4435.

[44] M.A. Hall and G. Holmes. “Benchmarking attribute selection techniques for discrete class data mining”. In: *Knowledge and Data Engineering, IEEE Transactions on* 15.6 (2003), pp. 1437–1447. ISSN: 1041-4347.

[45] H. van Hasselt and M.A. Wiering. “Reinforcement Learning in Continuous Action Spaces”. In: *IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*. 2007, pp. 272–279.

[46] Bernhard Hengst. “Discovering Hierarchy in Reinforcement Learning with HEXQ”. In: *In Maching Learning: Proceedings of the Nineteenth International Conference on Machine Learning*. Morgan Kaufmann, 2002, pp. 243–250.

[47] Todd Hester and Peter Stone. “Generalized model learning for reinforcement learning in factored domains”. In: *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems - Volume 2*. AAMAS ’09. Budapest, Hungary: International Foundation for Autonomous Agents and Multiagent Systems, 2009, pp. 717–724. ISBN: 978-0-9817381-7-8.

[48] Alain Hiel and Ivan Mervielde. “The Need for Closure and the Spontaneous Use of Complex and Simple Cognitive Structures”. In: *The Journal of Social Psychology* 143.5 (2003), pp. 559–568. ISSN: 0022-4545.

[49] A. Hillion, P. Masson, and C. Roux. “A nonparametric approach to linear feature extraction; application to classification of binary synthetic textures”. In: *9th International Conference on Pattern Recognition*. 1988, 1036–1039 vol.2.

[50] R. F. Holt. “Enhancing speech discrimination through stimulus repetition”. In: *J. Speech Lang. Hear. Res.* 54.5 (2011), pp. 1431–1447.

[51] W. Homenda. *Algorytmy, złożoność obliczeniowa, granice obliczalności*. Lecture Notes - Centrum Studiów Zaawansowanych Politechniki Warszawskiej: Nauki Ścisłe. Centrum Studiów Zaawansowanych Politechniki Warszawskiej, 2009.

[52] R.A. Howard. *Dynamic Programming and Markov Processes*. The M.I.T. Press, 1960.

[53] Zhenhua Huang, Xin Xu, and Lei Zuo. “Reinforcement learning with automatic basis construction based on isometric feature mapping”. In: *Information Sciences* 286.0 (2014), pp. 209 –227. ISSN: 0020-0255.

[54] A.G. Ivakhnenko. “Polynomial Theory of Complex Systems”. In: *Systems, Man and Cybernetics, IEEE Transactions on* SMC-1.4 (1971), pp. 364–378. ISSN: 0018-9472.

[55] C.D. Johnson. *Process Control Instrumentation Technology*. Pearson/Prentice Hall, 2006. ISBN: 9780131194571.

[56] Nicholas K. Jong. “State abstraction discovery from irrelevant state variables”. In: *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*. 2005, pp. 752–757.

[57] Anders Jonsson and Andrew Barto. “Casual Graph Based Decomposition of Factored MDPs”. In: *Journal of Machine Learning Research* 7 (2006), pp. 2259–2301.

[58] Anders Jonsson and Andrew G. Barto. “Automated State Abstraction for Options using the U-Tree Algorithm”. In: *Advances in neural information processing systems*. MIT Press, 2001, pp. 1054–1060.

[59] Chia-Feng Juang. “Combination of online clustering and Q-value based GA for reinforcement fuzzy system design”. In: *Trans. Fuz Sys.* 13.3 (June 2005), pp. 289–302. ISSN: 1063-6706.

[60] Rudolf Kalman. “On the general theory of control systems”. In: *IRE Transactions on Automatic Control* 4.3 (1959), pp. 110–110. ISSN: 0096-199X.

[61] RichardM. Karp. “Reducibility among Combinatorial Problems”. In: *Complexity of Computer Computations*. Ed. by RaymondE. Miller, JamesW. Thatcher, and JeanD. Bohlinger. The IBM Research Symposia Series. Springer US, 1972, pp. 85–103. ISBN: 978-1-4684-2003-6. DOI: 10.1007/978-1-4684-2001-2_9.

[62] H.K. Khalil. *Nonlinear Systems*. Prentice Hall PTR, 2002. ISBN: 9780130673893.

[63] Kenji Kira and Larry A. Rendell. “A practical approach to feature selection”. In: *Proceedings of the ninth international workshop on Machine learning*. ML92. Aberdeen, Scotland, United Kingdom: Morgan Kaufmann Publishers Inc., 1992, pp. 249–256. ISBN: 1-5586-247-X.

[64] Ron Kohavi and George H. John. “Wrappers for feature subset selection”. In: *Artificial Intelligence* 97.1 - 2 (1997), pp. 273 –324. ISSN: 0004-3702.

[65] G.D. Konidaris and A.G. Barto. “Skill Discovery in Continuous Reinforcement Learning Domains using Skill Chaining”. In: *Advances in Neural Information Processing Systems 22*. Ed. by Y. Bengio et al. 2009, pp. 1015–1023.

[66] George Konidaris. *Pinball Domain*. `http://www-all.cs.umass.edu/~gdk/pinball/`. [Online; accessed 30-March-2015]. 2009.

[67] George Konidaris and Andrew Barto. “Autonomous shaping: knowledge transfer in reinforcement learning”. In: *In Proceedings of the 23rd Internation Conference on Machine Learning*. 2006, pp. 489–496.

[68] George Konidaris and Andrew Barto. “Efficient skill learning using abstraction selection”. In: *Proceedings of the 21st International Joint Conference on Artificial Intelligence*. 2009.

[69] Arie W. Kruglanski and Donna M. Webster. “Motivated closing of the mind: "seizing" and "freezing"”. In: *Psychological Review* 103.2 (1996), pp. 263–283.

[70] Jan van Leeuwen. The MIT Press, 1994. ISBN: 0262720140.

[71] Lihong Li and Michael L. Littman. “Lazy Approximation for Solving Continuous Finite-horizon MDPs”. In: *Proceedings of the 20th National Conference on Artificial Intelligence - Volume 3*. AAAI’05. Pittsburgh, Pennsylvania: AAAI Press, 2005, pp. 1175–1180. ISBN: 1-57735-236-x.

[72] Lihong Li, Thomas J. Walsh, and Michael L. Littman. “Towards a Unified Theory of State Abstraction for MDPs”. In: *Proceedings of the Ninth International Symposium on Artificial Intelligence and Mathematics*. 2006, pp. 531–539.

[73] Stephen Lin and Robert Wright. “Evolutionary Tile Coding: An Automated State Abstraction Algorithm for Reinforcement Learning”. In: *Abstraction, Reformulation, and Approximation*. Vol. WS-10-08. AAAI Workshops. AAAI, 2010.

[74] Omid Madani, Mikkel Thorup, and Uri Zwick. “Discounted Deterministic Markov Decision Processes and Discounted All-pairs Shortest Paths”. In: *ACM Trans. Algorithms* 6.2 (Apr. 2010), 33:1–33:25. ISSN: 1549-6325.

[75] Sridhar Mahadevan. “Proto-value functions: Developmental reinforcement learning”. In: *Proceedings of the International Conference on Machine Learning*. 2005, pp. 553–560.

[76] Shie Mannor et al. “Dynamic Abstraction in Reinforcement Learning via Clustering”. In: *Proceedings of the Twenty-First International Conference on Machine Learning*. ACM Press, 2004, pp. 560–567.

[77] R. Andrew McCallum. “Instance-Based Utile Distinctions for Reinforcement Learning with Hidden State”. In: *Proceedings of the Twelfth International Conference on Machine Learning*. Morgan Kaufmann, 1995, pp. 387–395.

[78] John McCarthy. *AI@50: AI Past, Present, Future*. Filene Auditorium et al.

[79] Ishai Menache, Shie Mannor, and Nahum Shimkin. “Q-Cut - Dynamic Discovery of Sub-goals in Reinforcement Learning”. In: *Proceedings of the 13th European Conference on Machine Learning*. Vol. 2430. Springer-Verlag, 2002, pp. 295–306.

[80] K. Michalak and H. Kwasnicka. “Correlation-based Feature Selection Strategy in Neural Classification”. In: *Intelligent Systems Design and Applications, 2006. ISDA ’06. Sixth International Conference on*. Vol. 1. 2006, pp. 741–746.

[81] George A. Miller. “The magical number seven, plus or minus two: some limits on our capacity for processing information”. In: *Psychological Review* 63.2 (1956), pp. 81–97.

[82] Stefania Montani. “Case-Based Decision Support in Time Dependent Medical Domains”. In: *Artificial Intelligence in Theory and Practice III*. Ed. by Max Bramer. Vol. 331. IFIP Advances in Information and Communication Technology. Springer Berlin Heidelberg, 2010, pp. 238–242. ISBN: 978-3-642-15285-6.

[83] Andrew Moore. “Efficient Memory-based Learning for Robot Control”. PhD thesis. Robotics Institute, Carnegie Mellon University, 1991.

[84] Andrew Moore and Chris Atkeson. “Prioritized Sweeping: Reinforcement Learning with Less Data and Less Real Time”. In: *Machine Learning* 13 (1993), pp. 103–130.

[85] Richard E. Nance, C. Michael Overstreet, and Ernest H. Page. “Redundancy In Model Representation: A Blessing Or A Curse?” In: *Proceedings of the 1996 Winter Simulation Conference*. 1996, pp. 701–707.

[86] Trung Nguyen et al. “Online Feature Selection for Model-based Reinforcement Learning”. In: *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*. Ed. by Sanjoy Dasgupta and David Mcallester. Vol. 28. JMLR Workshop and Conference Proceedings, 2013, pp. 498–506.

[87] Ali Nouri and Michael L. Littman. “Multi-resolution exploration in continuous spaces”. In: *Advances in Neural Information Processing Systems 21*. 2009, pp. 1209–1216.

[88] William O’Donohue and Kyle E. Ferguson. *The Psychology of B F Skinner*. SAGE Publications, 2001. ISBN: 9780761917595.

[89] Sarah Osentoski. “Basis function construction in hierarchical reinforcement learning”. In: *Proceedings of the 9th international*. 2010.

[90] William J. Palm. *System Dynamics*. McGraw-Hill Science/Engineering/Math, 2009. ISBN: 0073529273.

[91] Bartosz Papis and Andrzej Pacut. “Multispace, dynamic, fixed-radius, all nearest neighbours problem”. In: *The 16th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*. 2014, to appear.

[92] Bartosz Papis and Andrzej Pacut. “Neighbourhood approach to bisimulation in state abstraction for quantized domains”. In: *The 19th International Conference on Methods and Models in Automation and Robotics (MMAR)*. 2014, to appear.

[93] Bartosz Papis and Paweł Wawrzyński. “dotRL: A platform for rapid Reinforcement Learning methods development and validation”. In: *Federated Conference on Computer Science and Information Systems (FedCSIS)*. 2013, pp. 129–136.

[94] David Park. “Concurrency and automata on infinite sequences”. English. In: *Theoretical Computer Science*. Ed. by Peter Deussen. Vol. 104. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 1981, pp. 167–183. ISBN: 978-3-540-10576-3.

[95] Emmanuel M. Pothos and Nick Chater. “A simplicity principle in unsupervised human categorization”. In: *Cognitive Science* 26.3 (2002), pp. 303–343. DOI: 10.1016/s0364-0213(02)00064-2.

[96] Jefferson Provost, Benjamin J. Kuipers, and Risto Miikkulainen. “Developing Navigation Behavior through Self-Organizing Distinctive State Abstraction”. In: *Connection Science* 18 (2006), p. 2006.

[97] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. 1st. New York, NY, USA: John Wiley & Sons, Inc., 1994. ISBN: 0471619779.

[98] Guido van Rossum. *Python tutorial*. Report CS-R9526. Centrum voor Wiskunde en Informatica (CWI), 1995, pp. iii + 65.

[99] Mark Rubin, Stefania Paolini, and Richard J. Crisp. "The relationship between the need for closure and deviant bias: An investigation of generality and process". In: *International Journal of Psychology* 46.3 (2011), pp. 206–213. ISSN: 1464-066X.

[100] Gavin A. Rummery and Mahesan Niranjan. *On-Line Q-Learning Using Connectionist Systems*. Tech. rep. University of Cambridge, 1994.

[101] Jendrik Seipp and Malte Helmert. "Counterexample-Guided Cartesian Abstraction Refinement." In: *ICAPS*. 2013.

[102] Murray Sidman. "Reflections on stimulus control." In: *Behav Anal* 31.2 (2008), pp. 127–35. ISSN: 0738-6729.

[103] Valeria Simoncini and Daniel B. Szyld. "Recent computational developments in Krylov subspace methods for linear systems". In: *Numerical Linear Algebra with Applications* 14.1 (2007), pp. 1–59. ISSN: 1099-1506.

[104] Satinder Singh. *Successes of Reinforcement Learning*. [Online; accessed 1-September-2014]. 2008. URL: `http://umichrl.pbworks.com/w/page/7597597/Successes\%20of\%20Reinforcement\%20Learning`.

[105] Satinder P. Singh and Richard S. Sutton. "Reinforcement learning with replacing eligibility traces". In: *Machine Learning* 22.1-3 (1996), pp. 123–158. ISSN: 0885-6125.

[106] Carl-Johan Sjöstedt. "The design of modular dynamical fluid simulation systems". In: *Proceedings from the OST Conference, KTH Machine Design*. QC 20101221. 2005, pp. 1–12.

[107] Nathan Sprague. "Basis iteration for reward based dimensionality reduction". In: *IEEE 6th International Conference on Development and Learning*. 2007, pp. 187–192.

[108] Ralf Stecking and Klaus B. Schebesch. "Variable Subset Selection for Credit Scoring with Support Vector Machines". In: *Operations Research Proceedings 2005*. Ed. by Hans-Dietrich Haasis, Herbert Kopfer, and Jörn Schönberger. Vol. 2005. Operations

Research Proceedings. Springer Berlin Heidelberg, 2006, pp. 251–256. ISBN: 978-3-540-32537-6.

[109] Richard Sutton, Doina Precup, and Satinder Singh. “Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning”. In: *Artificial Intelligence* 112 (1999), pp. 181–211.

[110] Richard S. Sutton. *Dyna, an Integrated Architecture for Learning, Planning, and Reacting*. 1991.

[111] Richard S. Sutton. “Learning to predict by the methods of temporal differences”. In: *Machine Learning* 3.9-44 (1988).

[112] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning*. The MIT Press, 1998.

[113] Kari Torkkola. “Feature extraction by non parametric mutual information maximization”. In: *J. Mach. Learn. Res.* 3 (Mar. 2003), pp. 1415–1438. ISSN: 1532-4435.

[114] Hiroaki Ueda et al. “State Space Segmentation for Acquisition of Agent Behavior”. In: *Proceedings of the IEEE/WIC/ACM international conference on Intelligent Agent Technology*. IAT ’06. IEEE Computer Society, 2006, pp. 440–446. ISBN: 0-7695-2748-5.

[115] Christopher Watkins and Peter Dayan. “Q-Learning”. In: *Machine Learning* 8 (1992), pp. 279–292.

[116] Paweł Wawrzyński. “Learning to Control a 6-Degree-of-Freedom Walking Robot”. In: *The International Conference on Computer as a Tool*. 2007, pp. 698–705.

[117] Shimon Whiteson, Matthew E. Taylor, and Peter Stone. *Adaptive Tile Coding for Value Function Approximation*. 2007.

[118] Michael Widenius and Davis Axmark. *Mysql Reference Manual*. Ed. by Paul DuBois. 1st. O’Reilly & Associates, Inc., 2002. ISBN: 0596002653.

[119] Stephen Willard. *General topology*. Addison-Wesley series in mathematics. Addison-Wesley Pub. Co., 1970.

[120] J. Záworka. “Project SIMONE-achievements and running development”. In: *Proceedings of the Second International Workshop SIMONE on Innovative Approaches to Modeling and Optimal Control of Large Scale Pipeline Networks*. 2003, pp. 1–24.