UNIVERSITY OF TURIN

DEPARTMENT OF COMPUTER SCIENCE



UNIVERSITÀ
DI TORINO

PHD PROGRAMME IN COMPUTER SCIENCE
CYCLE 35

THESIS TITLE

BUILDING EVOLVING SYMBOLIC SYSTEMS

Thesis' author:
Gabriele Sartor

Supervisor:
Rosa Meo

PhD Programme Co-ordinator
Viviana Patti

ACADEMIC YEARS OF ENROLMENT 2019-2022
CODE OF SCIENTIFIC DISCIPLINE INF/01

Gabriele Sartor: *Building Evolving Symbolic Systems*, © April 2023

ABSTRACT

In recent years, Artificial Intelligence (AI) systems have made remarkable progress in various tasks. Deep Reinforcement Learning (DRL) is an effective tool for agents to learn policies in low-level state spaces to solve highly complex tasks. Recently, researchers have introduced Intrinsic Motivation (IM) to the RL mechanism, which simulates the agent's curiosity, encouraging agents to explore interesting areas of the environment. This new feature has proved vital in enabling agents to learn policies without being given specific goals.

However, even though DRL intelligence emerges through a sub-symbolic model, there is still a need for a sort of abstraction to understand the knowledge collected by the agent. To this end, the classical planning formalism has been used in recent research to explicitly represent the knowledge an autonomous agent acquires and effectively reach extrinsic goals. Despite its limited expressive capabilities, the Planning Domain Definition Language (PDDL) demonstrated usefulness in reviewing the knowledge gathered by an autonomous system, making explicit causal correlations, and can be exploited to find a plan to reach any state the agent faces during its experience.

In this thesis, we propose two different architectures implementing open-ended learning systems able to synthesize their experience into a PDDL representation and update it over time. The first implementation is a ROS-based intrinsically motivated agent, able to explore a robotic environment exploiting its PDDL domain representation and updating it over time, learning new skills and facing unexpected events. Then, a second implementation focused on the exploration able to iteratively: (a) discover options, (b) explore the environment using options, (c) abstract the knowledge collected and (d) plan. Abstract representation can be used to visualize extrinsic and intrinsic goals set by an external user and by the agent's curiosity. This work proposes an alternative approach to implementing open-ended learning architectures exploiting low-level and high-level representations to extend its own knowledge continuously.

ACKNOWLEDGMENTS

I wish to express my sincere gratitude to several individuals who have contributed to the completion of this research work.

I would like to thank my supervisor, Rosa Meo, for her constant availability, support, and trust in my abilities. She has helped me shape this research and expand my knowledge of the field.

I am deeply grateful to my colleagues of the Institute for Cognitive Sciences and Technologies (ISTC) of Rome, in particular Angelo Oddi, Riccardo Rasconi and Vieri Giuliano Santucci, for their invaluable contributions to my work.

A special thanks goes to the reviewers, George Konidaris (Brown University) and Andrea Orlandini (Institute for Cognitive Sciences and Technologies of Rome), for their time and precious suggestions to improve the thesis.

I extend my heartfelt thanks to my friends, whose unwavering friendship and support have been a source of strength throughout this journey. Their encouragement, discussions, and shared experiences have enriched my work.

I would also like to express my deep appreciation to my family for their unwavering support, love, and understanding during my academic pursuits.

Lastly, but certainly not least, I am incredibly grateful to my girlfriend Francesca for her love, support, and understanding through this intense journey. Her presence in my life has brought joy, balance, and belief in my abilities, helping me to overcome challenges and reach my goals.

CONTENTS

1 Introduction 1

1.1 Overview 1

1.2 Problem and Objectives 2

1.3 Research Methodologies 4

1.4 Thesis Structure 5

2 Background 7

2.1 Building blocks and state of the art 7

2.1.1 From primitives to options 8

2.1.2 Options and Classical Planning 10

2.1.3 On the way to Long Term Autonomy 13

2.2 Related works 16

2.2.1 From Skills To Symbols 18

2.2.2 GRAIL: Goal-Discovering Robotic Architecture for Intrinsically-Motivated Learning 28

3 Intrinsically Motivated Planning Architecture for Curiosity-driven roboTs 32

3.1 The IMPACT Architecture 34

3.1.1 The Decisional Layer 34

3.1.2 The Functional Layer 36

3.1.3 The Executive Layer 38

3.2 Operational Evaluation 40

3.2.1 Low-level Operators and variables description 42

3.2.2 The Initial PDDL domain 44

3.2.3 Automatically synthesizing the new PDDL domain 45

3.2.4 Autonomous learning of new skills 47

3.2.5 Planning over the new Robotic Arm PDDL Domain 51

3.3 Discussion and Future Work 51

4 Intrinsically Motivated High-Level Planning for Agent Exploration 53

4.1 Architecture description 54
4.1.1 Option Discovery 55
4.1.2 Exploration 57
4.1.3 Abstraction 59
4.1.4 Goal Selector 61
4.1.5 Planning 65
4.2 Experimental analysis 65
4.2.1 Environment setup 65
4.2.2 The cycle 67
4.2.3 Results 70
4.2.4 Issues and limitations 77
5 Discussion and Conclusions 79
5.1 Future Work 79
5.2 Discussion 80
5.3 Conclusions 82

I Appendix
A Background: supplementary material 84
A.1 PDDL: Planning Domain Definition Language 84
A.2 PPDDL: the Probabilistic PDDL 87
A.3 ROS: Robot Operating System 88
B Experiments of Chapter 3 90
B.1 Robotic Arm Scenario: the Initial PDDL Domain 90
B.2 Robotic Arm Scenario: the initial set of Symbols 93
B.3 Robotic Arm Scenario: the New PDDL Domain 97
B.4 Robotic Arm Scenario: the new set of Symbols 100
C Experiments of Chapter 4 105

Bibliography 112

LIST OF FIGURES

Figure 2.1 Representation of the option *reach the ball*. 9

Figure 2.2 Symbolic and sub-symbolic representation examples. 16

Figure 2.3 The representation of an option respecting the *subgoal option condition*. 20

Figure 2.4 GRAIL architecture with main components. 29

Figure 3.1 The *Discover-Plan-Act* cycle. 32

Figure 3.2 IMPACT high-level software architecture. The red blocks are novel elements implemented to actually extend the three-layered architecture. 34

Figure 3.3 The representation of the Robotic Arm use case scenario. 41

Figure 3.4 The Rover Model used in the Robotic Arm Scenario. 42

Figure 3.5 The view field of the rover in the Robotic Arm scenario. 43

Figure 3.6 The rover grasping a cube-shaped rock and the vase-shaped rock. 47

Figure 3.7 The learning curve of the new skill of the rover. 50

Figure 4.1 The architecture of the intrinsically motivated agent using high-level planning for its exploration. 55

Figure 4.2 The conceptual depiction of the selection of the symbols necessary to represent a goal. 62

Figure 4.3 All the domain configurations used in the experimental analysis. 66

Figure 4.4 An extract of the PPDDL generated by the abstraction procedure. 69

Figure 4.5 Example of the symbols' semantics of a generated PPDDL's operator. 70

Figure 4.6 Example of PPDDL problems synthesized by the system. 71
Figure 4.7 The plans generated ω^{g} and ω^{EX}. 71
Figure 4.8 Success rate of ω^{g} using different exploration strategies over time. 73
Figure 4.9 Symbols and operators generated over time in *domain*3 and *domain*4. 75
Figure 4.10 Visualization of abstraction time and operators in different cycles. 76
Figure 4.11 The graphical representation of a potentially noisy symbol. 77
Figure A.1 *Dock Worker Robots* domain representation. 84
Figure A.2 Schemes of topics and services workflow. 88

LIST OF TABLES

Table 4.1 Obstacles to be solved to end the game. 67

Table 4.2 Main parameters used in the different environment configurations. 74

LIST OF ALGORITHMS

Algorithm 1 Abstraction algorithm 22

Algorithm 2 Discover-Plan-Act algorithm 56

Algorithm 3 Option Discovery 58

ACRONYMS

AP	Abstraction Procedure
AGI	Artificial General Intelligence
AI	Artificial Intelligence
DPA	Discover Plan Act
DQN	Deep Q-Learning
ESA	European Space Agency
FOL	First Order Logic
GRAIL	Goal-Discovering Robotic Architecture for Intrinsically-Motivated Learning
IM	Intrinsic Motivation
IMPACT	Intrinsically Motivated Planning Architecture for Curiosity-driven roboTs
KB	Knowledge Base
LTA	Long-Term Autonomy
ML	Machine Learning
PDDL	Planning Domain Definition Language
PS	Planning System
RL	Reinforcement Learning
SPA	Sense Plan Act
SVM	Support Vector Machines

TABLE OF NOTATION

Notation	Meaning
$\{a, b, ..., u\}$	set
$(a, b, ..., u)$	tuple
$[a, b, ..., u]$	sequence
a, A	action, set of actions
s, S	state, set of states
o, O	option, set of options
$o(I, \pi, \beta)$	option with the policy π, initiation set I and termination condition β
id, ID	initiation data entry, initiation dataset
td, TD	transition data entry, transition dataset
s_{init}, s_g	initial state of the game, final state of the game
σ, Σ	symbol, set of symbols
Σ_{target}	set of symbols grounding the state s_{target}
ω, Ω	plan, set of plans
$\mathcal{P}$	planning problem
$\mathcal{D}$	planning domain
$\alpha(pre, eff^+, eff^-)$	planning action with precondition pre, positive effects eff^+ and negative effects eff^-
Cl_c	probabilistic classifier learned over the class c
$Cl_c(s)$	probability that s belongs to the class c
$\eta(s)$	curiosity value for state s
ϕ	feature
f	factor
$p(s)$	probability distribution over the state s

1

INTRODUCTION

1.1 OVERVIEW

The development of Artificial General Intelligence (AGI) has always been the most challenging objective of AI research. AGI aims to develop machines that can perform any intellectual task that a human can carry out and potentially beyond. Such a system should adapt to new environments, learn from experience, understand natural language, and solve complex problems in real-time. The concept of AGI often evokes the notion of a robotic entity operating in the physical realm, exhibiting human-like behaviour and reasoning capabilities. Despite significant progress in AI, the development of AGI remains elusive, and several challenges must be overcome before we can achieve this ambitious goal.

Even though AGI is a fluid research field proposing different perspectives and evolving concepts [39], we could say that, generically, one of the main challenges in implementing AGI is the need for a *flexible and adaptable cognitive architecture*. The human brain is incredibly complex and is capable of adapting to a wide range of tasks and environments. By contrast, current AI systems, referred to as "narrow AI" [53], are highly specialized and designed to perform specific tasks. Developing an AI system that can adapt to new tasks and environments is a significant challenge that requires a deep understanding of the underlying principles of cognition. The workings of the brain remain a mystery; thus, replicating its architecture and mechanisms is a non-trivial challenge. Cognitive scientists have attempted to replicate AGI by creating systems with architectures that closely mimic the human brain, while other AI researchers have focused on learning complex policies using sub-symbolic models, despite their disregard for the human brain's structure. To date, neither of these approaches has developed a form of general intelligence comparable to that of humans. Clearly, one critical characteristic of such systems is their *responsiveness* to the complex and constantly

changing real world. A responsive AGI must be able to perceive and process all significant information efficiently in order to provide rapid solutions.

To replicate human intelligence, the *ability to learn from a limited amount of experience* plays a significant role, which contrasts with the approach adopted by current machine learning algorithms, requiring vast amounts of data to learn and generalize from. Humans can learn complex tasks from very few examples, suggesting that we must develop algorithms that can learn from a limited amount of data exploiting common sense knowledge and abstract concepts. The assimilated concepts should enable a virtuous circle driving the agent learning quicker new information exploiting the previous knowledge, the so-called *learning to learn* mechanism.

All these aspects have been faced individually, and researchers have proposed solutions for limited domains. An adequate system integrating all these parts is still missing, and this work aims to propose an alternative prototype solution.

1.2 PROBLEM AND OBJECTIVES

To build an effective cognitive system, in addition to a perfect understanding of the brain in terms of architecture and processes, it is necessary to consider *the importance of the representation* used to manage the knowledge. For instance, human beings can reason at different levels of abstraction, ranging from sensorial lower levels to high levels that deal with more abstract or generic concepts.

Although different methods have been proposed to manage knowledge, it is still unclear which is the most effective approach to reach AGI. Some researchers argue that the agent endowed with a reward function should be sufficient to reach that level of intelligence, emerging from its behaviours [98] captured by a sub-symbolic model. Still, other researchers investigate the utility of adopting textual representation (e.g. symbols, ontologies, etc.) to define intelligence and perform reasoning [54, 55]. However, it seems commonly accepted to require several layers of abstractions to deal with the complexity of the real world, from the sensorimotor space to more conceptual ones [51]. Reasoning decisions in the sensorimotor space is impracticable because of its high dimensionality. Instead,

reasoning and learning at a higher level are more natural and typical behaviours humans adopt to make decisions on different aspects with a different level of granularity. Considering symbolic and sub-symbolic systems, both present abstractions: in the first case, the use of symbols itself and the second case, neural network layers identify features of different granularities.

In autonomous agents, the efficient and effective acquisition of new knowledge is crucial for developing intelligent systems capable of adapting to a continuously changing environment. Therefore, in addition to possessing a robust knowledge representation, agents must be equipped with the ability to acquire new knowledge quickly and efficiently. This could be achieved by designing knowledge representations that provide building blocks for collecting new knowledge in a more efficient manner. As the agent learns new abilities and encounters different environmental configurations, it should be able to discover novel goals and capabilities, creating a virtuous cycle of learning and adaptation, also referred to as *open-ended learning (OEL)* [79]. By incorporating these capabilities into autonomous agents, the way towards developing more advanced and capable AI systems can be paved. In symbolic systems, symbols can be seen as building blocks of the agent's knowledge and could be exploited to reason on the next goal to be reached and also to discover new capabilities and symbols.

This work proposes a possible approach to building agents evolving, exploiting their previous knowledge and representing it symbolically. Symbols are an old-fashioned tool to label groups of environment configurations, generating an abstraction on which it is possible to reason effectively [81]. The symbolic approach also can have an essential role in the revision of the knowledge acquired and in understanding the motivations driving the agent to perform its choices, a fundamental feature to guarantee the *trustability* of the system. Especially, symbols provide reasoning and representation building blocks which can be exploited to drive the agent exploration, using planning, towards states with an interesting semantic, made explicit by the symbols. The aim of the research is to investigate the possibility of using classical planning in intrinsically motivated systems to extend the symbolic representation itself in an efficient way. Symbols can give the agent the capability of naming particular environmental aspects,

which can be combined to identify reachable and, potentially, unseen goals to define an imagined desired goal. The virtuous loop generated by such systems, expanding their sub-symbolic and symbolic knowledge continuously, could be a particularly compelling approach to explore further.

1.3 RESEARCH METHODOLOGIES

To address the problem discussed in the previous section, it is possible to combine the results of different research fields to create systems able to autonomously learn knowledge and exploit it to extend their intelligence further. Our approach has been integrating technologies coming from AI and cognitive science to integrate mainly four functionalities:

- Intrinsic impulse to discover new actions and situations;
- Action learning;
- Abstract representation synthesis;
- Problem-solving.

These aspects covered by different interrelated researches, together implementing an open-ended learning system, can potentially produce an AGI. However, the results of individual research studies are limited to specific use cases and it is still unclear the best approach to follow (symbolic, emergentist or hybrid) to produce it [39]. In this work, the technologies used to implement these functionalities are, respectively, 1) intrinsic motivation (IM), 2) reinforcement learning, 3) synthesis in classical planning representation, and 4) off-the-shelf planners to solve classical planning problems. With this set of tools, the purpose is not to solve the ambitious goal of implementing the AGI but building an alternative architecture to face the open-ended learning problem, which is an important but limited part of the problem of synthesizing a general intelligence.

1.4 THESIS STRUCTURE

This thesis proposes a possible architecture integrating technologies dealing with low-level and high-level information to learn actions, create an abstraction for the situations encountered and plan actions to reach goals required by external input or an intrinsic impulse (i.e. curiosity, interest, etc.) towards new configurations of the environment. Dealing with different levels of abstraction means learning complex policies creating sophisticated behaviours (e.g. moving robot's actuators), but also attributing them a naming used to reason on a chain of complex actions to reach goals distant in terms of time and space. The thesis is structured as follows:

- Chapter 2 discusses the formal background necessary to understand the architectures implemented in the rest of the thesis, the related works and the systems present in literature which have been integrated into the proposed solutions;
- Chapter 3 presents an architecture called IMPACT (Intrinsically Motivated Planning Architecture for Curiosity-driven roboTs) implemented in ROS for creating robotic planning agents able to use intrinsic motivations to learn new abilities and concepts, especially in the case of unexpected occurred events;
- Chapter 4 talks about a second implementation of agent exploiting planning to explore the environment faster to learn new knowledge more effectively. Compared to the previous system, in this case, the focus is on repeatedly executing the Discover-Plan-Act loop to discover new actions from scratch, abstract them and exploit them to reach extrinsic and intrinsic goals, producing a virtuous loop expanding the symbolic knowledge of the agent;
- Chapter 5 discusses the results of the work and suggests new directions for future works;
- Finally, an appendix includes additional technical details of Chapter 2, describing the PDDL definition language and the middleware ROS, and

Chapter 3, such as the symbolic representations produced during the experiments of the IMPACT system. Then, the symbolic representation data of the experiments of Chapter 4 are included for each domain, strategy and trial.

2

BACKGROUND

This chapter introduces the fundamental concepts required to comprehend developments and proposed solutions in the domain of intelligent agents that augment their initial knowledge. This thesis aims to investigate systems that can effectively handle unforeseen events and accomplish new objectives by utilizing an internal symbolic representation, a method which a rigid and invariable but inspectable representation has always characterized. Nonetheless, by incorporating intrinsic motivations and other components, it is possible to increase the flexibility of these systems while preserving their interpretability.

2.1 BUILDING BLOCKS AND STATE OF THE ART

To reach a high level of autonomy, an agent acting in the low-level space, sensing the environment with its sensors and modifying it through its actuators, must implement a series of layers of abstraction over its state and action spaces. As the human being requires both simple and more complex concepts to perform their activities, so robots have the necessity of building their own abstract representation of the world to deal with the related complexity, using *labels* to refer to actions and events to be recognized and reasoned over.

Therefore, solutions to deal with high-dimensional state and action spaces have been proposed in the intelligent agent research field. A historical example has been the use of *Classical Planning* to abstract the behaviour of an agent in an environment [81], reducing the number of possible situations by unifying states with similar characteristics and, consequently, the expected transitions in the world. Also the employment of the *options framework* to simplify and optimize the achievement of complex tasks in Reinforcement Learning has been viewed as action abstraction [101, 102]. The combination of different layers of abstraction can provide the agent with both the capabilities of working at a low

level, with the precise position of its actuators, and at a high level, reasoning over sequences of actions able to reach distant goals that cannot be reached by reasoning directly in the high-dimensional low-level space.

Finally, exploiting its higher abstractions, the agent can reason over unexpected events and its achieved results, preparing strategies to continue performing useful tasks. This is the Long Term Autonomy (LTA) [66, 79] challenge, which can be tackled only by adopting learning capabilities that extend the initial knowledge and skills of the agent to solve tasks and situations not considered at design time. Techniques based on Intrinsic Motivations (IMs) [68, 92] have emerged to explore unknown aspects of the environment and push the agent to find new ways to reach extrinsic and intrinsic goals.

All these aspects are discussed in the next sections, to understand the systems implementing and combining these characteristics to produce abstract, robust and extendable behaviours, which are presented in Chapter 3 and 4.

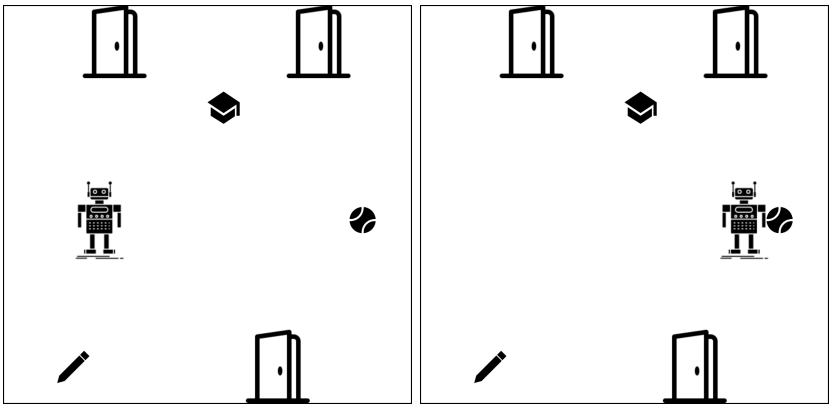
2.1.1 *From primitives to options*

As discussed before, at the lowest level, the agent sees the world with its sensor's values and changes it through the movement of its actuators. The most common formalism at this stage to deal with this type of representation is the Markov Decision Process (MDP), which models the environment as the tuple:

$$(S, A, R, T, \gamma), \tag{2.1}$$

in which S represents the set of possible high-dimensional states where each $s \in S$ is described by a vector of real values returned by the agent's sensors, A describes the set of low-level actions $a \in A$ in some cases also called *primitives*, R the reward function where $R(s, a, s')$ is a real value returned executing a from state s achieving s', T the transition function describing for $T(s'|s, a)$ the probability of reaching the state s' executing a from s, and the discount factor $\gamma \in (0, 1]$ describing the agent's preference for immediate over future rewards. Usually, in this setting, the goal is to maximize *return*, defined as

$$\mathcal{R} = \sum_{i=0}^{\infty} \gamma^i R(s_i, a_i, s_{i+1}). \tag{2.2}$$



(a) (b)

Figure 2.1: Representation of the option *reach the ball*.

However, dealing with the state and action spaces of the formulation (2.1) is, in certain cases, impractical due to the high dimensional spaces considered. An effective formalism introduced to reduce the complexity of the problems is the *option* framework [102]. The *option* is a temporally-extended action definition which employs the following abstracted representation:

$$o = (I_o, \pi_o, \beta_o), \tag{2.3}$$

where the option o is defined by an *initiation set* $I_o = \{s | o \in O(s)\}$ representing the set of states in which o can be executed, a *termination condition* $\beta_o(s) \rightarrow [0, 1]$ returning the probability of termination upon reaching s by o, and a policy π_o which can be run from a state $s \in I_o$ and terminated reaching s' such that the probability $\beta_o(s')$ is sufficiently high. A *policy* is a function defining the behavior of the agent, mapping the perceived state of the environment to the action to be taken from it [101]. It is worth noticing that options create a temporally-extended definition of the actions [102]. Indeed, the option is an abstraction defining an action as repeated execution of policy π from a state $s \in I_o$ to another s' in a maximum amount of time steps τ.

For instance, Figure 2.1 depicts an environment before and after executing the option *"reach the ball"*, respectively Figure 2.1a and 2.1b. When the agent wants to execute this option, it checks whether its current state belongs to the *initiation set* of the option. In our case, and assuming that the option's policy simply consists in moving towards the ball, the agent can execute the option

and runs the policy π_o until the *termination condition* returns a sufficiently high probability of success or τ time steps are reached. In the Figure 2.1b the options successfully terminates getting close to the ball.

Passing from low-level actions to options reduces the agent's action space. Adopting options in the MDP formalism implies moving to the semi-Markov Decision Process (SMDP):

$$(S, O, R, P, \gamma), \tag{2.4}$$

where S is the original state space, $O(s)$ is the set of options executable from state s, $R(s, \tau|s, o)$ describes the reward expected executing $o \in O(s)$ from state s reaching s' after τ time steps, $P(s', \tau|s, o)$ returns the probability of reaching state s' after τ time steps executing $o \in O(s)$ from state s, and the discount factor $\gamma \in (0, 1]$. Using options entails moving in the low-level state space S with abstracted actions permitting the agent to perform extended and more complex behaviors, reducing the number of actions to achieve a certain task and simplifying the problem.

2.1.2 *Options and Classical Planning*

The option formalism and its way of abstracting the dynamics of an environment share common characteristics with *classical planning*, in which the world is described in a simplified formal description considering only the aspects necessary to solve the agent's task [81]. Planning is the field of research studying formal methods to automatically find solutions, also called plans, to tasks requiring a sequence of actions $[a_1, \ldots, a_n]$ to reach a goal state s_g from an initial state s_{init}. The plan ω is obtained by giving in input a model of the environment dynamics and the problem definition to a planner, returning a solution applying general optimization algorithms.

Classical planning is a particular instance of this methodology exploiting a logical language (i.e. propositional logic, first-order logic, etc.) to capture an abstract symbolic description of the world [81]. The *symbol*, which is the core of classical planning, is a name given to a certain set of states $s \in S$ satisfying a specific condition in the sensorimotor space. This *mapping* from the symbol to

the real world states is called *grounding*. Specifically, a symbol $\sigma_Z \in \Sigma$, with Σ set of the available symbols, is the name given to a *test* τ_Z, and the corresponding set of low-level states where the test is satisfied $Z = \{s \in S \mid \tau_Z(s) = True\}$, with the high-dimensional low-level state space S [51]. To determine whether or not a low-level state s_i belongs to the state set Z, the $\sigma_Z(s_i)$ test is run, which will return either *True* or *False*. Again, both Z and σ_Z provide the *semantics* of the symbol; Z is the symbol's *grounding set*, while σ_Z is its *grounding classifier*. Symbols can be combined using *operators* having the following meaning:

- $\neg\sigma_Z$ corresponds to the *negation* of symbol σ_Z;
- $\sigma_X \vee \sigma_Y$ corresponds to the *union* of symbols σ_X and σ_Y (i.e., the union of their respective *grounding sets*);
- $\sigma_X \wedge \sigma_Y$ corresponds to the *intersection* of symbols σ_X and σ_Y (i.e., of their respective *grounding sets*).

In classical planning, operators and symbols are used to describe actions in the following form

$$\alpha_i = (pre_i, eff_i^+, eff_i^-), \tag{2.5}$$

meaning that the action $\alpha_i \in \mathcal{A}$ can be executed when all the symbols $\{\sigma|\sigma \in pre_i\}$, also called *preconditions*, are *True*, and executing α_i produces the changes of the value of some symbols, also called *effects*, implying that all symbols $\{\sigma|\sigma \in eff_i^+\}$ assume the value *True* and symbols $\{\sigma|\sigma \in eff_i^-\}$ become *False*.

Finally, using symbols Σ and high-level actions $\mathcal{A}$ as building blocks, it is possible to define the model of environment $\mathcal{D}$, also called *domain*, and the *problem* $\mathcal{P}$ to solve. A classical planning domain can be defined as

$$\mathcal{D} = (\Sigma, \mathcal{A}, \Gamma), \tag{2.6}$$

using a set of symbols Σ, actions $\mathcal{A}$ and a *state-transition function* $\Gamma : \hat{\Sigma} \times \mathcal{A} \to \hat{\Sigma}$, where $\hat{\Sigma}$ is the set of possible subsets of Σ. The *state-transition* function $\Gamma(\hat{\Sigma}_s, \alpha_i) = (\hat{\Sigma}_s - eff_i^-) \cup eff_i^+$, if α_i is applicable to $\hat{\Sigma}_s$, where $\hat{\Sigma}_s$ is the set of symbols whose *grounding set* intersection defines the state $s \in S$. These elements are sufficient to describe the dynamics of the environment, over which

the planner can reason and create chains of actions to reach a final goal. The function Γ encapsulates the transition model of the environment in each action model described by (2.5), defining the possible way to build sequences of them. Instead, the problem can be formalized as

$$\mathcal{P} = (\hat{\Sigma}_{s_{init}}, \hat{\Sigma}_{s_g}), \tag{2.7}$$

where $\hat{\Sigma}_{s_{init}}$ is the set of symbols whose *grounding set* intersection describes s_{init} and $\hat{\Sigma}_{s_g}$ the set of symbols whose *grounding set* intersection characterizes s_g. Then, the plan solving the problem $\mathcal{P}$ is expressed as the sequence of actions

$$\omega = [\alpha_1, ..., \alpha_n], \tag{2.8}$$

resulting in a sequence of state transitions

$$[\hat{\Sigma}_{s_0}, \hat{\Sigma}_{s_1}, \dots, \hat{\Sigma}_{s_n}], \tag{2.9}$$

such that $\Gamma(\hat{\Sigma}_{s_0}, \alpha_1) = \hat{\Sigma}_{s_1}, \Gamma(\hat{\Sigma}_{s_1}, \alpha_2) = \hat{\Sigma}_{s_2}, \dots, \Gamma(\hat{\Sigma}_{s_{n-1}}, \alpha_n) = \hat{\Sigma}_{s_n}$ with initial state $\hat{\Sigma}_{s_0} = \hat{\Sigma}_{s_{init}}$ and final state $\hat{\Sigma}_{s_n} = \hat{\Sigma}_{s_g}$. In particular, the formulation presented based on a finite set of symbols Σ and state-transition system $\mathcal{T} = (\Sigma, \mathcal{A}, \Gamma)$ is called a *set-theoretic representation* [81]. To implement classical planning systems, it is necessary to describe the domain of the considered world and the tackled problem by using a planning definition language, and give both as input to a solver returning a solution ω. In this work, Planning Domain Definition Language (PDDL) and its probabilistic extension PPDDL are the languages employed, which are discussed respectively in Appendix A.1 and A.2.

It is worth noticing that option o and the planning action α are similar. Indeed, o can be converted in its corresponding high-level action α finding the right set of symbols pre, eff^+, eff^- whose grounding sets satisfy the classifiers I_o as preconditions and β_o as effects, for the execution of π_o. As it will be described in section 2.2, the similarity between options and set-theoretic planning actions has been exploited to create automatic abstraction procedures able to convert options' execution data into a working high-level planning description, enabling the creation of systems integrating low-level and high-level information.

2.1.3 *On the way to Long Term Autonomy*

Although planning is a powerful tool for solving problems, it lacks flexibility. Indeed, the standard approach used in this field is to employ an expert on the task the agent is going to solve, to define in advance an arbitrary set of symbols and actions to reach the final goal of the system. Consequently, if it encounters some issues or unexpected events, the agent cannot continue its activities or exploit new favourable situations.

Instead, human-like systems need to be able to handle various situations, from a slight difference in the world to a new goal to be achieved, resulting in a simple variation in the already available skills and the acquisition of new skills. In the second case, when the agent needs to learn new actions to achieve new goals, the problem takes the name of *open-ended learning* (OEL). When this process is repeated continuously by the system, it is called *lifelong open-ended learning autonomy* (LOLA) [79] or *long-term autonomy* (LTA) [66]. In other words, LTA assumes the capability of acquiring new knowledge to perform its activities robustly for a long time in a partially predictable environment.

Architectures tackling the LTA challenge should include different modules simulating *motivations*, *memory*, *online learning*, *attention* and other characteristics of the human being. The following paragraphs briefly tackle the management of motivations and knowledge, which are the main aspects debated in the next chapters.

INTRINSIC MOTIVATION The first cited aspect, the impulse to drive the agent away from the monotony of its activities, which psychologists and cognitive scientists have studied under the name of *intrinsic motivation*, is one of the most important elements enabling OEL. The research in the field of Intrinsic Motivation (IM) concerns the study of human behaviors not influenced by external factors but characterized by internal stimulus (i.e. curiosity, exploration, novelty, surprise). In the case of artificial agents, we can summarize such aspect as anything that can drive the agent's behavior which is not directly dependent on its assigned task.

The insights provided by the IMs gave the researchers new ideas to model the stimulus of the agent (e.g. curiosity). Indeed, some models have been implemented driven by prediction error (PE), more precisely on its improvement [87], learning how to predict better the effects of the agent's actions. A formal definition of agent driven by its curiosity has been formulated by Schmidhuber [92] as simply maximizing its *future success* or *utility*, which is the conditional expectation

$$u(t) = E_\mu \left[\sum_{\tau=t+1}^{T} r(\tau) \middle| h(\leq t) \right], \tag{2.10}$$

over $t = 1, 2, \ldots, T$ time steps, receiving in input a vector $x(t)$, executing the action $y(t)$, returning the reward $r(t)$ at time t, taking into consideration the triples $h(t) = [x(t), y(t), r(t)]$ until time step t as previous data experienced also called *history*, with respect to an unknown probability distribution μ reasoning the responses of the environment. To maximize (2.10), the agent also has to build a predictor $p(t)$ of the environment to anticipate the effects of its actions. The reward signal is defined as follows

$$r(t) = g(r_{ext}(t), r_{int}(t)), \tag{2.11}$$

which is a certain combination g of an external reward $r_{ext}(t)$ and an intrinsic reward $r_{int}(t)$. In particular, $r_{int}(t+1)$ is seen as *surprise* or *novelty* in assessing the improvements in the results of p at time $t+1$

$$r_{int}(t+1) = f|C(p(t), h(\leq t+1)), C(p(t+1), h(\leq t+1))|, \tag{2.12}$$

where $C(p, h)$ is a function evaluating the *performance* of p on a history h and f is a function combining its two parameters (e.g. in this case, it could be simply the improvement $f(a, b) = a - b$). It is important to notice that, as a baby does, an intrinsically motivated agent needs to find regularities in the environment to learn something. Consequently, if something does not present a *pattern*, there is nothing to learn, becoming boring for both an agent and a human being.

In literature, IMs have also been categorized into different typologies [7, 59, 68]. An important discriminant aspect is the kind of signal received by the agent which can be of two types: *knowledge-based* (KB-IMs), depending on the prediction model of the world, and *competence-based* (CB-IMs), depending on

the improvement of the agent's skills. CB-IMs proved to be more suitable for gathering a repertoire of skills, and it is used by GRAIL [87], which is an IMs-based system used in Chapter 3 and discussed in detail in section 2.2.2.

KNOWLEDGE REPRESENTATION The data experienced during the agent's life can be managed in several ways. Indeed, different architectures have been used to build intelligent agents, starting from the symbolic systems that characterized the first AI systems [81], to sub-symbolic systems, which gained enormous importance with the growth of deep learning [10]. These architectures can be divided into the following groups:

- **Symbolic**, exploiting textual labels to refer to concepts, events and actions [36, 54, 55, 64, 81]. These architectures were more popular in the first decades of AI research, nowadays still used for limited planning and scheduling tasks;
- **Sub-symbolic**, based on mathematical models encapsulating the knowledge in its internal weights, or parameters, producing intelligent behaviours without the possibility of inspecting its meaning (e.g. deep learning)[10, 15];
- **Hybrid**, integrating both the approaches (e.g. using symbols to manage a high-level model of the world and sub-symbolic models to learn low-level policies [3, 34, 66]).

All these approaches present weak points. For instance, there is no architecture really able to merge past and new information effectively. Although deep neural networks evolve continuously, incorporating new data, the model used cannot be changed in its structure to be extended when it becomes saturated with information. At the same time, this limitation is also valid for the symbolic systems able to generate their representation automatically from experience [51].

However, it seems straightforward that intelligent systems need to create different layers of abstraction, building a hierarchy of representations (e.g. hierarchies of MDP or other formalisms) to deal with the world's complexity, as the human being does. A central point is that an abstraction appears necessary

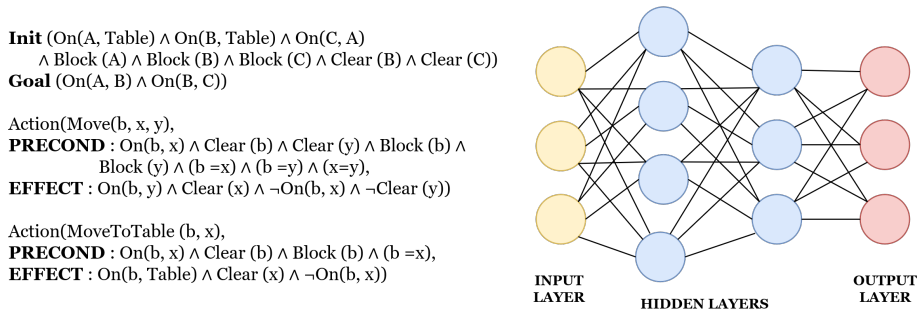


Figure 2.2: An example of symbolic representation (left). The conceptual architecture of a neural network containing several parameters (right).

to identify a suitable representation of the task to optimize its solution [49]. The systems presented in this thesis have mainly used the symbolic representation integrated with other mathematical tools, and the production of the symbolic representation has been carried out by an abstraction procedure described in section 2.2.1.

2.2 RELATED WORKS

In the last few years, new AI systems have solved incredible tasks. These tasks include real-world games, such as chess [20] and Go [95–97], videogames such as Atari [67], Dota [14], and different robotics tasks [8, 44, 47, 62]. These results have been mostly achieved through the intensive use of RL with the rediscovered technology of neural networks and deep learning [10]. Usually, "standard" RL focuses on acquiring policies that maximise the achievement of fixed assigned tasks (through reward maximisation) with a predefined collection of skills. New approaches have been proposed to enrich RL, allowing the agent to extend its initial capabilities over time inspired by neuroscience and psychology. Indeed, studies on animals [19, 40, 60] and humans [12, 13, 82] have explored the inherent inclination towards novelty, which is further supported by neuroscience experiments [28, 75, 76]. The field of intrinsically motivated open-ended learning (IMOL [88]) tackles the problem of developing agents who aim to improve their capabilities to interact with the environment without any specific assigned task.

More precisely, Intrinsic Motivations (IMs [5, 69]) are a class of self-generated signals that have been used to provide robots with autonomous guidance for several different processes, from state-and-action space exploration [9, 31], to the autonomous discovery, selection and learning of multiple goals [16, 25, 87]. In general, IMs guide the agent in acquiring new knowledge independently (or even in the absence) of any assigned task. This knowledge will then be available to the system to solve user-assigned tasks [93] or as a scaffolding to acquire new knowledge cumulatively [30, 99] (similarly to what has been called curriculum learning [11]).

The option framework has been combined with IMs and "curiosity-driven" approaches to drive option learning [99] and option discovery [56, 66, 89, 90]. In the hierarchical RL [6] setting, where agents must chunk together different options to achieve complex tasks properly, IMs have been used to foster sub-task discovery [50, 63] and exploration [9]. Autonomously learning and combining different skills is a crucial problem for agents acting in complex environments, where task solving consists of reaching several (possibly unknown) intermediate sub-tasks that are dependent on each other. An increasing number of works are tackling this problem [16, 70, 78], most focused on low-level, sub-symbolic policy learning [4], in turn combined in a hierarchical manner using some sort of meta-policy [103]. While promising, these approaches necessarily face the problem of exploration, which becomes slower and less efficient as the space of states and actions increases.

In contrast to sub-symbolic methods, symbolic approaches and planning [61, 81] enable the use of higher-level objects (referred to as symbols), resulting in quicker execution, facilitating the composition of complex sub-task sequences, and making its internal knowledge interpretable, provided that the high-level representation of the planning domain is appropriately defined in advance. Planning requires prior knowledge of the world in which the agent operates, the range of available actions, and the consequences that follow from taking those actions. The need to be provided with an *ad-hoc* symbolic representation of the environment limits the utilization of high-level planning for artificial agents in unknown or highly unstructured settings, where the acquisition of new knowledge and new skills is a progressive result of the agent's autonomous

exploration of the environment. However, some works tried to improve classical planning with autonomous model learning [24], the possibility to add new symbols to the symbolic representation supported by the human [58] and adding a cognitive layer to manage an intermediate representation [77].

Recently, some ideas have appeared in the literature proposing methodologies for integrating sub-symbolic and symbolic approaches, or more generally, low-level and high-level modules [32]. On the one hand, some works tried to reconcile deep learning for planning [33, 34], goal recognition [1] and synthesizing symbolic representation of the domain [2, 3]. On the other hand, the integration has also been performed through a specific algorithm designed to produce an automated symbolic abstraction of the low-level information acquired by an exploring agent [51] in terms of a high-level planning representation such as the PDDL formalism [37], which explicitly describes the context necessary to execute an action on the current state (i.e., the *preconditions* and the *effects*) making use of symbols. This algorithm has been used as a module in the architecture of IMPACT [65, 66], the system which will be discussed in Chapter 3, integrating abstraction, planning and intrinsic motivations. Since IMPACT includes the algorithm defined by Konidaris [51] and the IMs-driven system GRAIL [87], they are analyzed in the next sections.

2.2.1 *From Skills To Symbols*

The algorithm presented in this section enables the conversion of unprocessed data obtained by an agent into an abstract symbolic representation. The abstraction procedure, from here AP, has been formalized by Konidaris [51] and used in the systems discussed in Chapter 3, where the deterministic version is used, and Chapter 4, where the probabilistic version is adopted. The thesis refers to the first version as AP1 and the second one as AP2. The pipeline of AP1 and AP2 is almost the same, while their adopted data structures differ. To aid comprehension of the algorithm and the common workflow between AP1 and AP2, the next section introduces relevant definitions. The algorithm is

subsequently presented with reference to AP1, followed by an explanation of AP2.

2.2.1.1 *Definitions and properties*

To understand the abstraction algorithm two fundamental concepts (representing together the option's *characterizing symbols* or *characterizing set*) are defined: the *Precondition* and the *Image*. First, *precondition*

$$Pre(o) = \sigma_{I_o} \tag{2.13}$$

is the symbol naming the initiation set of the option o. Then, the *Image*

$$Im(X, o) = \{s' | \exists s \in X, P(s'|s, o) > 0\} \tag{2.14}$$

is the set of possible resulting states executing option o from a state $s \in X \subseteq S$. To perform symbolic planning, it is essential to find a way to symbolically compute the image operator, which could be an expensive operation. To go beyond this difficulty and allow an effective translation from the continuous low-level domain to the symbolic high-level domain, it is necessary to convert the general options class in a subclass called *abstract subgoal options*, as they are characterized by simplicity and regularity in the image operators that lead themselves to a compact and concise representation.

A *subgoal* is typically defined as a compact set of states; generally, an option's policy is designed to lead the agent to that set. A *subgoal option* is defined as an option whose set of states in which the agent has a non-zero probability of arrival after execution, does not depend on where the option starts. To identify the subgoal, it is introduced the *effect set*

$$Effect(o) = \{s' | \exists s \in I_o \text{ s.t. } P(s'|s, o) > 0\}, \tag{2.15}$$

as the set of states which can be reached after the execution of o from anywhere it can be launched (it can be seen as $Effect(o) = \cup_X Im(X, o)$) and its corresponding *effect symbol*

$$Eff(o) = \sigma_{Effect(o)}. \tag{2.16}$$

As anticipated before, given the possible complexity of computing the *image* of an option, it would be preferable to use the *effect* as a substitute for it. This is possible in two different conditions:

- **Weak Subgoal Condition**: this condition holds for an option o when $(Effect(o) \subseteq I_{o_i}) = (Im(X, o) \subseteq I_{o_i})$ for all start sets $X \subseteq I_o$ and some options o_i. This condition states that substituting the effect set for the image, which in fact, widens the outcome of the image operator to match the effect set, never alters the feasibility of a plan.

- **Strong Subgoal Condition**: this condition holds when the effect set and the outcome of the image operator are always equal: $Effect(o) = Im(X, o)$, for all start sets $X \subseteq I_o$. This condition is much stronger than the previous one, as it entails *strict equality* between the option's effect set and the image operator.

Although weak and strong subgoal option conditions could seem restrictive, they could occur in the case of robots using a precise feedback controller. However, for their definitions, some options could not fulfil these properties. It is the case of the option *walk through the door you are facing* (see Figure 2.3). It is straightforward that the room in which the agent will find itself after the execution of the option will strongly depend on the state where the action started. Using the subgoal options significantly simplifies the representation of an option o_i in relation to the execution of the successive options o_j, as only the symbolic expression in the form $Effect(o_i) \subseteq I_{o_j}$ must be evaluated, for each pair of options o_i and o_j.

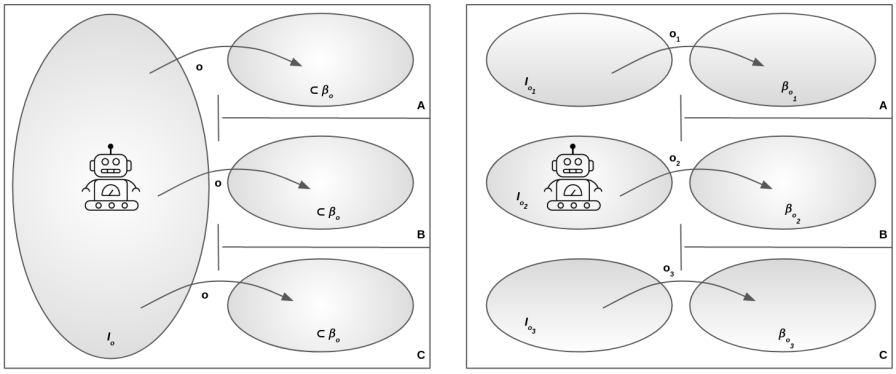


Figure 2.3: The picture depicts the initiation set and the effect set of different options definitions. On the left, the option *walk through the door you are facing*, moving the agent in the room in front of him. On the right, the option is split into three options: *go to room A*, *go to room B* and *go to room C*, respecting the *subgoal option* condition.

It is worth noticing that the *subgoal option* considers the change of all the low-level aspects when an agent is likely to modify only a part of the environment. For this reason, it is necessary to introduce the definition of *abstract subgoal option*. An abstract subgoal option o is characterized by a list of indices of the low-level variables, called *mask*, which are changed with the execution of o, without modifying other variables. In addition, the changing variables' values do not depend on their initial value.

Consequently, part of state space remains unchanged and another changes. To permit the change in a specific part of the state, it is introduced the *projection* operator

$$Project(X, v) = \{s | \exists x \in X, s[i] = x[i], \forall i \notin v\}. \tag{2.17}$$

In this case, the projection removes the restriction over the variables of v on the set of states X. Therefore, the operator removing the boundaries on the state variables belonging to $mask(o)$ is the *remainder*

$$Remainder(X, o) = Project(X, mask(o)). \tag{2.18}$$

Assuming that the value of the variables of $mask(o)$ after the execution of o is independent of the state where the option started, it is possible to define its *image* operator as follows:

$$Image(X, o) = Effect(o) \cap Remainder(X, o). \tag{2.19}$$

Finally, it is possible to generalize the weak and strong subgoal conditions for the abstract subgoal options:

- **Weak Abstract Subgoal Condition**: this condition holds for an option o when $(Effect(o) \cap Remainder(X, o) \subseteq I_{o_i}) = Image(X, o) \subseteq I_{o_i}$ for all start sets $X \subseteq I_o$ and options o_i.
- **Strong Abstract Subgoal Condition**: this condition holds if $(Effect(o) \cap Remainder(X, o)) = Image(X, o)$, for all start sets $X \subseteq I_o$.

An abstract representation enabling planning must adhere at least to the abstract subgoal condition, having symbols to identify the precondition and

Algorithm 1 Abstraction algorithm

```
1: procedure ABSTRACTION(data_experience)
2:     partitions ← Partition_Data(data_experience) // Partitioning data collected executing options
3:     for partition in partitions do // Calculating initiation and effect sets for each partition
4:         partition.characterizing_sets ← Calculate_Characterizing_Sets(partition)
5:     end for
6:     factors ← Calculate_Factors(partitions) // Factors calculus
7:     symbols ← Calculate_Symbols(partitions, factors) // Generating symbols
8:     operators ← {}
9:     for partition in partitions do // Calculating initiation and effect sets
10:        operator = Calculate_Operator(partition, symbols)
11:        operators.Append(operator)
12:    end for
13:    domain ← Generate_PDDL(symbols, operators) // Converting operators into a (P)PDDL domain
14:    return domain
15: end procedure
```

effect of each option, but also the symbols to define all the possible output of the remainder operator that could occur computing the image.

Now, it is possible to define the steps necessary to create the symbolic representation, as reported in the pseudo-algorithm 1:

1. *partitioning* options, if the options are not abstract subgoal options (line 2, covered in the previous paragraphs);
2. creation of the classifiers that empower the *characterizing sets* (lines 3-5);
3. computing *factors*, low-level variables changing always together (line 6);
4. building the vocabulary of the *symbols* (line 7);
5. building the *PDDL operators* (lines 8-12).

In the next sections, the abstraction procedure for deterministic planning (AP1) is briefly described; for further details, it is possible to find the full version formalized by Konidaris [51]. Then, some notes on the probabilistic version (AP2), following the same steps for the abstraction but using *probabilistic distributions* to represent symbols and option operators, will be discussed.

2.2.1.2 *The algorithm for deterministic planning (AP1)*

BUILDING THE CLASSIFIERS As anticipated in section 2.1.2, the semantic of a symbol σ_Z is decided by running a test on the set of low-level state variables that represent σ_Z's grounding set. This operation is, in all respect, a classification. In machine learning and statistics, classification is the problem of identifying to which of a set of categories (sub-populations) a new observation belongs, based on a training set of data containing observations (or instances) whose category membership is known. An algorithm that implements classification, especially in a concrete implementation, is known as a *classifier*. In this case, if an agent can determine whether or not an option can be run at a particular state, then it can simply learn its *initiation set* by interacting with the domain and collecting data of the form $(s, I_o(s))$ for each option, and using it as training data for the *Initiation Set Classifier*. Similarly, it can learn the effect set by simply executing the option and collecting transition data of the form (s, o, s'), and using the s' data as input for the *Effect Set classifier*.

SYNTHESIS OF THE FACTORS *Factors* are fundamental elements of the abstraction process. The factorization process represents partitioning the whole set of low-level variables into a number of *subsets containing all the state variables that are modified by the same set of options*. The algorithm needs two other operators: $options(f_j)$, returning the set of options that modifies the state variables contained in f_j, and $modifies(s_i)$, returning the set of options that modify s_i, considering a low-level state $s = [s_1, \ldots, s_n]$[1]. The process of *factorization* consists of a loop over the state variables s_i with $i \in \{1, \ldots, n\}$ in which:

- if $\exists f_j \in F$ s.t. $options(f_j) = modifies(s_i)$, then $f_j \leftarrow f_j \cup s_i$, adding s_i to the existing factor f_j;
- otherwise, $F \leftarrow F \cup \{s_i\}$, the variable s_i is used to create a new factor f_j.

1 In this paragraph a slight abuse of notation is adopted, using the subscript of the state to identify a specific variable. In the rest of the document, we always use the subscript to refer to a specific state.

SYNTHESIS OF THE SYMBOLS The purpose of this step is to create all the necessary symbols $\Sigma = \{\sigma_1, \ldots, \sigma_n\}$ to represent preconditions and effects of the final PDDL operators correctly. Also, this step is split into two parts:

- First, the procedure iterates over the options and, starting from each option's factors $f \leftarrow factors(o_i)$ and effects $e \leftarrow Effect(o_i)$ sets, synthesizes all the symbols by iterating over all the option's factors. If the factor f_i obeys the *independent condition*

$$f_i \in f \text{ s.t. } Project(e, f_i) \cap Project(e, f \setminus f_i) = e, \tag{2.20}$$

  a symbol $\sigma_{Project(e, f \setminus f_i)}$ is generated and added to Σ. After creating the symbol, the next iteration is calculated over the remaining factors $f \leftarrow f \setminus f_i$ and setting $e \leftarrow Project(e, f_i)$.

- Then, when all the symbols have been generated, for each subset of the remaining factors $f_r \subseteq factors_r(o_i)$, after projecting out the independent factors, it is created a symbol with classifier $Project(Effect_r(o_i), f_r)$, with $Effect_r(o_i)$ denoting the set remaining after projecting out all the independent factors from $Effect(o_i)$.

SYNTHESIS OF THE PDDL OPERATORS At this point, the set of symbols Σ describing the environment is ready to be used for defining the PDDL operators. These symbols are used to describe the precondition and effect of each option. The *effects* of option o_i are of three types:

- the symbols created through the $Effect(o_i)$. This case includes every symbol σ_i synthesized as a direct (positive) effect of option o_i;

- the symbols for which the entire grounding set $A \subseteq I_{o_i}$ and for which the set $factors(A) \subseteq factors(o_i)$; these symbols are added as negative effects of the option o_i. This case includes every symbol σ_j entirely defined over state variables that are in $mask(o_i)$ and that are therefore overwritten by o_i's execution;

- the symbols whose entire grounding set $B \subseteq I_{o_i}$ such that the set $f_{bi} = factors(B) \cap factors(o_i) \neq \emptyset$ and $factors(B) \nsubseteq factors(o_i)$; these symbols

are added to the negative effects. Moreover, a new symbol with a grounding set equal to $Project(B, f_{bi})$ is added to the positive effects. This case includes every symbol σ_k defined over state variables that partially overlap the variables in $mask(o_i)$.

Instead, the preconditions, representing the conditions to execute the option successfully, are computed as follows. The key operation of this step entails the generation of the powerset of Σ, which contains all the synthesized symbols; each generated subset Σ_c will undergo a series of conditions which, if verified, will allow the insertion of Σ_c as a proper precondition symbol for o_i.

The conditions that the subset Σ_c has to verify to represent a valid *precondition* are the following:

- the union of all the factors associated with the symbols in Σ_c ($\bigcup_{\sigma \in \Sigma_c}$) must be contained in $factors(I_{o_i})$, where $factors(I_{o_i})$ is the set containing all the factors related to the state variables on which o_i's initiation set I_{o_i} is defined;

- the intersection of the grounding sets of the symbols in Σ_c must be contained in the grounding set of I_{o_i}. This condition guarantees that the symbols in Σ_c have semantics compatible with I_{o_i};

- the factors in Σ_c must not mutually overlap to avoid incompatibilities among symbols. In fact, our operator effects model ensures that no two propositions with grounding symbols defined over the same factor can be true simultaneously.

Following these steps, the algorithm produces a valid deterministic PDDL domain exploitable for planning.

2.2.1.3 *The algorithm for probabilistic planning (AP2)*

The abstraction procedure AP2 differs from AP1 in using *probability distributions* for planning and symbols. In this representation, the symbol becomes a *distributional symbol* σ_Z which is the naming associated with a distribution Z over states. Consequently, the grounding distribution of a distributional symbol can be learned by performing a *density estimation*.

In this case, it is necessary to review some of the definitions using distributions:

- starting from a distribution Z and an option o, the corresponding *probabilistic image* is computed as

$$Image(Z, o) = \frac{\int_S P(s'|s, o)Z(s)P(s \in I_o)\, ds}{\int_S Z(s)P(s \in I_o)\, ds}, \tag{2.21}$$

  returning a distribution of states where the agent could find itself executing o from a state in Z.

- a *conditional symbol* σ_C is the naming of a function $P(C(s) = True)$, which is a *probabilistic classifier*, returning the probability that the condition C holds at state $s \in S$.

- the *probabilistic precondition* is the conditional symbol related to the initiation set I_o of the option o defined as $Pre(o) = \sigma_{P(s \in I_o)}$.

- the *effect distribution* of an option o is the distribution over states where the agent could be after performing o from anywhere $Effect(o) = P(s'|o)$ and the effect symbol is the distributional symbol naming the effect distribution.

Passing from $Image$ to $Effect$ operator would enable probabilistic planning. As in the deterministic case, this is possible when a *strong probabilistic subgoal condition* or *weak probabilistic subgoal condition* holds. The first case occurs when the image operator and effect distribution are equal:

$$Image(X, o) = Effect(o), \tag{2.22}$$

while the second one holds if for any starting distribution Z_o and option o_j:

$$\int Image(Z_o, o)(s)P(s \in I_{o_j})ds = \int Effect(o)(s)P(s \in I_{o_j})ds. \tag{2.23}$$

As anticipated in (2.19) for the deterministic case, in the probabilistic one when an option modifies only some state variables, the probabilistic image operator can be defined as

$$Image(X, o) = Effect(o)X_r, \tag{2.24}$$

where $Effect(o)$ is a distribution only over the variables in the mask, $X_r = \int X(s)db$ and $b = mask(o)$. It is worth noticing that this multiplication assumes statistical independence between the variables after the execution.

Finally, the probabilistic case also assumes the possibility of describing $Effect(o)$ with multiple outcomes as single mixture distribution when they have the same mask, modelled individually when they do not. The resulting option o_i will have a single common precondition $Pre(o_i)$ and a set of effect outcomes $Eff^j(o_i)$, having its own mask and probability p_j of occurring. The options, if necessary, are partitioned statistically using a specific algorithm to find partitioning C such that

$$P(s'|o,s) = P(s'|o,C(s)), \tag{2.25}$$

finding a partition C such that s' becomes independent of the value of s if $s \in C$. In the next paragraphs, the main steps for building the probabilistic domain definition of the environment are discussed. The selected domain format is the PPDDL (Probabilistic Planning and Domain Definition Language), describing the environment with a set of symbols Σ and a set of actions $\mathcal{A}$ in which each action is described as

$$\alpha_i = (precond_i, \{(\rho_1, effect^+_{i1}, effect^-_{i1}), \ldots, (\rho_k, effect^+_{ik}, effect^-_{ik})\}), \tag{2.26}$$

given its peculiar multiple outcomes.

SYNTHESIS OF THE FACTORS AND SYMBOLS The factors are broadly synthesized following the procedure discussed in the deterministic case of section 2.2.1.2. Instead, to generate the symbols Σ it is necessary to consider a different definition of independence applicable to distributions. Indeed, the independence of the factor f_s holds in the effect distribution $Effect(o_i)$ iff:

$$Effect(o_i) = \left[\int Effect(o_i)df_s\right] \times \left[\int Effect(o_i)d\hat{f}_s\right] \tag{2.27}$$

where $\hat{f}_s = factors(o_i) \setminus f_s$, corresponding to the statistical independence of variables f_s and $\hat{f}_s$ in the joint distribution $Effect(o_i)$. Consequently, the symbols are generated in two cases as before:

- a symbol with grounding distribution $\int Effect(o_i)d\hat{f}_s$ is synthesized for each option o_i and factor f_s independent in $Effect(o_i)$;

- a symbol with ground distribution $\int Effect_r(o_i)df_r$ for each set of factors $f_r \subseteq factors_r(o_i)$;

considering that $Effect_r(o_i)$ defines the distribution of the effect resulting after integrating out the independent factors from $Effect(o_i)$ and $factors_r(o_i)$ the remaining factors.

SYNTHESIS OF THE PPDDL OPERATORS For the probabilistic case, the abstract representation adopted is an extension of PDDL, which is the PPDDL (Probabilistic Planning and Domain Definition Language) [105]. PPDDL operators, following the format (2.26), assume the possibility of having multiple effects. For this reason, for each option o_i with possible outcomes $Effect^j(o_i)$, the consequent effects are composed by:

- every propositional symbol having grounding distribution for each factor independent in $Effect^j(o_j)$ and, when necessary, $Effect^j_r(o_i)$ are set to true;
- all propositional symbols $\{\sigma_j | factor(\sigma_j) \subseteq factors(o_i)\}$ with grounding in I_{o_i} are set to false;
- all true propositional symbols σ_j having factors f_{ij} in common with factors of o_i without being its subset and which could ground into states belonging to I_{o_i} are set to false. Then, for each σ_j it is set to true the predicate integrating out f_{ij}.

Each of these outcomes is collected with probability p_j. Finally, preconditions follow the guidelines discussed in section 2.2.1.2, completing the elements necessary to generate the PPDDL representation.

2.2.2 *GRAIL: Goal-Discovering Robotic Architecture for Intrinsically-Motivated Learning*

GRAIL [83, 85, 87] is a software architecture endowed with intrinsically-motivated learning capabilities to implement autonomous agents. By integrating GRAIL

architecture into robotic systems, agents gain the ability to explore their surroundings, acquire new skills, and adapt their behavior based on intrinsic motivations. More precisely, it is able to discover its own goals autonomously, select them based on a self-generated competence-based intrinsic motivations (CB-IMs) signal, select the proper computational resources to learn the goal-related skills, and autonomously recognise the achievement of a goal. It is composed of 4-levels reflecting its functionalities:

1. **goal-discovery** mechanism, discovering new interesting events and storing them as goals,
2. **goal-selector**, selecting the goals to achieve based on CB-IMs,
3. **expert-selector**, selecting the expert that has to be used to train the policy for the selected goal,
4. **experts**, controlling the robot and learning the skills to achieve the goals.

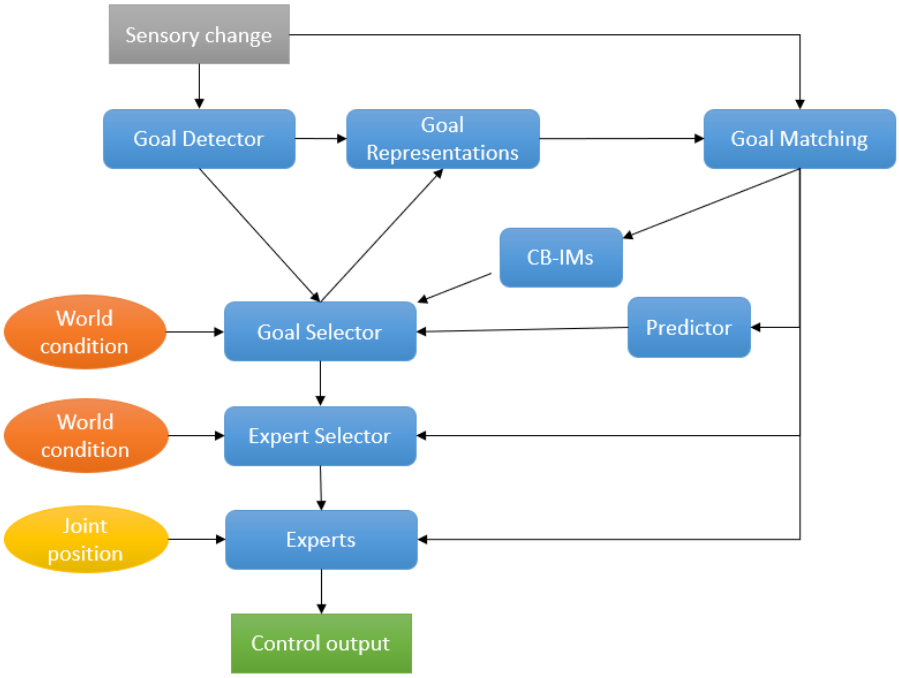


Figure 2.4: GRAIL architecture with main components.

Through its four modules, GRAIL implements a complete open-ended learning system for a robotic environment, discovering how events and learning how to

reproduce them. In the next paragraphs, the modules are described briefly; for further details, see Santucci's work [87].

GOAL DISCOVERY Its first level has the functionality of recognizing variations in the environment and storing them as goals. Given that these changes cannot be foreseen in advance and considered at design time, GRAIL demonstrates remarkable flexibility and robustness in its activities. In addition, it is also in charge of producing a *goal-matching* signal when the goal is achieved and a learning signal for the whole system. This mechanism acquires a visual input highlighting the variations in the environment, applying a subtraction between two consecutive images producing an image with pixels with value 0 (black) where no changes occurred and value 1 (white) where something changed. The resulting image is the representation of an *event*, which is stored in the explicit representation map (ER-M). Then, the event is processed by a winner-take-all (WTA) competitive network [22] returning an implicit representation vector (IR-V), an abstract event representation. Summarising, this module learns through two neural nets 1) a mapping of the input event to IR-V vector and 2) IR-V vector to the event stored in ER-M.

GOAL SELECTOR The goal-selector consists of N units, each connected to a fixed IR-V. It selects, using a softmax, the goal on which performing experiments to improve the competence of the skills reaching that goal. In the softmax, the probability of selecting the unit k at time t is defined as

$$p_k = \frac{exp\left(\frac{Q_k}{\tau}\right)}{\sum_{i=0}^{n} exp\left(\frac{Q_n}{\tau}\right)}, \tag{2.28}$$

where Q_k is the value of unit k (updated using an exponential moving average $Q_k^t = Q_k^{t-1} + \alpha(ir - Q_k^{t-1})$ with the intrinsic reinforcement ir) and τ the softmax temperature.

The activation of each unit is defined by an exponential moving average of the intrinsic reinforcement, or CB-IM signal, for reaching the correlated goal. Given that the signal describes the improvement in that competence, the goal selector is more likely to choose skills with higher learning rates to respect others.

EXPERT SELECTOR While the goal-selector chooses the goal to be reached, the expert selector decides which expert has to be used to achieve the selected task. It is defined as $M \times N$ units, where each goal unit is connected to M units, each corresponding to a specific expert, given that each goal can be pursued through any expert. Also, in the expert selector, the activation of the selected unit is updated with the exponential moving average and the expert is selected using a softmax selection rule.

EXPERTS Finally, the experts are actor-critic networks encapsulating a policy to be executed using the agent's actuators, similarly to options described in section 2.1.1. The expert takes in input the raw values from the sensors and its output are the commands to be sent to the actuators, with the addition of noise. A TD reinforcement learning algorithm is used to train the expert.

GRAIL has already been integrated in simulated and real robotic agents, demonstrating its wide applicability. IMPACT is an additional use case in which GRAIL has been successfully adopted.

3

INTRINSICALLY MOTIVATED PLANNING ARCHITECTURE FOR CURIOSITY-DRIVEN ROBOTS

IMPACT (*Intrinsically Motivated Planning Architecture for Curiosity-driven roboTs*) aims at investigating the possibility of employing Artificial Intelligence (AI) techniques to increase both the cognitive and operational *autonomy* of artificial agents in general, and of robotic platforms targeted at the space domain in particular. The idea is based on the creation of a virtuous loop in which the agent increases its learned competence and knowledge through a direct interaction with the real environment, and then exploits the autonomously acquired knowledge to execute activities of increasing complexity. This process is cumulative and virtually *open-ended* [5, 27] since the information and abilities acquired up to a certain time are employed to further increase the agent's knowledge on the application domain, as well as the skills to adequately operate in it. This self-induced tendency towards autonomously learning new skills, based on *intrinsic motivations* (IM) [5, 41, 82], will enable the agent to face situations and solve problems not foreseeable when the agent is designed and implemented, especially because of the limited knowledge on the environment the agent will operate in.

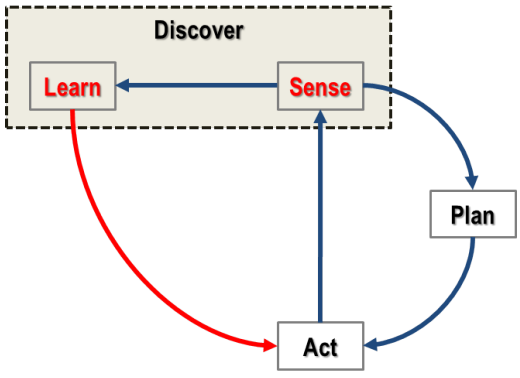


Figure 3.1: The *Discover-Plan-Act* cycle, extending the known SPA paradigm by adding the *Learn* capability between the *Sense* and the *Act* step.

The IMPACT idea proposes a possible extension of the well-known robot control architecture [17, 35, 46] formed by *Decisional*, *Executive* and *Functional* layers, commonly adopted within autonomous robotics to support the *Sense-Plan-Act* (SPA) autonomous deliberation and execution cycle. The proposed architecture realizes a *Discover-Plan-Act* (DPA) cycle (see Figure 3.1) which directly extends the SPA cycle with a more general open-ended learning step (*Discover*) directed to acquire new knowledge from the external environment. In particular, the innovative aspect of our contribution is the addition of an autonomous learning capability in the three-layered architecture, implementing the following features:

1. Autonomous learning of new *skills* based on self-generated goals driven by intrinsic motivations (intrinsic goals) [23, 87];

2. Automatic *abstraction* of the newly acquired skills [51, 65], from a low-level (sub-symbolic) to a high-level symbolic representation, e.g. expressed in *Planning Domain Definition Language – PDDL* [57]);

3. Autonomous enrichment of the planning domain by adding knowledge on new states and operators expressed through the high-level generated symbols [61].

This DPA architecture was integrated with a simulated robot platform implemented using the Gazebo Robot Simulator [48] [1] used to test the architecture. The test in particular involved a robot arm scenario inspired by a typical space exploration mission involving a rover. The scenario demonstrates the ability of the architecture to discover new ways to interact with the environment, abstract a high-level representation of the newly acquired capabilities, and integrate them in its planning domain.

The rest of the chapter is organized as follows. Section 3.1 provides a high level overview of the IMPACT framework concepts. Section 3.2 evaluates its performances in the test scenario. Finally, Section 3.3 provides some conclusions and discusses possible directions of future work.

1 http://gazebosim.org/

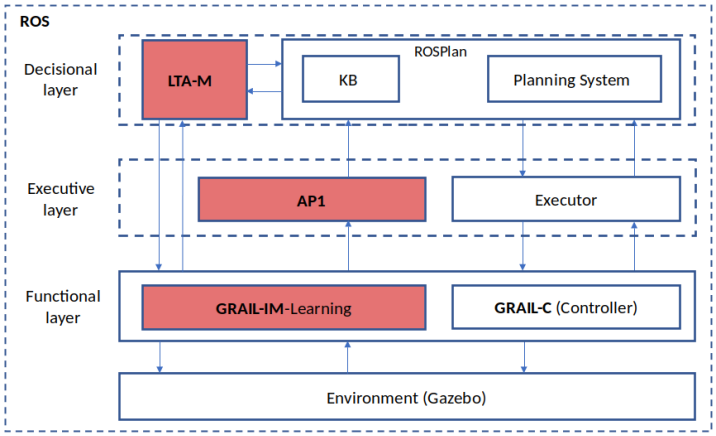


Figure 3.2: IMPACT high-level software architecture. The red blocks are novel elements implemented to actually extend the three-layered architecture.

3.1 THE IMPACT ARCHITECTURE

Figure 3.2 shows the high-level software architecture of the IMPACT framework. The framework is based on the *three-layer* robot control architecture [17, 35, 46] commonly accepted in autonomous robotics to support the *Sense-Plan-Act* (SPA) autonomous deliberation and execution paradigm. In this architecture, the *decisional* layer implements high-level planning, plan execution, and (re)planning capabilities; the *executive* layer controls and coordinates the execution of the functions distributed in the software, according to the requirements of the high-level tasks that have to be executed; lastly, the *functional* layer implements all the basic, built-in robot sensing, perception, and actuation capabilities. With respect to the classical structure of the architecture, Figure 3.2 presents the new blocks representing the proposed extensions that implement the learning capabilities, highlighted in red. In the following subsections, each layer of the architecture containing these blocks will be described in detail.

3.1.1 *The Decisional Layer*

Within the *classical* framework of the three-layered architecture, the decisional layer contains a task planner that generates a sequence of operations whose execution reaches a *goal* provided by the users. IMPACT tackled the main

challenge of integrating this planning capabilities with autonomous, intrinsically motivated learning (IM-learning) algorithms for increasing the number of the robot's skills, and the automatic extension of the related high-level knowledge base. It is worth noting that this novel vision of the three-layered architecture can be seen as one of the possible ways for addressing the *long-term autonomy* (LTA) problem for robotic systems [52]. In general, LTA can be seen as: (i) the ability of a robotic system to perform reliable operations for long periods of time under changing and unpredictable environmental conditions; (ii) the capability of autonomously increasing knowledge about the working environment. Within the previous LTA interpretation, the integration of IM-based learning capabilities is one of the possible "enablers" for long-term robot autonomy.

As Figure 3.2 shows, the decisional layer is composed of mainly two modules: (i) the ROSPlan system (proposed by Cashmore [21]), and (ii) the novel long-term autonomy manager (LTA-M).

ROSPLAN ROSPlan is composed of two main ROS [73] nodes: (1) the Knowledge Base (KB) and the (2) Planning System (PS), and implements the above mentioned *Sense-Plan-Act* (SPA) autonomous deliberation and execution paradigm (such cycle is named *Knowledge-gathering*, *Planning*, and *Dispatch* by Cashmore [21], but it substantially coincides with SPA) that traditionally characterizes autonomous robotics. In particular, the PS realizes both the automated planning and dispatching capabilities, while the KB is a collection of interfaces intended to store the up-to-date high-level model of the environment. *Planning* is the well-known process of generating a sequence of actions (as instances of PDDL operators, to be described later) to achieve a given goal from an initial state, whereas *dispatching* controls and coordinates the execution of the functions distributed over the various functional level modules according to the task requirements.

To summarize, the PS: (i) synthesizes the high-level PDDL representation $\mathcal{D}$ of the initial low-level state leveraging the data stored in the KB, thus creating a PDDL problem instance $\mathcal{P}$; (ii) passes the PDDL problem instance to the planner; (iii) dispatches each action of the resulting plan ω, and decides when to reformulate and *re-plan*.

ROSPlan realizes the dispatching process by selecting the activities belonging to the current plan one-by-one, and passing them to the *Executive layer* module, which mediates between the decisional and the functional layer, activating or deactivating the reactive functions according to the planner's specification. In the typical SPA cycle, the execution process can either terminate with a success, in which case the dispatching passes to the next activity if one exists, or a failure, in which case a re-planning process is triggered by the LTA-M based on a reformulation of the problem. As opposed to this typical SPA cycle, when an execution failure is returned our DPA architecture allows the LTA-M to possibly command a new *learning process*, in addition (or prior) to commanding the re-planning. An example of learning process triggered by the LTA-M will be presented in Section 3.2, in relation to the robotic-arm scenario.

LTA-M The LTA-M represents the architecture component that provides a set of strategies to coordinate the achievement of both *extrinsic* and *intrinsic goals*, i.e. respectively the mission goals coming from Mission Control and the learning goals generated by curiosity driven behaviors, thus realizing the overall system's long-term autonomy behavior. In short, the LTA-M is a novel component that decides when the system will operate in the ordinary SPA execution mode, exchanging information with the ROSPlan module, and when it will operate in the *Discover-Plan-Act* (DPA) execution mode, exchanging information with the GRAIL-IM learning module.

As it will be shown, the DPA cycle of the IMPACT architecture represents a significant extension of the SPA cycle, introducing an open-ended learning step (*Discover*) that basically implements the process of autonomous enrichment of the planning domain by adding knowledge on new states and operators expressed in high-level symbols.

3.1.2 *The Functional Layer*

The functional layer (called *skill layer* by Bonasso [17]) has access to the system sensors and actuators and provides reactive behavior which is robust even under

environmental disturbances, for example with the help of closed-loop control. In the IMPACT architecture the functional layer has a twofold role (Figure 3.2): (1) it provides a set of stable controllers to operate in the environment (GRAIL-C component), and (2) it guides the autonomous learning of new skills on the basis of self-generated goals (GRAIL-IM-Learning component).

These two functions are guaranteed to implement a version of the GRAIL system [87], modified to take into consideration contextual features [84]. GRAIL leverages IMs to autonomously discover interesting states to be stored as possible goals. Moreover, it uses competence-based intrinsic motivations (CB-IMs; [86]) to select them. Goals are then used to drive the autonomous learning of the skills to be achieved. These skills are stored in data structures called *experts*. GRAIL is in itself a hierarchical system with different components: (i) a *goal-discovery* layer that recognizes relevant (new or unexpected) changes in the environment and stores the resulting states as possible goals; (ii) a *goal selector* that selects a goal, the skill related to which is trained, on the basis of the context and of CB-IMs (CB-IMs lead to selecting the goal whose skill has the highest competence improvement rate); (iii) a *goal-matching* component that autonomously recognizes the achievement of the selected goal; (iv) the context-dependent *experts*, each storing the skill competence associated to a different goal; (v) a *predictor*, used to both assess the current skill competence on a specific goal (given a context) and to compute the CB-IMs signal used for goal-skill selection. These components constitute the GRAIL-IM-Learning unit in the functional layer, while GRAIL-C stores the acquired skills (the experts) resulting from the learning process.

GRAIL can be considered as a general architecture, a sort of blueprint, to perform autonomous open-ended learning. The mechanisms and functions described above can be implemented in different ways (e.g., [87, 93]) and their in-depth analysis is beyond the scope of this work. However, in sec. 3.2.4 some details are provided on how the specific GRAIL functions have been implemented within IMPACT architecture to tackle the experimental scenario. Other details about GRAIL have been discussed in the subsection 2.2.2.

3.1.3 *The Executive Layer*

The executive layer mediates between the decisional and functional layer, i.e. it activates or deactivates the reactive functions according to the deliberator's directives. It controls and coordinates the execution of such functions distributed over the various functional level modules, i.e. the *experts*, according to the task requirements. The problem of mapping symbolic abstract plans to continuous actions in real working domains is well recognised in the literature [17, 35, 46].

As shown in Figure 3.2, in our architecture the executive layer has been enriched with an abstraction subsystem[2]. The abstraction procedure executed by this subsystem will be called AP1, standing for Abstraction Procedure 1, to highlight the difference with the algorithm used in the system presented in chapter 4. This module has the task of automatically translating the newly acquired skills from a low-level sub-symbolic representation generated by the GRAIL-IM module, to a high-level symbolic representation. Indeed, the agent senses the environment through its sensors, which return low-level information only, generally represented by all the values of the continuous variables that constitute the low-level state representation. Consequently, the representation of the skills that the agent learns by direct experience with the environment will be initially represented in sub-symbolic terms, a representation not suitable for high-level reasoning. In particular, the learned skills are expressed at low-level by leveraging an *option-based* representation [102]. Each skill $o = \langle Cl(I), Cl(E) \rangle$ is characterized by two *classifiers* (e.g., implemented as *decision trees* [74]), namely the *Initiation Set* classifier $Cl(I)$ and the *Effect Set* classifier $Cl(E)$. In particular, given a skill o and a low-level state s, the $Cl(I)$ classifier is used to test whether o can start from s or not (i.e., s belongs to o's *initiation set*). Similarly, the $Cl(E)$ classifier is be used to test whether s is a possible "effect state" after executing o.

The classifiers for each learned skill o are trained on the basis of the data obtained by repeatedly executing o, and saving these data as `true` (positive case) or `false` (negative case) training instances, depending on the classifier's purposes. In particular, to build the $Cl(I)$ training dataset, as positive cases are

2 A thorough description of the abstraction algorithm is beyond the scope of this work; the interested reader may refer to subsection 2.2.1 or the original paper [51].

considered all the low-level variable values before the successful execution of the skill, and as negative cases all the low-level variable values in conditions where GRAIL tried to execute the skill but the *predictor* (see Section 3.1.2) was always zero. To build the $Cl(E)$ training dataset, as positive cases are considered all the low-level variable values following a successful execution of the skill, while as negative effect cases have been used all the low-level variable values before the execution of that skill, whether it succeeded or not (since it is known that GRAIL will not execute the skill if its goal/effect is already achieved).

To summarize, AP1 accepts in input a $\langle Cl(I), Cl(E) \rangle$ pair for each skill, and returns the complete set-theoretic PDDL representation (described below) related to the agent's currently learned low-level skill set. Every *proposition* of the returned PDDL domain corresponds to a *symbol* automatically produced by AP1 out of the effects of each skill; all the produced symbols are saved in the set Σ. As AP1 proceeds, a different classifier $Cl(\sigma_i)$ is created for each produced symbol $\sigma_i \in \Sigma$. Ultimately, the $Cl(\sigma_i)$ classifier will be used to determine σ_i's *truth value* every time it is necessary. Every symbol σ_i is associated to those subset of low-level states that determines σ_i's semantics (σ_i's *Grounding Set*).

As anticipated above, each skill learned by the agent will be abstracted into its related PDDL operator expressed in a Set-theoretic representation, whose preconditions and effects sets are subsets of the set Σ. Formally, a set-theoretic PDDL domain specification $\mathcal{D}$ is expressed in terms of a set of propositional symbols $\Sigma = \{\sigma_1, ..., \sigma_n\}$ (each associated to a grounding classifier $Cl(\sigma_i)$) and a set of operators $A = \{opt_1, ..., opt_m\}$. Each operator opt_i is described by the tuple $opt_i = \langle pre_i, eff_i^+, eff_i^- \rangle$ with $pre_i, eff_i^+, eff_i^- \subseteq \Sigma$, where pre_i contains all the propositional symbols that must be *true* in a state s to allow the execution of opt_i from s, while eff_i^+ and eff_i^- contain the propositional symbols that are respectively set to *true* or $false$ after opt_i's execution. All the other propositional symbols remain unaffected by the execution of the operator.

For example, let us consider the following operator, called `graspObject`, whose action corresponds to grasping a stone-shaped object on behalf of a robotic arm (additional information about the related PDDL domain will be given in Section 3.2):

```
(:action opt_4
    :parameters ()
    :precondition (and (Symbol_0) (Symbol_7)
    (Symbol_9))
    :effect (and (Symbol_5) (not (Symbol_7)))
)
```

The automatically synthesized symbols have the following meaning:

- $(Symbol_0)$ = "*object in sight*";
- $(Symbol_7)$ = "*object not grasped*";
- $(Symbol_9)$ = "*object not stowed*";
- $(Symbol_5)$ = "*object grasped*".

As immediately observable, the operator follows the standard set-theoretic PDDL, where $pre_i = \{Symbol_0, Symbol_7, Symbol_9\}$, $eff_i^+ = \{Symbol_5\}$, and $eff_i^- = \{Symbol_7\}$ (note that the symbol's name is automatically generated). In order to provide the reader with some information about the meaning of the symbols that populate the previous operator, the semantics of `Symbol_0` proposition in the operator's preconditions is "*object in sight*" (the object must be in sight of the agent in order for the agent to grasp it). More specifically, `Symbol_0`'s grounding classifier will return the *true* value only from the low-level states sensed by the robot as having the v_3 variable in the *true* range (see Figure 3.5: the v_3 pixel returns the image of an object within the rover's grasping distance).

Lastly, it is observed that the AP1 module has a *dual role* in comparison to the Executor; in fact, the learning process can be seen as opposite to the execution process as it maps low-level sub-symbolic representation into abstract symbolic representation.

3.2 OPERATIONAL EVALUATION

This section is dedicated to describing the behavior of our architecture within a simulated test case, called *Robotic Arm* scenario. The *Robotic Arm* test case entails

the presence of an exploration rover equipped with a gripper actuator attached to a manoeuvrable arm, trying to grasp a "vase shaped" rock whose size exceeds the max opening span of the gripper. The robot is thus not able to pick-up the rock using its basic grasping skill; however, upon a "surprising" failure (i.e., a failure occurring when performing a skill supposed to be successful), the system will automatically trigger the learning of a new skill and at the end of the learning process the robot will be able to pick-up the "vase shaped" rock. Finally, the new skill will be abstracted in terms of a new PDDL operator, and will be seamlessly added to the existing PDDL domain, ready to be used by the planner.

In this scenario, the rover is supposed to explore a region of the Mars surface divided in two zones ($zone_1$ and $zone_2$); both zones can be reached through a single waypoint ($zone_3$, see Figure 3.3). The Gazebo model of the robotic arm

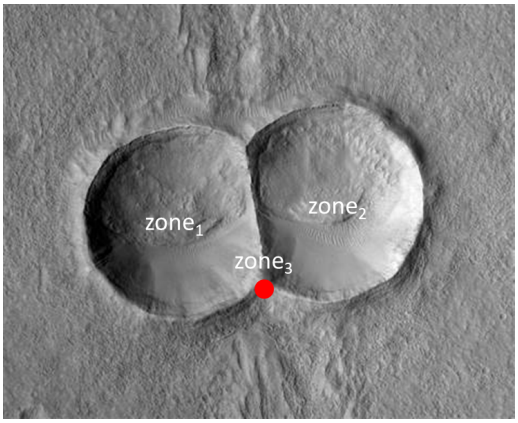


Figure 3.3: The scenario selected for the Robotic Arm use case, representing an area of the martian terrain characterized by two twin craters, that the rover should explore for sample grasping purposes. It is possible to travel from $zone_1$ to $zone_2$, and vice versa, through $zone_3$ (depicted in red).

used for this test case is a simulation of a KUKA robotic arm (called Youbot [3]). The behaviour of the simulator is basically intended to mirror the behaviour of the real Youbot (see Figure 3.4), including the physics and all the topics and exchanged messages. The Youbot arm is used to perform all the grasping activities, as well as the stow activities. In particular, the Youbot model will be used during the learning process of the new grasping activity.

3 https://github.com/islers/youbot_simulation

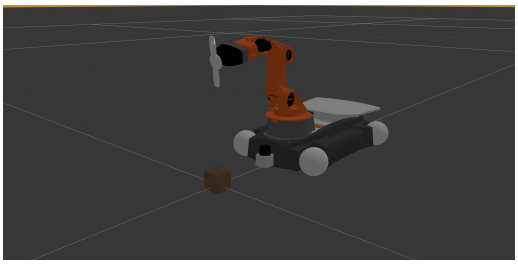


Figure 3.4: The Rover Model used in the Robotic Arm Scenario.

3.2.1 *Low-level Operators and variables description*

At the beginning of the test case, the rover is supposed to be equipped with a pre-compiled PDDL planning domain (described in Section 3.2.2), composed of the following high-level operators:

- `moveBeforeObject_1` (`opt_0`): the rover moves from $zone_3$ to $zone_1$ and places itself in front of an object, in a position suitable for grasping it;
- `moveBeforeObject_2` (`opt_1`): the rover moves from $zone_3$ to $zone_2$ and places itself in front of an object, in a position suitable for grasping it;
- `moveBackFrom_1` (`opt_2`): the rover moves from $zone_1$ to $zone_3$;
- `moveBackFrom_2` (`opt_3`): the rover moves from $zone_2$ to $zone_3$;
- `graspObject` (`opt_4`): the rover activates the robotic arm to grasp the object that is in front of it;
- `stowObject` (`opt_5`): the rover uses the robotic arm to stow a grasped object inside the rover's onboard container;
- `readyGripper` (`opt_6`): the rover resets its robotic arm in a position suitable for the next grasp (the gripper is supposed to hold no object).

This scenario is used to provide an example of autonomously synthesized "enrichment" of the current PDDL domain representation, as a consequence of an intrinsically motivated learning activity through a direct interaction (i.e., sensing + acting) with the environment. The final result will be a version of the PDDL planning domain automatically extended with a *new* grasping operator

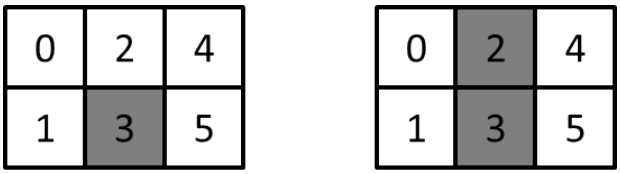


Figure 3.5: The view field of the rover is composed of 6 low-level variables (pixels) numbered from 0 to 5. The dark pixels represent that an object is in sight; the number of darkened pixels will provide some simplified information about the type of the observed object: stone-shaped object (left), vase-shaped object (right).

that allows the rover to grasp differently shaped objects in a different fashion w.r.t. the original grasping.

The complete low-level variables set S is composed of 10 variables $S = \{v_0, v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9\}$ under continuous observation on behalf of the rover, whose meaning is the following:

- $v_0, ..., v_5$: low-level variables that constitute the rover's view field. Each such variable corresponds to a single pixel, and therefore the view field will be composed of a 2×3 pixel matrix (see Figure 3.5)[4];
- v_6: low-level variable that senses the grasping of an object ($v_6 = false \rightarrow$ no object is grasped, $v_6 = true \rightarrow$ an object has been grasped and it is being held by the gripper);
- v_7: low-level variable that senses the presence of an object in the stowing bucket ($v_7 = false \rightarrow$ no object is stowed, $v_7 = true \rightarrow$ the object is being stowed);
- v_8: low-level variable that senses the rover's being $zone_1$ ($v_8 = false \rightarrow$ the rover is not in $zone_1$, $v_8 = true \rightarrow$ the rover is in $zone_1$);
- v_9: low-level variable that senses the rover's being in $zone_2$ ($v_9 = false \rightarrow$ the rover is not in $zone_2$, $v_9 = true \rightarrow$ the rover is in $zone_2$);

In general, each low-level variable can take an infinite set of real values; without loss of conceptual generality w.r.t. AP1's abstraction capabilities, it

4 In general, a camera view field is composed by a much larger array of pixels; it is used a small set without loss of generality within the scope of this work.

is however assumed that such values will be translated to boolean values depending on the values captured by the rover's sensors. For instance, if the position tracking system (returning the x and y coordinates of the rover) detects that the rover is *sufficiently* close to the center of $zone_1$, then the variable v_8 (which can be interpreted as a "*proximity to* $zone_1$" signal) will be set within the *true* range, while the $zone_2$ proximity variable will be set within the $false$ range, and vice versa.

3.2.2 *The Initial PDDL domain*

The initial PDDL domain $\mathcal{D}$ proposed for this scenario is supposed to be automatically synthesized by means of AP1. In the following, it is provided an example of utilization of such PDDL domain, presenting a solution of a PDDL problem representing a typical mission goal that the rover is supposed to fulfill once on the martian surface. The problem $\mathcal{P}$, expressed in PDDL syntax, is the following:

```
(define (problem marsprob) (:domain RobGripper)
(:init (Symbol_1) (Symbol_4)
       (Symbol_6) (Symbol_7) (Symbol_9))
(:goal (and (Symbol_8) (Symbol_2)))
)
```

For clarity purposes, it is proposed the list of the symbols' meanings in appearance order:

- *Symbol*_1 = "*rover in* $zone_1$";
- *Symbol*_4 = "*rover not in* $zone_2$";
- *Symbol*_6 = "*object not in sight*";
- *Symbol*_7 = "*object not grasped*";
- *Symbol*_9 = "*object not stowed*":
- *Symbol*_8 = "*object is stowed*";

- *Symbol_2* = "*rover in* $zone_2$".

This problem's initial condition entails the rover starting from $zone_1$, with the object not yet visible , nor grasped or stowed. The problem's goal is stowing an object in $zone_2$.

The solution ω, obtained by submitting the problem to an off-the-shelf planner, is the following:

```
SOLUTION PLAN:
opt_2:   [moveBackFrom_1]
opt_1:   [moveBeforeObject_2]
opt_4:   [graspObject]
opt_5:   [stowObject]
```

The rover moves from $zone_1$ to $zone_3$ (`opt_2`) and then from $zone_3$ to $zone_2$ (`opt_1`) in front of the object, then grasps the object using its grasping skill (`opt_4`), then stows the object (`opt_5`).

3.2.3 *Automatically synthesizing the new PDDL domain*

Let us now suppose that, unknown to both the mission's managers and the rover's designers, the rover encounters a concave, vase-shaped object in $zone_2$ during the exploration. According to the rover's high-level logic, the object is still recognized as an interesting sample, and therefore the rover attempts to grasp it using the usual grasping skill. Let us now suppose that the object is wider than the maximum span of the gripper; *in this case, the grasping action will fail*. The failure is immediately captured by the functional layer of the architecture, and passed on to the LTA-M in the decisional layer. The LTA-M is therefore called to decide whether the failure requires a re-planning or a new learning process (see Section 3.1.1). If the LTA-M opts for the second case, the failure thus detected will trigger a new learning goal, e.g., driven by the surprise deriving from having failed the execution of a skill whose reliability was held high. The net result is that the rover will be autonomously prompted to initiate a process to learn how to grasp the new object, leveraging the GRAIL-IM module, as

described in the next section. When the learning process is terminated, a new skill will be acquired where the rover is now capable to grasp the vase-shaped object (e.g., by pinching it by its edge, or by inserting the closed gripper in the vase and then opening it, depending on the particular evolution of the learned policy).

After the learning of the new skill and before the synthesis of the new PDDL domain, a period of environment exploration or "execution rehearsal" of all the known skills (old and new) is commanded by the LTA-M, in order to acquire all the information to train the new classifiers necessary to build the new domain[5], as described in Section 3.1.3. When the LTA-M decides that the amount of training data is sufficiently high, the $Cl(I)$ and $Cl(E)$ classifiers' training is triggered anew for all the old skills as well as for the one newly acquired. As a last step, the AP1 abstraction procedure can be commanded again by the LTA-M, and the new PDDL domain is autonomously synthesized. In the remainder of this section, only the differences between the old and the new PDDL domain will be highlighted for reasons of clarity.

The first important difference between the two domain representations is the presence in the new domain of an extra symbol (i.e., `Symbol_10`) whose semantics is "Object is VASE". This symbol has been created as a consequence of the learning of the new grasping skill, and it is very important as it represents the symbolic element that discriminates between the two different classes of objects that the rover has encountered during its exploration, i.e., the *only* class of objects the rover was supposed and trained to operate on (the stone-shaped rocks), and the class of objects that the rover has incidentally discovered (the vase-shaped objects) and whose grasping had to be learned. The presence of this new symbol in the PDDL high-level representation of the domain is of great importance as it allows the planner to correctly synthesize plans that involve the grasping of either kinds of object *directly at symbolic level*. In other words, it allows the planner to produce plans that entail the utilization of the correct grasping operator, depending on the class the object to be grasped belongs to.

5 In our system, the training information is stored in the usual Attribute Relationship File Format (ARFF).

The second important difference is that in the new PDDL domain a new high-level operator is created: `graspObject_2`, assuming automatically the name `opt_7` and synthesized as follows :

```
(:action opt_7
    :parameters ()
    :precondition (and (Symbol_0) (Symbol_10)
    (Symbol_7) (Symbol_9))
    :effect (and (Symbol_5) (not (Symbol_7)))
)
```

This operator is the high-level representation of the newly learned low-level skill, and therefore it is the operator that will be used to grasp the vase-shaped objects. Most interestingly, it can be readily confirmed that the newly created `Symbol_10` ("Object is VASE") has been added to the preconditions of the `graspObject_2` operator, thus addressing the fact that, in order for the new grasping operator to be activated, the rover not only has to find itself in front of an object (i.e., `Symbol_0` must be verified), but it must also be a vase-shaped object. The new PDDL domain will be further analyzed in Section 3.2.5.

3.2.4 *Autonomous learning of new skills*

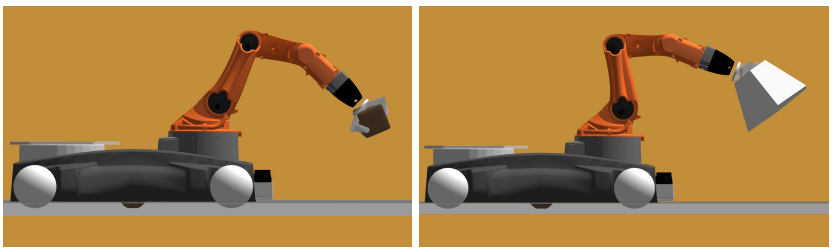


Figure 3.6: The rover, the Youbot arm and the gripper in the Robotic Arm scenario. The robot grasping the cube-shaped rock (left). The robot grasping the vase-shaped rock (right).

IMPLEMENTATION OF GRAIL IN THE EXPERIMENTAL SCENARIO. It is now briefly described the implementation of the mechanisms and functions

introduced in Section 3.1.2, and that are actually involved in the current experimental scenario.

In the Robotic Arm scenario, it is assumed that the system is already endowed with a series of different goals and with the related skills necessary to achieve them (see Section 3.2.1) from a generic starting condition (the *context*). Here, a goal is intended as a state (or an effect) the system could try to achieve. If a state s is represented as a set of n features $\{\phi_1, \phi_2 ..., \phi_n\}$, a goal can be a state where a certain feature (e.g. ϕ_2) assumes a certain value x. This section is focused on the goal related to the grasping of an object positioned in front of the robot, and therefore the *goal discovery* and *goal selection* mechanisms can be considered "switched off", so that the system is continuously selecting the same goal. In general, however, the autonomous motivational system is fully operational in the architecture, providing the goal with the intrinsic motivations that would drive the system in pursue it, even when competing with other possible tasks. Different contexts are associated to each goal stored by the system, which is thus able to compare them and recognize if a new context is encountered. In this specific implementation, the context is a set of sensory inputs the system perceived: the two different visual inputs in Figure 3.5 might constitute two different contexts when selecting an expert to perform a skill. In the particular case tackled here, it is simply assumed that the system has one expert for each context; whenever a new situation is encountered, a new context-related "branch" is created together with a new expert that is a copy of a previous expert used for the same goal. The experts control the arm and the gripper of the robot and, given the selected goal and the current context, they learn the policy necessary to achieve it. Experts are currently implemented through discrete-movement Dynamic Movement Primitives (DMPs, [45, 91]), i.e. dynamical models that can generate a movement trajectory on the basis of end-position parameters (controlling arm and gripper final postures) and shape parameters (controlling the shape of the trajectory). In particular, each controller is composed of a set of 9 DMPs, each one used to control a different joint of the robotic arm and the activity of the gripper (9 DoF). Training trials are composed of 10 attempts. For each attempt, Gaussian noise is generated and added to the parameters of the DMPs to generate new trajectories. The noise is dependent on expert performance: the higher the performance,

the smaller the noise. After each attempt, DMPs are updated through Policy Improvement Black Box (PI^{BB}, [100]), that modifies the parameters of the DMPs composing the policy on the basis of a path-integral reward function for each attempt.

When an expert is used, the representation of the related goal is pre-activated so that GRAIL is able to check the achievement of the goal through a goal matching mechanism. As described in Section 3.1.1, when a failure occurs the LTA-M can allow GRAIL to interact with the environment and try to acquire new competences. In particular, if the robot tries to grasp a bigger object ("the vase", see Figure 3.6, bottom) with its pre-designed skill, it will not be able to accomplish the associated grasping goal. This will generate a sequence of different effects: (i) a failure signal will be generated through the goal-matching function; (ii) the context in which the system is working is checked and since having the vase in front of the robot is different from having the rock, a new context is generated for the "grasping goal", together with a new expert; (iii) a novelty based intrinsic motivation signal is generated for that goal so that the system, even when more goals are present at the same time, is biased to re-select the same goal and possibly start to acquire the new skill needed in the new context; (iv) the robot will start to attempt the grasping goal, and the more the system will be able to improve its ability in generating a goal matching from the new starting condition, the more a CB-IM signal will be computed to drive the system towards focusing on the same goal and properly acquire the new skill. In particular, the CB-IMs are calculated based on the system improvement in achieving a goal: in this sense, CB-IMs can be considered as a derivative (over a certain time-window) of the system performance on a task. This formulation leads to the effect that CB-IMs are present only when they are needed: if the system is able to systematically generate the desired goal or, on the contrary, it is not able to improve its competence, the IMs will eventually "fade away" and the robot will start selecting other goals, or the control will pass again on the higher level component of architecture.

LEARNING THE NEW SKILL. Figure 3.7 shows the performance of the robot in grasping the vase-shaped rock. After a first failure that triggered the

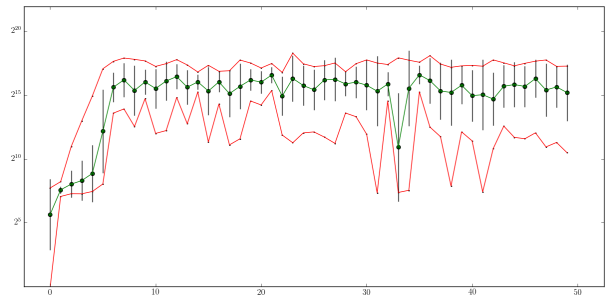


Figure 3.7: The learning curve of the new skill. Red curves indicate the performance of the best and the worse attempt (within a trial of 10 attempts). The green curve indicates the average performance within the trial, and whiskers show the standard deviation. Because the performance grows geometrically, the scale of the y axis is logarithmic.

exploration and learning processes, the system starts trying to achieve the goal from a new context (the vase-shaped rock never encountered before). Using a copy of the expert trained to work with the small rock, at the beginning the robot is not able to achieve the goal. However, after the first successes due to exploratory noise, the robot improves its competence in the first 5 trials. Nonetheless, at the beginning the noise variance over the parameters of the DMPs is high because the acquired competence is still growing: as a consequence, the robot continues to explore through the parameters and its competence improvement is growing slowly. However, after other 5 trials the system starts to get persistent improvements, resulting in higher level of rewards and better average performance over the attempts.

Note that even when the robot has reached a proper competence with the new skill (after 30 trials, see red dots in Figure 3.7) the performance still has a high variance even though the variance of parameters is low or zero. This is due to the fact that the simulation has an inner, independent noise, caused by a non-stationary gap in communication between the controller and the simulator. Although on the one hand this seems to impair a perfect learning of the skills, on the other hand it might be considered as a replication of the interaction between the controller and a real robot, where a little amount of noise in the communication can never be avoided.

3.2.5 *Planning over the new Robotic Arm PDDL Domain*

The correctness of the new autonomously synthesized domain is proven by demonstrating that the agent is now able to reach the goal of grasping the vase-shaped object in $zone_2$, again starting from $zone_1$. It is worth noticing that the previous goal of grasping and stowing the stone-shaped object is still correctly planned with the new domain. Obviously, the PDDL problem will be expressed exactly in the same terms as the one presented in Section 3.2.2, under the assumption that the vase-shaped objects can be found in $zone_2$ only.

The obtained solution ω is the following:

```
SOLUTION PLAN:
opt_2: [moveBackFrom_1]
opt_1: [moveBeforeObject_2]
opt_7: [graspObject_2]
opt_5: [stowObject]
```

Again, it can be observed that the plan is correct; the rover moves from $zone_1$ to $zone_3$ (`opt_2`), then moves from $zone_3$ to $zone_2$ in front of the object (`opt_1`), then grasps the object using the correct new operator (`opt_7`), and then the object is finally stowed (`opt_5`).

3.3 DISCUSSION AND FUTURE WORK

This chapter presents an architectural design of an extended version of the well-known three-layered architecture that typically implements the behavior of a robotic system. The proposed architecture realizes a *Discover-Plan-Act* cycle (DPA), thus enhancing the well-known *Sense-Plan-Act* (SPA) paradigm with a more general open-ended learning step (Discover). New functionalities have been added to the classical three-layered architecture: (i) a goal-discovering and skill-learning architecture (GRAIL) connected to the symbolic abstraction procedure (AP1) that creates a processing pipeline from the continuous to the symbolic environment representation; (ii) a long-term autonomy manager

(LTA-M), that strategically coordinates the achievement of both the external mission goals and the internal goals generated by curiosity driven behaviors. The complete functionalities of the system have been tested on a scenario inspired by a typical planetary rover mission.

Despite the core of the proposed architecture being the integration of an open-ended low-level learning process and a symbols abstraction process, this integration is more than the mere sum of its parts. Indeed, the composition of both previous features in a dynamic environment fosters the capability of integrating newly discovered with previously abstracted knowledge, thus boosting the environment discovery capabilities, which is an invaluable added value to long-term autonomy. Notwithstanding the relative simplicity of the experimental scenario, this chapter presents for the first time a complete and working architecture that spans from low-level autonomous exploration, discovery, and skill learning to high-level automated reasoning, thus demonstrating the capability to autonomously learn new symbols and operators by direct and autonomous interaction with the environment.

The development of open-ended learning as an enabling factor for long-term autonomy of robotic systems opens many possible branches for future work. Other current works are related to the probabilistic version of AP1, capable of capturing the stochastic nature of more general real-world phenomena. A second interesting direction of work is the definition of extended *long-term autonomy strategies*, i.e., the integration of symbolic planning and open-ended learning to increase the ability of one agent to autonomously acquire new symbolic knowledge based on previously acquired symbolic models (*bootstrap learning* [27]). As a possible extension, it is possible to define strategies to generate symbolic plans for collecting new intrinsic goals to achieve. Other strategies can be designed for planning to reach different sets of preconditions for learning the achievement of a given goal.

4

INTRINSICALLY MOTIVATED HIGH-LEVEL PLANNING FOR AGENT EXPLORATION

This chapter presents a preliminary implementation of an open-ended learning agent which, starting from a set of action primitives, is able to (i) discover options, (ii) explore the environment using them, (iii) create a PPDDL representation of the collected knowledge and (iv) plan to improve its exploration and solve the problem set by the game.

The aim of this study is to assess the potential of *abstraction* in autonomous systems and propose a new approach for planning systems, extending them with learning capabilities and behaviors driven by IMs. IMs are employed for discovering new options in a *surprise-based* manner at low-level and exploring farther states driven by *curiosity* at high-level. By the term abstraction, we simply mean mapping a certain problem space into a simpler one (e.g. converting a continuous domain into a discrete domain). In the proposed architecture, the abstraction is applied at two levels: a) passing from the *primitive action space* to the *options action space* and b) converting *low-level data* collected during the exploration into a *high-level domain representation* suitable for high-level planning, thus from raw sensors' data to a PPDDL representation.

Performing the pipeline depicted in Figure 4.1, the system creates different layers of abstraction, enriching the agent's knowledge with causal correlations between options and enabling more efficient reasoning (i.e. using classical planning). Given its characteristics, especially in the aspect of the representation of the acquired knowledge, this approach could be also interesting in the trustworthy AI field [104], to study new ways to create reliable and safe agents that expose their internal decisional process explicitly. Symbols can be seen as knowledge building blocks that can be used to search for interesting states and find new knowledge in a virtuous loop. To experience this innovative paradigm, the system has been tested in different configurations of the so-called Treasure

Game environment, a simple Gym environment presenting different latent dependencies between states.

4.1 ARCHITECTURE DESCRIPTION

As depicted in Figure 4.1, the system's architecture comprises five modules combining learning, abstraction and planning phases. The pipeline of the system performs the following phases:

1. **Option Discovery:** using a small set of primitives $A = \{a_1, a_2, ...\}$ belonging to the agent, the system combines them to create higher level actions, in this case, the *options*;

2. **Exploration:** the set of options $O = \{o_1, o_2, ...\}$ discovered in the previous step is used to explore the environment. In the meantime, the visited low-level states data are collected into two datasets: the *initiation data* ID and *transition data* TD containing, respectively, samples of states in which an option can be run and the transition between states executing it;

3. **Abstraction:** datasets ID and TD, of the visited low-level states until that moment are processed by the algorithm of abstraction, generating a PPDDL domain $\mathcal{D}$ of the knowledge collected;

4. **Planning:** the PPDDL representation $\mathcal{D}$ is used to assess whether the final goal of the game s_g can be reached with the current synthesized knowledge and a plan ω^{EX} to explore interesting areas of the domain is generated, suggested by the **Goal Selector** module.

5. The system will execute again the loop from step 1, exploiting ω^{EX} suggested by the **Goal Selector** module to improve the Option Discovery and Exploration phases.

In this setting, the agent is only endowed with a set of primitive movements $A = \{a_0, ..., a_m\}$ and the world $s \in S$ is represented by a vector $(v_0, ..., v_n)$, such that $v \in \mathcal{R}$, consisting in the low-level values of the sensors belonging to the agent.

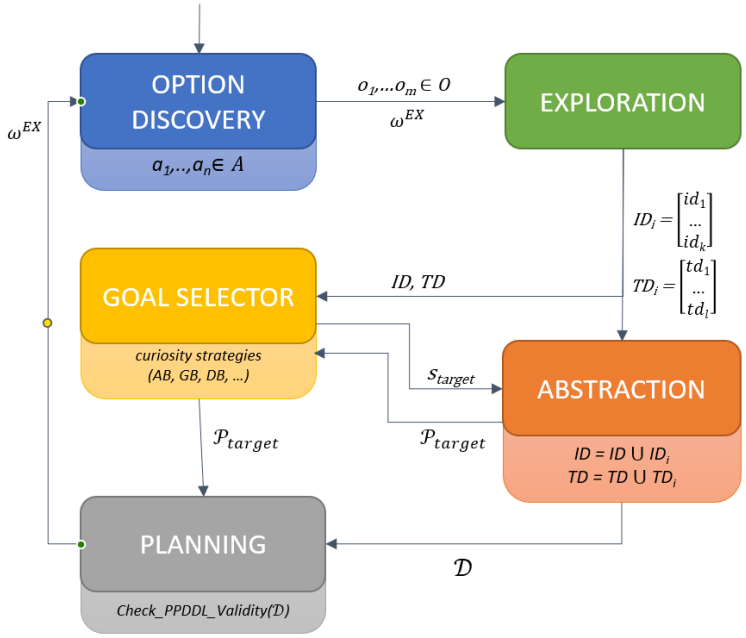


Figure 4.1: The architecture of the system. The system, equipped with five primitive actions, iteratively (1) learns some options from scratch, (2) uses the options to explore the environment and gather low-level states data, (3) creates a PPDDL representation of the collected experience, (4) plan to solve the game and (5) to improve its exploration.

Executing iteratively this framework, an abstract representation of the world emerges from the raw data collected by the agent and it is exploitable to enrich itself over time with new actions and symbolic concepts. The loop details depicted in Figure 4.1 are described by Algorithm 2. In the following subsections, the modules are explained individually, referring to the pseudocode.

4.1.1 *Option Discovery*

In this section, we will analyze the **Option Discovery** module (line 8 of Algorithm 2) in greater detail with the help of the pseudocode depicted in Algorithm 3. As anticipated previously, discovering new options is considered driven by the *surprise* of the agent in finding available new primitives during its operations. When the agent encounters a change in the availability of its

Algorithm 2 Discover-Plan-Act algorithm

procedure DISCOVER_PLAN_ACT($cycles, dpa_eps, dpa_steps, d_eps, d_steps$)
  $c \leftarrow 0$ //Cycle initialization
  $O \leftarrow \{\}$ //Option set initialization
  $ID \leftarrow \{\}$ //Initiation Data initialization
  $TD \leftarrow \{\}$ //Transition Data initialization
  $\omega^{EX} \leftarrow \{\}$ //Initially, the high-level plan is empty
  while $c < cycles$ **do** //For each cycle
    $O_new \leftarrow DISCOVER(d_eps, d_steps, \omega^{EX})$ //Learning the available options
    $O \leftarrow O \cup O_new$
    $ID_{new}, TD_{new} \leftarrow Collect_Data(dpa_eps, dpa_steps, O, \omega^{EX})$
    $ID \leftarrow ID \cup ID_{new}$
    $TD \leftarrow TD \cup TD_{new}$
    $\mathcal{D} \leftarrow Create_PPDDL(ID, TD)$
    $s_{target} \leftarrow Get_Target_State()$
    $\mathcal{P}_{target} \leftarrow Generate_PPDDL_Problem(s_{target})$
    $\omega^{EX} \leftarrow Plan(\mathcal{D}, \mathcal{P}_{target})$
    $Check_PPDDL_Validity(\mathcal{D})$
    $c \leftarrow c + 1$
  end while
end procedure

primitive abilities, it stores this event as a low-level skill that can be re-used later to explore the surrounding environment.

By executing the algorithm, the agent can discover a set of options O from scratch; such options are generated by repeatedly executing a certain primitive $a \in A$ among the available ones and collecting the produced changes in the environment. This procedure is intentionally implemented in a simplified way, given that the focus of the work is proposing an extendable symbolic knowledge architecture, reusable to reach intrinsic and extrinsic goals autonomously.

In more detail, the agent creates new options considering the following modified definition

$$o(a^p, a^t, I, \pi, \beta), \tag{4.1}$$

where a^p and a^t are primitive actions such that $a^p \neq a^t$, a^p is used by the execution of π, a^t stops the execution of π when it becomes available, π is the policy applied by the option, consisting in repeatedly executing a^p until a^p can no longer be executed or a^t becomes available, I is the set of states from which a^p can run; β is the termination condition of the option, corresponding to the

availability of the primitive a^t or to the impossibility of further executing a^p. For the sake of simplicity, the option's definition will follow the more compact syntax

$$o(a^p, a^t) \tag{4.2}$$

in the remainder of the chapter, meaning that I is the set of states in which a^p can run, β checks the following two conditions: a^t becomes available or a^p is no longer available, and π is the policy corresponding to repeatedly executing a^p until β verifies.

Algorithm 3 describes the option discovery procedure previously cited. At the beginning of each discovery episode, the plan ω^{EX} is executed to reach a new area where to start learning new options (line 7-9). Then, for a maximum number of episodes and steps, the agent saves the current state s and randomly selects a primitive a^p which can be executed in s (line 10-12). a^p is repeatedly executed towards reaching the state s' until either a^p is no longer available or new primitives beyond a^p become available. If $s \neq s'$, the procedure creates a new option $o(a^p, a^t)$ whether a new option a^t has been enabled, or $o(a^p, \{\})$ if there are no new primitives enabled. In the end, o is added to the other collected options if it has not been discovered before. It is important noticing that the options are independent on the state where the agent is, but it is defined by the primitives' availability. This definition makes options reusable on different floors and with different objects, just depending on the agent's abilities. The procedure can discover a subset of the options available in the original implementation[1] sufficient to solve the entire game problem.

4.1.2 *Exploration*

After discovering a set of valid options O as explained in the previous section, the system exploits them to explore the environment, collecting data about the reached low-level states (line 10 of Algorithm 2). Considering that the abstracted representation of the world does not change significantly with a small amount of new data, the function *Collect_Data()* is in charge of executing *d_steps* options

1 Link to the original implementation of [51]: https://github.com/sd-james/skills-to-symbols

Algorithm 3 Option Discovery

procedure OPTION_DISCOVERY($d_eps, d_steps, \omega^{EX}$)
  $O_{new} \leftarrow \{\}$
  $ep \leftarrow 0$
  while $ep < d_eps$ **do** //For each episode
    $T \leftarrow 0$
    Reset_Game()
    for $option$ **in** ω^{EX} **do** // *Execute IM plan*
      $Execute(option)$
    end for
    while $T < d_steps$ **do** //For each step
      $s \leftarrow$ *Get_State()*
      $a^p \leftarrow$ *Get_Available_Primitive()*
      while $Is_Available(a^p)$ **and not** (*New_Available_Prim()*) **do**
        $Execute(a^p)$
        $s' \leftarrow$ *Get_State()*
      end while
      if $s \neq s'$ **then**
        if *New_Available_Prim()* **then**
          $a^t \leftarrow$ *Get_New_Available_Prim()*
          $o \leftarrow Create_New_Option(a^p, a^t)$
        else
          $o \leftarrow Create_New_Option(a^p, \{\})$
        end if
        $O_{new} \leftarrow O_{new} \cup o$
      end if
      $T \leftarrow T + 1$ //End *For each step*
    end while
    $ep \leftarrow ep + 1$ //End *For each episode*
  end while
  return O_{new}
end procedure

for *d_eps* episodes, in which the agent starts its exploration from the initial configuration of the environment.

Each timestep, the agent attempts to perform an option $o \in O$ from a certain low-level state $s \in S$. The selection of the action o during the exploration can follow different strategies, which are described in the section 4.1.4. In the case the execution of the option changes the low-level variables of the state s and, consequently, the *mask*[2] m is not null, the system registers two types of data tuple (line 10): the *initiation data* tuple *id* and the *transition data* tuple *td*. The multiple instances of these tuples are stored, respectively, in the datasets ID, for the initiation data, and TD, for transition data. A single *initiation data* tuple id_i takes the following structure

$$id_i = (s, o, f(s, o)), \tag{4.3}$$

where the function $f(s, o)$ returns the feasibility of executing o from s (*True* if $s \in I_o$ and *False* otherwise). The *transition data* tuple td_j takes the following structure

$$td_j = (s, o, r, s', g, m, O'), \tag{4.4}$$

where s' is the state reached after executing option o from the state s, g is a flag saying if the final objective of the game has been reached, m is the *mask* of the option and O' is a list defining the options that can be executed from s'. When all the steps of the episode have been executed, the environment is reset and the next episode is started until the maximum number of episodes *d_eps* is reached, in which case the *Collect_Data()* procedure terminates, and the stored data are added to the existing datasets ID and TD (line 11-12).

4.1.3 *Abstraction*

The datasets collected in the previous step are then used as input for the function $Create_PPDDL()$ (line 13), returning a symbolic representation $\mathcal{D}$ of the agent's current knowledge expressed in PPDDL formalism (PPDDL domain). The main

2 The *mask* is the list of state variables changed by a specific option [51]. See definitions in section 2.2.1.1

advantage of the obtained PPDDL representation is that it makes explicit the causal correlations between operators that would have remained implicit at the option level. In the following, we provide a summary description of the abstraction procedure, for further details, the reader is referred to Chapter 2 and the original article [51].

The procedure executes the following five steps:

1. **Options partition**: this step is dedicated to partitioning the learned options into *abstract subgoal options*, a necessary assumption of the abstraction procedure. Therefore, options are characterized by a single precondition and effect set. However, given the uncertainty of the actions' effects in the environment, the operators' effects will be modelled as *mixture distributions* in the *effect estimation* step (see section 2.2.1.1). This phase utilizes the transition dataset TD collected before, as it captures the information about the domain segment the option modifies. Basically, the transition dataset is divided into sets of transition tuples presenting the same option o and mask m. Then, for each set, the partitions are ultimately obtained by performing clustering on the final states s' through the DBSCAN algorithm [29] implemented in the scikit-learn toolkit [72]. If some clusters overlap in their initiation set, a unique partition is created with different effects and probabilities of occurring.

2. **Precondition estimation**: this step is dedicated to learning the classifiers that will identify the *preconditions* of the PPDDL operators. This operation utilizes the initiation dataset ID to have negative examples of the initiation set classifier for each operator, considering all the samples with option o and $f(s, o) = False$. Instead, the positive examples comprise the initial states s taken from TD tuples belonging to the same partition. The initiation set classifier of the option is computed using the Support Vector Machines (SVM) [26] implemented in scikit-learn. Then, the output of this phase is the set of all the *initiation set* classifiers of all the operators.

3. **Effect estimation**: analogously, this step is dedicated to learning the symbols that will constitute the *effects* of the PPDDL operators. The effects

distribution was modelled through the Kernel density estimation [71, 80], taking in input the s' of each partition.

4. **PPDDL Domain synthesis**: finally, this step is dedicated to synthesising the PPDDL domain, characterized by the complete definition of all the operators associated with the learned options in terms of preconditions and effect symbols. The data structures generated during the previous steps can be easily translated into symbolic operators.

As described in the previous section, the produced PPDDL domain can be potentially used to reach any subgoal that can be expressed in terms of the available generated symbols at any point during the Discovery-Plan-Act (DPA) loop. One interesting aspect of the proposed DPA framework is that the semantic precision of the abstract representation and its expressiveness increase as the DPA cycles proceed, as will be described in the experimental section.

4.1.4 *Goal Selector*

This module aims at simulating the *intrinsic motivations* driving the agent towards interesting areas to satisfy its *curiosity* and optimize its exploration. In particular, one of the most fascinating aspects of this system is the capability of setting its high-level goals, which potentially could be a never experienced combination of symbols. In other words, abstract reasoning could be the driving criterion for exploring the environment. The previous ambitious goal is still the object of future work.

The selection of the target state $s_{target} \in S$ to be reached in the next exploration cycle is performed at line 14 of Algorithm 2, calling the procedure *Get_Target_State*(). The *Goal Selector* suggests the system such state following an internal strategy, which can be *Action Babbling*, *Goal Babbling* and *Distance-based Goal Babbling*. *Action Babbling* is the simplest strategy of the system for the exploration, consisting of a pure random walking of the agent because, returning $s_{target} = NULL$, no plan is generated and executed in the next exploration.

Then, IMs can be implemented in several ways to simulate different agents' behaviours. In the described system, curiosity is formalised as an interest to

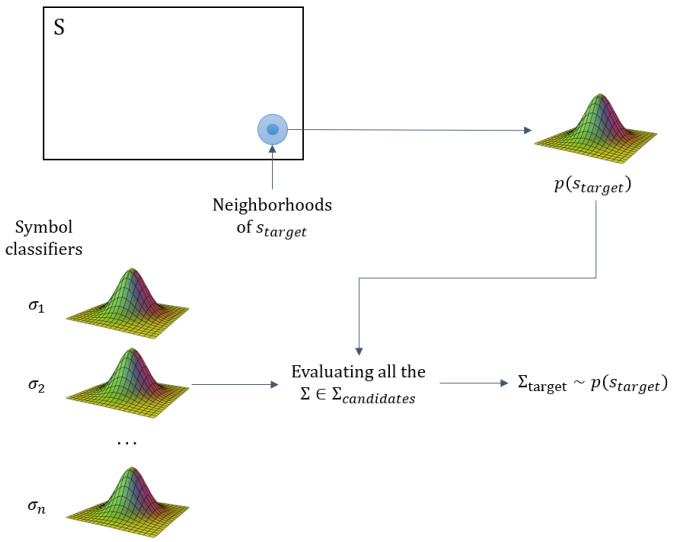


Figure 4.2: The conceptual depiction of the selection of the symbols necessary to represent the goal s_{target}. The set of symbols having the distribution most similar to the goal will define the goal of the PPDDL problem file.

reach (1) a random goal already acquired and (2) the border of its synthesized knowledge. The first strategy is represented by *Goal Babbling*, consisting in randomly selecting a low-level state in the environment and trying to reach it. Usually, the assumption of *Goal Babbling* is that all the goals which can be set belong to the low-level states reachable of the world and the goal is formalised as the configuration of joints or position to be reached with the robot's actuators [87]. Since not all the states $s \in S$ are valid (e.g. the agent can't move inside the wall), this strategy selects a random state s_{target} among the visited ones (line 14). Then, s_{target} is translated into a set of propositional symbols $\{\sigma_1, ..., \sigma_k\}$, as described successively in subsection 4.1.4.1, representing the high-level goal solvable using an off-the-shelf PPDDL planner. The capability of translating low-level states into symbols gives the agent a chance to reason on causal dependencies and, consequently, plan. It is important to notice that a pure *Goal Propositional Babbling*, consisting of the selection of a random subset of high-level symbols, would not be an effective strategy because only a limited number of combinations of symbols conjunctions are valid goals. Consequently, *Goal Propositional Babbling* is not taken into consideration.

The second strategy, modelling the curiosity towards the less explored states, is implemented as a modified version of the *Goal Babbling* influenced by the goal's distance from its starting location s_{init}. We refer to it as *Distance-based Goal Babbling*. In this case, the curiosity level for a state is defined as

$$\eta(s) = \|s_{init} - s + \mathcal{Z}\|, \tag{4.5}$$

consisting of the norm of the difference between the state vector of the agent at the beginning of each episode s_{init} and the visited state s. In this case, the low-level s_{target} is selected between the farthest visited states as following

$$s_{target} = \max_{s} \eta(s), \forall s \in S_{visited}. \tag{4.6}$$

Precisely, the low-level states s target of the exploration is the one maximizing the distance $\eta(s)$, adding a further Gaussian noise $\mathcal{Z} \sim \mathcal{N}(0,1)$ to facilitate the reaching of different states. The *Goal Selector*, in particular, when driven by η, is the main responsible for the knowledge growth of the agent, pushing the system to the border of it. Moving towards the frontier's states, the agent is likely to encounter novel states, which will then be synthesized as symbols, making them reachable through planning, enabling the discovery of new states, and so on.

4.1.4.1 *Translating low-level states into symbols*

The peculiarity and, simultaneously, the biggest challenge of this software architecture is using an abstract symbolic representation to manage the evolution of the agent's knowledge. Using a symbolic representation to describe a desired environment configuration is a powerful tool for an optimized exploration of the world.

After selecting s_{target} (line 14), the purpose of the planning module is twofold. On the one hand, it can be employed as usual to verify the reachability of the goal s_g of the environment (i.e. get the treasure and bring it "home" in the Treasure Game, in line 17) and, on the other hand, it can be exploited to generate a plan driving the agent towards states, in our case s_{target}, relevant to extend the knowledge of the system (line 16). In both cases, to take advantage of planning

it is necessary to transform a low-level state into a high-level one. This operation requires finding the combination of propositional symbols

$$\hat{\Sigma}_{target} = \{\sigma_1, ..., \sigma_k\} \tag{4.7}$$

representing best the portion of state space including s_{target}. In other words, we look for a subset of symbols $\hat{\Sigma}_{target}$ whose *grounding* is s_{target}. These symbols make it possible to generate the definition of a planning problem (line 15), which, together with the domain definition, can be used to perform planning and solve the problem (line 16).

For selecting the right symbols conjunction, the system creates the classifier $Cl_{target} \sim p(s_{target})$ approximating a distribution over s_{target}. Cl_{target} is a SVM classifier trained on the states

$$\{s| \left\|s_{target} - s\right\| \leq \epsilon\}, \forall s \in S_{visited}, \tag{4.8}$$

which are all the neighbours of s_{target} with a maximal distance ϵ, as positive samples and, as negative samples, all the remaining states encountered in the agent's experience. Once the classifier is generated, all the propositional symbols whose *mask* is equal to a *factor*[3] contained by Cl_{target} are collected as candidates $\hat{\Sigma}_{candidates}$. Then, all the subsets of symbols $\hat{\Sigma}_i \subset \hat{\Sigma}_{candidates}$, *without mask overlap* are evaluated as representations of s_{target}. From each $\hat{\Sigma}$, m state samples $S_\Sigma = \{s_1, ..., s_m\}$ are generated and the score of the subset $\hat{\Sigma}$ is calculated as

$$score(\Sigma) = \frac{1}{m} \sum_{s \in S_\Sigma} Cl_{target}(s), \tag{4.9}$$

where $Cl_{target}(s)$ returns the probability that s belongs to the positive class of the classifier Cl_{target}. Then, the subset of symbols $\hat{\Sigma}$ maximizing the *score* function is used as a goal in the problem definition $\mathcal{P}_{target}$. Finally, the problem can be given in input to an off-the-shelf planner and the exploration plan is returned.

3 *"Sets of low-level state variables that, if changed by an option execution, are always changed simultaneously"*[51], see definitions in section 2.2.1.1.

4.1.5 *Planning*

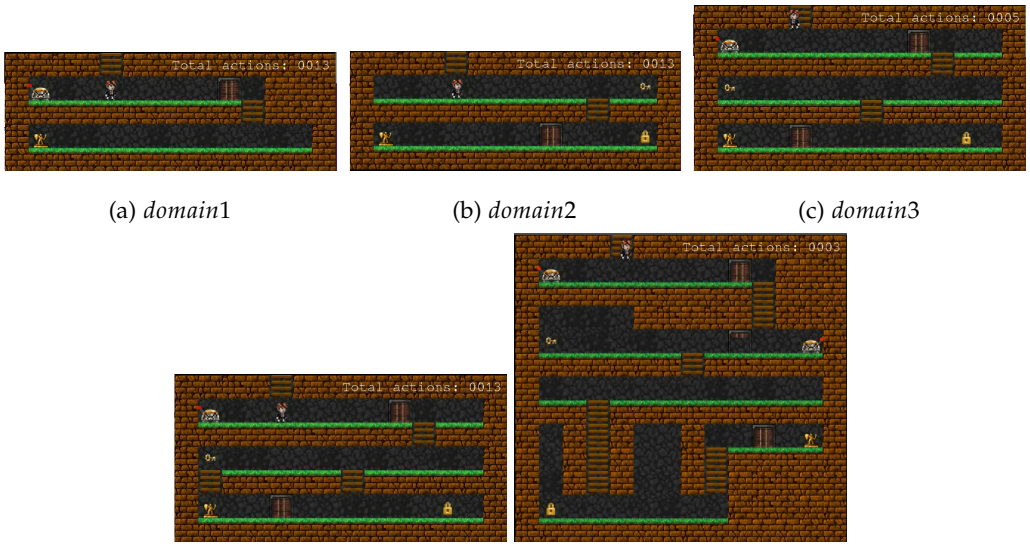
At the end of a cycle, planning generates a plan to reach s_{target} and s_g. In both cases, to create a PDDL problem $\mathcal{P}$, it is necessary to find the set of symbols $\hat{\Sigma}_{init}, \hat{\Sigma}_g$ and $\hat{\Sigma}_{target}$, describing respectively the most suitable high-level state representation for s_{init}, s_g and s_{target}, following the step of the previous subsection 4.1.4.1. Indeed, the couples $(\hat{\Sigma}_{init}, \hat{\Sigma}_g)$ and $(\hat{\Sigma}_{init}, \hat{\Sigma}_{target})$ define the problems $\mathcal{P}_g$ and $\mathcal{P}_{target}$. At line 15, $\mathcal{P}_{target}$ is generated as previously discussed and the plan to solve it, ω^{EX}, is generated by the planner(line 16). Before moving to the next cycle, the system tries to solve also the problem $\mathcal{P}_g$, performing the function *Check_PPDDL_Validity*. The resulting plan ω^g is only used internally by the system to keep track of the success ratio of the planner with the evolution of the synthesized knowledge of the agent.

4.2 EXPERIMENTAL ANALYSIS

The system which has been described in this chapter is designed to discover options available in the environment, collect low-level data of the visited states, create a PPDDL representation of the domain and use it to reach intrinsic and extrinsic high-level goals. This section, dedicated to the experimental analysis, will first describe the dynamics of the environment, followed by an example of the system's cycle execution with its outputs and, finally, the overall results collected over different environment configurations.

4.2.1 *Environment setup*

The implemented system has been tested in the so-called Treasure Game domain [51]. In such an environment, an agent can explore the maze-like space by moving through corridors and doors, climbing stairs, interacting with handles (necessary to open/close the doors), bolts, keys (necessary to unlock the bolts) and a treasure. The agent starts its activity from the ladder on top of the maze (home location) and its overall task is to find the treasure and bring it back to



(a) *domain*1 (b) *domain*2 (c) *domain*3

(d) *domain*4 (e) *domain*5

Figure 4.3: All the domain configurations used in the experimental analysis. The game's purpose is to get the treasure at the bottom of the maze and bring it back to the top ladder.

the starting location. In our experimentation, the agent starts endowed with no previous knowledge about the possible actions that can be executed in the environment; the agent is only aware of the basic motion primitives at its disposal $A = \{go_up, go_down, go_left, go_right, interact\}$, respectively used to move the agent up, down, left or right by 2-4 pixels (the exact value is randomly selected with a uniform distribution) and to interact with the closest object. The interaction with a lever changes the state (open/close) of the doors associated with that lever (both on the same floor or on different floors) while the interaction with the key and/or the treasure simply collects the key and/or the treasure inside the agent's bag (in the bottom-right corner of the screen). Then, the interaction with the bolt opens the door next to the treasure and it is feasible only when the agent has the key in its bag. The state $s \in S$ is defined as

$$s = (x_{agent}, y_{agent}, \theta_1, \theta_2, ..., x_{key}, y_{key}, x_{bolt}, x_{treasure}, y_{treasure}) \tag{4.10}$$

in which x_{agent}, y_{agent} is the (x,y) position of the agent, θ_i is the angle of the lever i, x_{key}, y_{key} is the (x,y) location of the key, x_{bolt} is the state of the bolt (1 if open and 0 if locked) and $x_{treasure}, y_{treasure}$ is the (x,y) location of the treasure.

Domain	Levers	Keys	Bolts	Notes
*domain*1	1	0	0	None.
*domain*2	0	1	1	None.
*domain*3	1	1	1	None.
*domain*4	1	1	1	Shortcut available.
*domain*5	3	1	1	2 levers going to the treasure, 1 going home.

Table 4.1: Obstacles to be solved to end the game.

The system is tested in five different configurations of the environment of increasing complexity: two small (Figure 4.3a and 4.3b), two medium (Figure 4.3c and 4.3d) and one big instance (Figure 4.3e). For instance, in the setting depicted in Figure 4.3e the agent has to (1) pull the two levers on the top of the maze to open the doors, (2) get the key which is used to (3) unlock the bolt, (4) get the treasure and (5) bring it back on top of the environment.

The other configurations' obstacles to be overtaken are shown in Table 4.1. The experiments have been run on a computer with operating system Ubuntu 18.04, Python 3.7, processor i7-7700 @ 3.60GHz, 16GB RAM, and using the high-level planner mGPT [18] for solving the automatically generated planning problems.

4.2.2 *The cycle*

In the next paragraphs, a complete execution cycle of the system is briefly described showing the output of each phase of an intermediate cycle (10^{th}) performed on *domain*3 (see Figure 4.3c) in the *Goal Babbling* strategy case.

OPTION GENERATION At the beginning of each cycle, the agent executes Algorithm 3 to collect some options exploiting the agent's primitives, before the exploration can commence. In the Treasure Game environment selected for this work, the agent executes $d_eps = 1$ episodes, composed by $d_steps = 200$ primitive actions. After the execution of the plan ω^{EX}, the random exploration starts using primitives A. The result is the following set of learned options (11 in total) following the formalization 4.2:

```
O = {(go_up,{}), (go_down,{}), (go_left,{}), (go_left,go_up), (go_left,
      go_down), (go_left,interact), (go_right,{}), (go_right,go_up),
     (go_right,go_down), (go_right,interact), (interact,{})}.
```

It is important to note that, in general, the discovered options are not all the possible options but only those experienced by the agent during the exploration. This procedure is incremental, adding options to the set O each iteration of the Algorithm 2. The agent could potentially discover new options at each iteration of Algorithm 2; analyzing the effects of this iterative improvement is the object of ongoing work. As described in Section 4.1.1, this procedure implements the use of IMs *at low level*, capturing the curiosity of the agent when it discovers to have new available primitives to exploit. These simple options autonomously discovered are then used to explore the environment and proved to be suitable for the generation of a PPDDL representation.

EXPLORATION Also in this step, the plan ω^{EX} is executed before starting the random walk over the available options O. During this step, 600 data entries have been collected in the dataset ID and 6600 in TD. It is important to remember that the number of collected data over the cycles is not constant because when an option does not produce effects on the environment, no data is collected.

ABSTRACTION The PPDDL domain depicted in Figure 4.4 is a part of the output of the abstraction procedure. As a standard PPDDL domain, it lists symbols and operators, available for the planner to generate a plan to reach a certain state, if achievable, which can be represented with a subset of these synthesized symbols. In this specific case, the system generated 26 symbols and 716 operators.

PLANNING The PPDDL domain is used by the system to (1) solve the entire game and (2) improve the exploration. In Figure 4.6 two PDDL problems are illustrated: on the left, the goal of solving the game ($\mathcal{P}_g$) and, on the right, the goal of the *Goal Selector* module, interested in reaching the left corner of the middle floor ($\mathcal{P}_{target}$). The solution for both problems is indicated in Figure 4.7.

```
(define (domain TreasureGame)
    (:requirements :strips :probabilistic-effects :rewards)

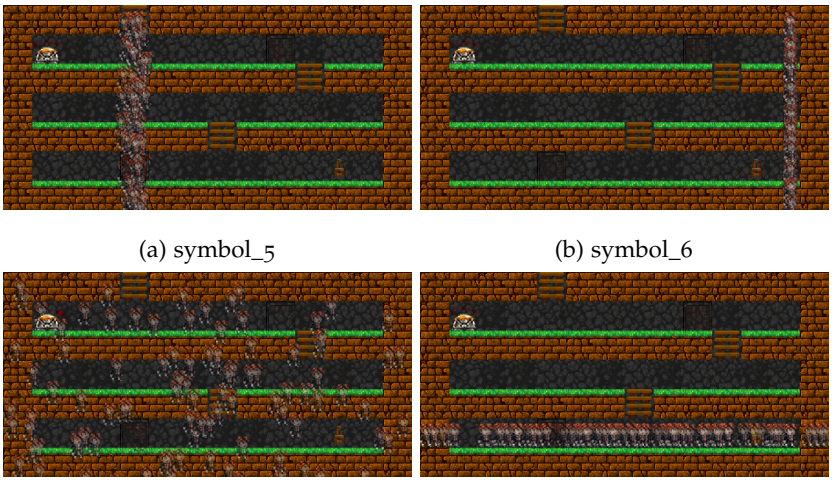
    (:predicates
        (notfailed)
        (symbol_0)
        (symbol_1)
        (symbol_2)
        ...
        (symbol_25)
    )

    (:action option-0-partition-0-0
        :parameters ()
        :precondition (and (notfailed) (symbol_5) (symbol_22)
                    (symbol_14))
        :effect (and (symbol_6) (not (symbol_5)) (decrease
                (reward) 36.00))
    )

    ...

     (:action option-10-partition-3-715
        :parameters ()
        :precondition (and (notfailed) (symbol_20) (symbol_22)
                    (symbol_14) (symbol_15) (symbol_0))
        :effect (and (symbol_13) (not (symbol_20)) (decrease
                (reward) 90.00))
    )
)
```

Figure 4.4: An extract of the PPDDL generated by the abstraction procedure. Notice that `option-0-partition-0-0` can be explained by looking at the symbols of Figure 4.5. In fact, the effect is to change the x position replacing *symbol*_5 with *symbol*_6, maintaining invariant the presence of *symbol*_14 and *symbol*_22. Consequently, it is the option moving the agent on the right on the last floor.



(a) symbol_5 (b) symbol_6

(c) symbol_14 (d) symbol_22

Figure 4.5: Semantics of the symbols employed by the first operator of the generated PPDDL: (a) and (b) mean being in a certain x position, (c) mean the lever is pulled and (d) being on the bottom floor.

The first solution, ω^g, solves the game problem in 21 moves and the second one, ω^{EX}, reaches the *Goal Selector* goal in 6 moves.

4.2.3 *Results*

In this subsection, the overall results of the system over different domain instances are shown. First, the setting of the environment is discussed, then the baseline adopted and the other strategies enabling the planning exploration. Finally, some charts will be illustrated highlighting the system's performance in terms of success ratio on the game, abstraction time and symbols/operators generation. In the last subsection, we will discuss some issues that occurred in the system which should be solved and the limitations of the technologies employed.

For our purposes, the Treasure Game[4] has been used with five different mazes (see Figure 4.3), to focus on the performances of the system on smaller domains, highlighting pros and cons of the symbolic approach proposed. The system has been executed, following the workflow described in Algorithm 2, in the cited five

4 Github repository: https://github.com/sd-james/gym-treasure-game

```
(define (problem game_goal)
    (:domain TreasureGame)

    (:init (notfailed) (symbol_0) (symbol_1)
           (symbol_2) (symbol_3)
           (symbol_4) (symbol_5) )

    (:goal (and (notfailed) (symbol_4)
           (symbol_17) (symbol_18)) )
)
```

```
(define (problem im_goal)
    (:domain TreasureGame)

    (:init (notfailed) (symbol_0)
           (symbol_1) (symbol_2)
           (symbol_3) (symbol_4)
           (symbol_5) )

    (:goal (and (symbol_13) (symbol_24)
           (symbol_14) (symbol_2)
           (notfailed)) )
)
```

Figure 4.6: On the left, the problem of solving the game is synthesized by the system $\mathcal{P}_g$ and, on the right, the problem generated by the *Goal Selector* module $\mathcal{P}_{target}$.

```
PLAN GAME GOAL:
[1:(go_down,{}), 2:(go_left,go_down), 3:(interact,{}),
 4:(go_right,go_down), 5:(go_down,{}), 6:(go_right,{}),
 7:(interact,{}), 8:(go_left,{}), 9:(go_right,go_down),
 10:(go_down,{}), 11:(go_right,interact), 12:(interact,{}),
 13:(go_left,go_down), 14:(interact,{}), 15:(go_right,go_up),
 16:(go_up,{}), 17:(go_right,{}), 18:(go_left,go_up),
 19:(go_up,{}), 20:(go_left,go_up), 21:(go_up,{})]

PLAN IM GOAL:
[1:(go_down,{}), 2:(go_left,go_down), 3:(interact,{}),
 4:(go_right,go_down), 5:(go_down,{}), 6:(go_left,interact)]
```

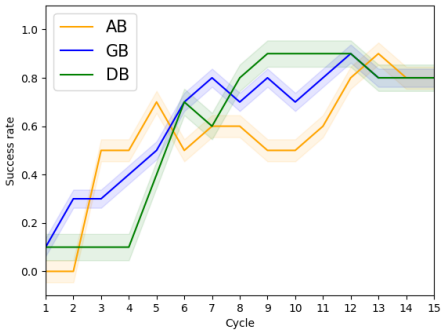
Figure 4.7: The plans generated ω^g and ω^{EX}.

mazes configurations using parameters suitable to solve the tasks described later in this section. To summarize, the system iteratively (1) looked for new options O_{new} performing d_steps primitives for d_eps episodes, (2) explored for dpa_eps episodes the maze executing dpa_steps, (3) created a symbolic abstraction of the domain $\mathcal{D}$, (4) created a plan ω^{EX} to optimize the exploration of the successive cycle $c+1$. It is important noticing that the options autonomously discovered as described in Algorithm 3 successfully produced a valid high-level model able to build correct plans. To assess the accuracy of the current abstraction $\mathcal{D}$, we evaluate its ability to reach the goal s_g at the end of each cycle. This is done by requesting the planner to solve the problem $\mathcal{P}_g$ using the current symbolic domain description $\mathcal{D}$.

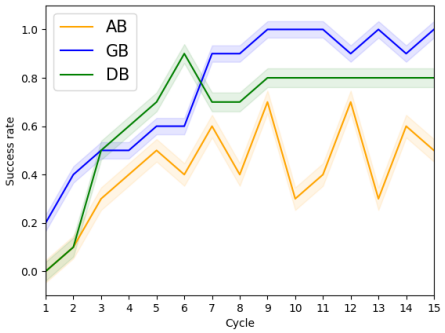
The first strategy used in the experiments is the random walk, which is called *Action Babbling* [24]. As said before, this strategy simply executes the Algorithm 2 without using planning because the *Goal Selector* returns $s_{target} = NULL$ and no plan is generated. This simple strategy is used as a baseline for the experimental analysis, and it is necessary to notice the advantages of using the symbolic approach. The exploration is equivalent to the one used by Konidaris [51], and we use it to observe the development of the symbolic model over time.

To support the use of symbolic planning, two other strategies have been considered: *Goal Babbling* and *Distance-based Goal Babbling*, already described in subsection 4.1.4. These three strategies could be seen as having an increasing complexity and effectiveness in the exploration. Indeed, *Action Babbling* selects completely random actions, *Goal Babbling* selects random goals reaching them with a planned sequence of actions, and *Distance-based Goal Babbling* selects the farthest goals and reaches them by using planning (see subsection 4.1.4). It is straightforward that in smaller domains where the reasoning is less necessary, the differences do not occur, while in the bigger domains, the time to reach the goal changes significantly.

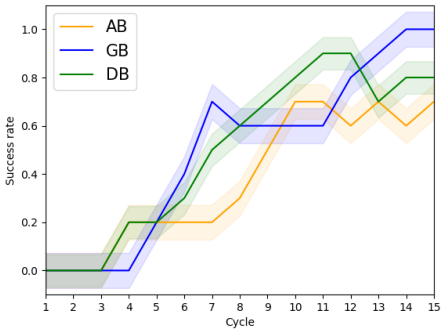
The five configurations considered are depicted in Figure 4.3, applying to the exploration function *Collect_Data* the parameters shown in Table 4.2. Smaller domains *domain*1 and *domain*2 needed 50 options per episode, 150 for *domain*3 and 200 *domain*4, then 800 for *domain*5.



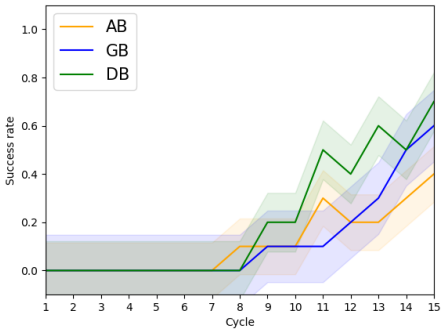
(a) domain 1



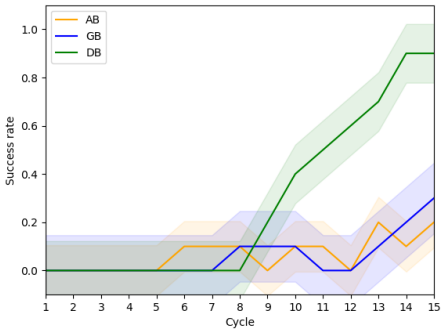
(b) domain 2



(c) domain 3



(d) domain 4



(e) domain 5

Figure 4.8: The figure depicts the success rate of ω^g using different exploration strategies over time. In figure 4.8a, 4.8b, 4.8c, 4.8d and 4.8e, we have respectively the results of the domain of figures 4.3a, 4.3b, 4.3c, 4.3d, 4.3e over 15 cycles of exploration.

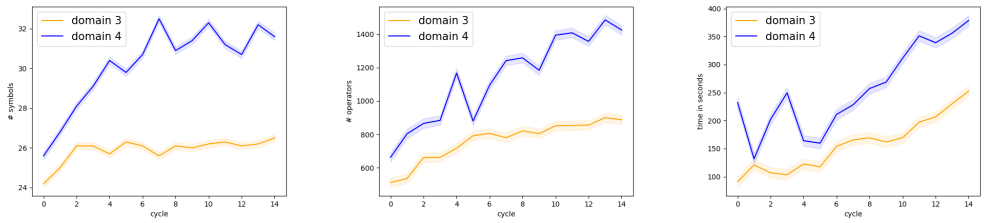
Domain	*dpa_eps*	*dpa_steps*	minimal solution steps
*domain*1	4	50	11
*domain*2	4	50	13
*domain*3	4	150	19
*domain*4	4	200	15
*domain*5	4	800	31

Table 4.2: Main parameters used in the different environment configurations.

The main results of the system are depicted in Figure 4.8. Precisely, in the graphs, we can see the probability of success in solving the game over time using different strategies. Such success entails the generation of a sufficiently mature domain $\mathcal{D}$ and a correct problem formalization of $\mathcal{P}_g$, resulting in a correct plan ω^g. The system has been run with $cycles = 15$, storing at the end of each cycle whether ω^g was able to solve the game using the current synthesized PPDDL domain representation $\mathcal{D}$. This mechanism has been performed for ten trials. Consequently, in the charts of Figure 4.8 are depicted cycles on the *x-axis*, and the percentage of trials that have succeeded in solving the game in the specific cycle on the *y-axis*.

In the smaller domains *domain*1, *domain*2, *domain*3 and *domain*4, the planning contribution is limited and sometimes not visible at all. Especially in the simplest domain *domain*1 the three strategies present similar performances, solving in the last cycles 80-90% of the trials (Figure 4.8a). In domains *domain*2 and *domain*3, presenting slightly higher complexities, the advantages of executing plans start being visible (Figure 4.8b and 4.8c). It is evident that *Goal Babbling* behaves almost perfectly in the last cycles, demonstrating the usefulness of collecting transition data with reasonable sequences of actions. The combination of randomness in selecting the goal to be reached and the reasoned transitions speed up the exploration and the consequent maturity of the PPDDL representation of the environment. Instead, the *Distance-based Goal Babbling* seems better than a completely random search but less effective than the precedent. This result entails that applying an exploration focused on the frontier's goals is not

convenient in smaller domains. In fact, in the conclusive cycles of *domain*2 and *domain*3, *Goal Babbling* maintains 90-100% of success ratio, *Distance-based Goal Babbling* around 80% and *Action Babbling* between 60-70%.



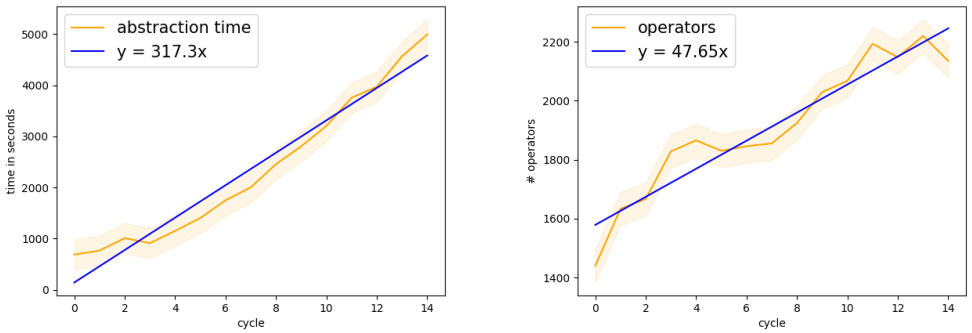
(a) symbols (b) operators (c) time

Figure 4.9: Comparison between *domain*3 and *domain*4 regarding symbols/operators generation and iteration time for each cycle.

Although the problem faced by *domain*3 seems easier than *domain*4, presenting respectively minimal solutions of 19 and 15 steps (see Table 4.2), the system needs more steps per episode to solve *domain*4 using the same amount of episodes used by *domain*3. The cause of this behaviour is given by the higher complexity of synthesizing a PPDDL domain $\mathcal{D}$ for this scenario. In fact, the scenario *domain*4 presents a sort of "shortcut" to reach the treasure, which does not require collecting the key and opening the door. From the point of view of the abstraction procedure, the shortcut is another branch of the possible choices of the agent, presenting additional concepts. Consequently, the system generates additional symbols and operators, requiring more experience to strengthen the transition model captured by the PPDDL and supply satisfying plans. It is worth noticing that to obtain our results in *domain*4, more exploration steps have been necessary to reach significant results. In Figure 4.9, it is straightforward that *domain*4 contains more information captured by the system, with respect to *domain*3. Then, In *domain*4 the success of the different strategies are similar around the 15th cycle, with the *Action Babbling* slightly worse than its competitors.

The significance of the frontier exploration emerges in domain *domain*5 where planning boosts evidently the results of the agent. Indeed, being bigger than other domains, *domain*5 is more difficult to be solved and planning results being efficient in driving the exploration of the agent immediately towards the borders

of its knowledge. The main result highlighted by the charts is that in bigger domains (Figure 4.8e), the impact of using planning is evident. Planning is able to easily drive the agent towards interesting visited states, where it can continue exploring. Statistically, after collecting the transition data of 48k random option executions, the system struggles to solve the problem, never exceeding 10% of success. Similar behaviour for *Goal Babbling*, reaching most 20% of success. Instead, *Distance-based Goal Babbling* results being extremely effective, reaching 90% of success in 15 cycles. Conceptually, the strategy continuously pushes the agent towards the frontiers of the visited states, increasing the probability of encountering new unexplored areas and reducing redundant data.



(a) Abstraction time mean on *domain*5 using *Distance-based Goal Babbling* strategy (orange) and the corresponding linear regression (blue).

(b) Operators mean on *domain*5 using *Distance-based Goal Babbling* strategy (orange) and the corresponding linear regression (blue).

Figure 4.10: Visualization of abstraction time and operators in different cycles.

The results regarding *abstraction execution time and operators* generation do not present significant dissimilarities using different strategies and domains. For the sake of simplicity, figures 4.10a and 4.10b depicted the mean results of the strategy *Distance-based Goal Babbling* on the bigger domain (see Figure 4.3e). Adding around 3k and 30k data entry, respectively, to TD and ID, the time employed by the system seems to increase linearly, requiring 317 seconds more to create the new abstraction. Instead, operators seem initially to grow linearly, but in the last four cycles, the number of operators becomes constant at around 2100 units. As shown in Chapter C, this parameter has not a regular trend, and it is not a quality metric because the number of operators (and symbols)

is influenced by the data and stochastic methods applied for classification and partitioning. For such reason, the best quality metric to apply has been the success rate of Figure 4.8.

4.2.4 *Issues and limitations*

As shown in Figure 4.8, the results are not deterministic and change over time. On the one hand, the system is continuously evolving in terms of knowledge and, on the other hand, ML techniques introduce further stochasticity to the final representation generated. During the abstraction procedure, described in subsection 4.1.3, some statistical tools are employed to create data structures to represent preconditions, effects and symbols. For instance, an erroneous[5] clustering phase could generate unexpected effects on symbols and operators. An example highlighting this fact is the *noisiness of some symbols*, interfering with the generation of a correct formalization of $\mathcal{D}, \mathcal{P}$ and, consequently, the resulting plan ω. In Figure 4.11 it can be seen the graphical representation of the symbol "on the highest y-axis position", meaning "on the top ladder" because it is the only possible state with such y-axis value (it is not allowed to move inside the walls). In some cases, it could happen that the initial state of the game

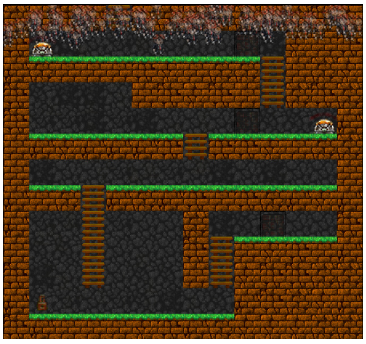


Figure 4.11: The graphical representation of the symbol "on the higher y-axis position". It can be seen that it is significantly noisy because some samples of the agent are depicted on the top y-axis and other samples almost on the lower floor.

5 Statistical methods are not mistaken because they just create a representation based on the data provided. However, for our purposes, some models' instances could be obstacle on the reaching of our goal.

is interpreted as being at the top floor under the ladder and, consequently, it is not necessary to climb down the ladder to execute the agent's task. Then, although almost the whole plan ω is correct to complete the game, without the option of climbing down the ladder as the first action, the plan is incomplete, and the trial is considered unsuccessful. On the one hand, this is a drawback of using ML tools, which can introduce noise in the symbolic representation. On the other hand, it is the strong point of the probabilistic approach, always proposing solutions, even though presenting a certain degree of error.

The system proposed is limited by the *propositional logic* representation, which does not permit the inclusion of arguments in operators and symbols. Such limitation could be overtaken by developing a new abstraction procedure following other logics (e.g. First Order Logic (FOL)). However, at the moment, it seems not available such a solution in literature and its formalization could not be trivial. In addition, also the implementation and formalization of parametrized policies are not minor tasks [94].

Another aspect to be analyzed deeper is the *management of the knowledge data*. Unfortunately, the abstraction module has no real incremental nature, but the abstraction procedure is executed over all the data, each cycle from scratch. This way of operating is not computationally efficient and does not reflect the learning process of the human being. However, given that in our domains all the necessary knowledge is discovered in a certain amount of time and does not change, a first improvement to be applied to the system could be maintaining a maximum of transition and initiation data discarding over-sampled transitions. Therefore, a sort of filter in the knowledge acquisition could stop the growth in terms of the abstraction procedure's time in stationary and limited domains but also reduce it for all the other cases.

Finally, the abstraction time seems linear or slightly worse (in the worst-case exponential according to Konidaris [51]). Consequently, it would be fundamental to find solutions to mitigate this aspect. Possible actions could be to filter the acquired data or structure the collected knowledge in another form more efficiently for the abstraction process and, eventually, produce different complementary representations which are used according to the task to be tackled.

5

DISCUSSION AND CONCLUSIONS

The objective of this thesis was to present an open-ended learning system that integrates planning and learning. The proposed system aims to discover and learn new low-level actions that can be leveraged at a high level through planning.

For this purpose, this thesis proposes two novel software architectures for open-ended learning in autonomous agents, which are based on intrinsically motivated planning with PDDL representation. The first architecture aims at learning complex policies using sub-symbolic models, while the second architecture focuses on the discovery of new knowledge, such as options and states.

In Chapter 3, the IMPACT architecture was presented as an extension of the standard three-layer architecture well-known in robotics, providing some modules to integrate intrinsic motivations in the agent to discover new actions and improve its operational capabilities.

Then, in Chapter 4, another implementation of such a solution was discussed. With respect to the previous architecture, in this case, the focus was on the discovery of new knowledge relaxing the problem of learning complex robotic skills. The presented system demonstrated to use intrinsic motivations to learn new actions from scratch in a one-shot approach and construct with them a high-level representation for planning, which has been exploited to speed up the discovery of further new knowledge.

5.1 FUTURE WORK

The implementation of open-ended learning agents towards the AGI level has several directions to be investigated. One critical aspect of the research is the *knowledge representation*, the choice influencing all the others. PDDL and

PPDDL have been adopted in this work, and investigating alternative knowledge representation frameworks is necessary. While PDDL has proven useful in this context, it may be worthwhile to explore other knowledge representation frameworks, such as RDDL or use more sophisticated logics (e.g. first-order logic) and compare their effectiveness in open-ended learning scenarios. However, to reach this goal, it is also necessary to *enhance the abstraction capabilities* of the system. The abstraction capabilities of the proposed system could be further developed to enable more sophisticated generalization and transfer of knowledge across tasks and domains. The field of explainable AI may provide insights for improving the management of the data collected for abstraction [42]. In addition, integration with Large Language Models (LLM) [106] should be analysed to investigate the possibility of translating the symbolic domain automatically synthesized into a more readable version.

Then, the optimization of the abstraction process should be approached from several perspectives. One of them is to take advantage of the already generated knowledge representation as prior information whenever possible instead of ignoring it every time the abstraction process is executed. Since much of the knowledge remains constant over time, it is worth exploring how to process only the new experiences and how to efficiently integrate them with the previous ones by implementing an *incremental abstraction process*, which could lead to significant improvements in the efficiency and effectiveness of the overall system. Alternatively, one way to optimize the system could be filtering out less significant experience data.

Other learning algorithms should be analysed, such as meta-learning [43]. It consists in using previous learning experiences to improve the learning process itself, such as adapting to new tasks or changing conditions. Meta-learning can help in improving the efficiency and generalization of the learning process.

5.2 DISCUSSION

The experiments described in Chapter 3 and 4 demonstrated the feasibility of using an approach integrating components deriving from the field of re-

inforcement learning (for learning new policies and the adoption of intrinsic motivations) and classical planning to have an extendable symbolic system. RL and, consequently, sub-symbolic methods have been demonstrated to be extremely powerful tools for learning complex robotic policies involving actuators, and a classical planning representation results in organizing different policies to concatenate them in an effective way. Intrinsic motivations revealed their fundamental role in these systems, showing that they can effectively extend the agent's knowledge and skills without relying on external supervision or rewards.

The systems presented in the previous chapters have demonstrated certain limitations, such as the *limited capability for option learning*. While the action learning modules implemented in this work were found to be suitable for limited applications, there is a need for more generic and effective modules. Although significant progress has been made in reinforcement learning for solving complicated tasks, the challenge of implementing an agent capable of autonomously learning a wide range of capabilities for real-world use remains. Thus, it is worth investigating the implementation of a more sophisticated and flexible parametrized form of option, potentially through the use of advanced sub-symbolic models.

In order to handle the increased complexity of the action model, a suitable representation must be introduced. PDDL and PPDDL have been utilized in our systems and have shown to be suitable for our use cases, but they lack compactness when used for modelling propositional logic. In order to use these representations in a more compact manner, it could be more effective to adopt first-order logic. Although PDDL and PPDDL are compatible with this formalism, it is not straightforward to autonomously derive a representation of this type. It is necessary to consider the feasibility of implementing a FOL version of the abstraction process.

Finally, the level of abstraction used by the system influences the whole problem-solving and detection pipeline. Consequently, a crucial aspect of the AGI will be finding the right abstraction for each problem it will try to solve to tackle it efficiently [49].

5.3 CONCLUSIONS

In this thesis, we presented a novel approach for open-ended learning in autonomous agents based on intrinsically motivated planning with PDDL representation. Our approach integrates two powerful paradigms, intrinsic motivation and classical planning, to enable agents to continuously learn and improve their knowledge and skills without relying on external supervision or rewards.

This work suggests an alternative or complementary approach to the advanced and popular sub-symbolic methods, demonstrating interesting features. First, it allows agents to explore and learn in a self-directed and open-ended manner without being limited to a predefined set of goals or tasks. Second, it enables agents to represent and reason about their knowledge and skills in a structured and formal way, which can facilitate planning and generalization to new situations. Third, it can incorporate intrinsic motivations that drive the agent to explore and learn beyond extrinsic goals, which can enhance the agent's adaptability, robustness, and creativity. However, there are still several challenges and opportunities for future research in this area to enable the systems to perform complex activities in a relevant operational environment.

Overall, we believe that our approach represents a promising step towards more autonomous and intelligent agents that can continuously learn and improve in an open-ended and self-directed manner. We hope our work will inspire and guide further research in this exciting and rapidly evolving field.

Part I

APPENDIX

A

BACKGROUND: SUPPLEMENTARY MATERIAL

This chapter provides a comprehensive description of additional technical tools referenced in the thesis, aiming to enhance the understanding of the systems' output. In particular, languages used for planning (PDDL and PPDDL) and the middleware ROS are briefly described.

A.1 PDDL: PLANNING DOMAIN DEFINITION LANGUAGE

After defining the general semantics of classical planning in section 2.1.2, this section has the aim of giving a flavor of the syntax used by classical planning, in particular by the Planning Domain Definition Language (PDDL) [57]. PDDL is a language to define *domains* and *problems* in classical planning for deterministic environments. While PDDL offers various features, only the fundamental elements are discussed in this section. Providing a detailed description of this language falls outside the scope of this thesis.

To describe the fundamental elements of PDDL syntax, it is proposed below a minimal version of the *Dock Worker Robots (DWR) domain* [38], which involves a container in a pile, a crane that can pick up and put down the container, and a robot that can carry the container and move it from one location to another.

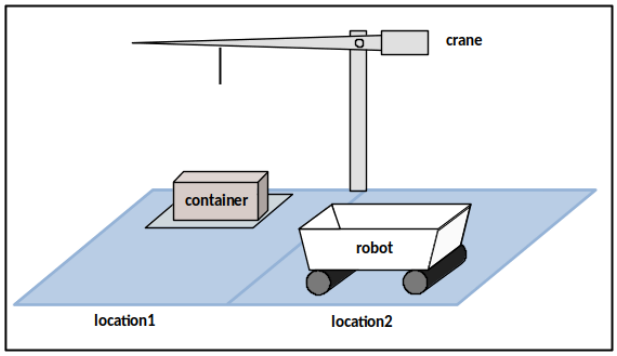


Figure A.1: *Dock Worker Robots* domain representation.

```
(define (domain DockWorkerRobots)
    (:requirements :strips)
    (:predicates
        (onground)
        (onrobot)
        (holding)
        (atlocation1)
        (atlocation2)
    )
    (:action take
        :parameters ()
        :precondition (onground)
        :effect (and (holding) (not (onground)))
    )
    (:action put
        :parameters ()
        :precondition (holding)
        :effect (and (onground) (not (holding)))
    )
    (:action load
        :parameters ()
        :precondition (and (holding) (atlocation1))
        :effect (and (onrobot) (not (holding)))
    )
    (:action unload
        :parameters ()
        :precondition (and (onrobot) (atlocation1))
        :effect (and (holding) (not (onrobot)))
    )
    (:action move1
        :parameters ()
        :precondition (atlocation2)
        :effect (and (atlocation1) (not (atlocation2)))
```

```
    )
    (:action move2
        :parameters ()
        :precondition (atlocation1)
        :effect (and (atlocation2) (not (atlocation1)))
    )
)
```

The *domain* is the description of the environmental's dynamics. After assigning it a name (in this case *DockerWorkerRobots*), the *requirements* are listed, keywords to communicate the domain's characteristics to the chosen planner. These keywords determine whether the domain is solvable by the specific solver. In the provided example, a solver capable of comprehending STRIPS-like representation is required; otherwise, planning becomes infeasible.

Then, the two most important elements of the PDDL domain are *predicates* and *actions*. *Predicates* are precisely the *symbols* described in section 2.1.2, a name given to a certain set of states $s \in S$ satisfying a specific condition. In the DWR domain, `(onground)` holds when the container is on the ground, `(onrobot)` means that the container is on the robot, `(holding)` means that the crane is holding the container, `(atlocation1)` means that the robot is at location 1 and `(atlocation2)` means that the robot is at location 2.

Finally, *actions* are the operations available in the environment, generating a transformation of the world. As already mentioned in the thesis, *actions* are characterized by *preconditions*, the set of predicates must be true to execute the action, and *effects*, indicating which predicates become true and which false. For instance, the action `take` is feasible when `(onground)` holds and its effects are to negate it and set to true the predicate `(holding)`.

Instead, the *problem* definition is responsible for describing the current state of the world, listing all the predicates true before launching planning, and the desired goal to be reached.

```
(define (problem drwProb) (:domain DockWorkerRobots)
    (:init (onground) (atlocation2))
    (:goal (and (onrobot) (atlocation2)))
)
```

Indeed, the problem depicted above, after naming the problem `drwProb` and declaring the domain to be used for reasoning, states with the keyword `:init` the list of the predicates initially true in the world and with `:goal` the list of predicates which have to be true after executing the plan provided by the solver. In this specific case, a possible plan would be $\omega = (take), (move1), (load), (move2)$.

A.2 PPDDL: THE PROBABILISTIC PDDL

PPDDL [105] extends standard PDDL [37] by adding both *probabilistic effects* and *rewards* to represent Markov Decision Processes (MDPs) and, consequently, stochastic domains.

First, *probabilistic effects* employee the following syntax:

$$(probabilistic\ p_1\, e_1\ \ldots\ p_k\, e_k), \tag{A.1}$$

listing all the possible action's outcomes e_i and the corresponding probability of occurring p_i. PPDDL requires that the effects of each action fulfill the constraint $\sum_{i=1}^{k} p_i = 1$ with $p_i \geq 0$. However, when $\sum_{i=1}^{k} p_i < 1$, PPDDL implicitly considers an additional null outcome with probability $q = 1 - \sum_{i=1}^{k} p_i$.

Then, *reward* is defined for each possible action's effect using the syntax $(\langle additive-op \rangle\ reward\ \langle f-exp \rangle)$, where $\langle additive-op \rangle$ identifies an increment or decrease of the reward and $\langle f-exp \rangle$ a numeric expression defining the reward's change.

```
(:action take
    :parameters ()
    :precondition (onground)
    :effect (probabilistic
                0.99 (and (holding) (not (onground))
                     (increase (reward) 0.2) )
                0.01 (and (onrobot) (not (onground))
                     (increase (reward) 1.0) )
            )
)
```

A probabilistic version of the previous action `take` is proposed above, in which it is assumed that, after executing this action, the crane holds the container in 99% of the cases or, unexpectedly (and imaginatively), launches it directly on the robot otherwise. In addition, both outcomes increase the agent's reward, in the first case by 0.2 and in the latter one by 1.0, given the particularly lucky result.

A.3 ROS: ROBOT OPERATING SYSTEM

Robotic Operating System (ROS) [73] is an open-source meta-operating system particularly suitable for robotics applications. The implementation of the IMPACT system using this software tool proves that our architecture can be applied to real-world robots.

Indeed, it implements a lot of useful services such as hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management.

Briefly, implementing an application in ROS means creating a graph of independently running processes (called nodes), communicating to each other using standard or customized messages, working on the same machine or distributed over a network of machines. Usually, nodes perform in a loop their activities, interacting with other nodes in two ways: using 1) *topics* and 2) *services*.

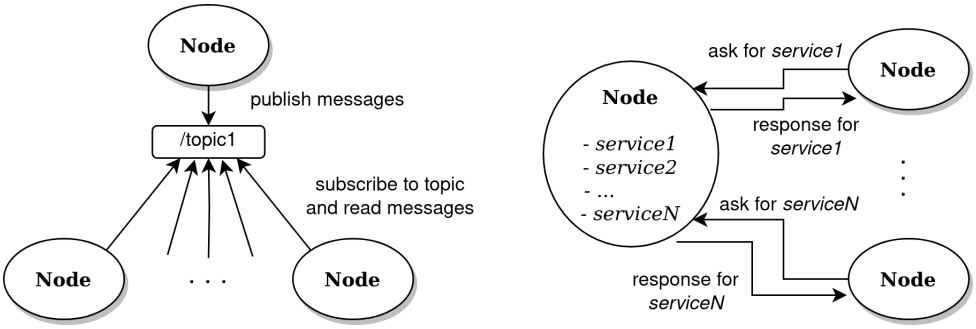


Figure A.2: Schemes of topics and services workflow.

TOPICS The first communication method is using *topics*. They implement in ROS the pattern *publisher—subscriber*, in which a node owns an information and publishes it continuously over a channel called *topic*. Other nodes, working

in the same ROS environment, know the available topics and the type of the messages published by them. Therefore, nodes can subscribe to the topics they need, receiving all the messages published from that moment on.

SERVICES Instead, *services* implement the pattern *request-response*, similar to the *client-server* one. In this case, the node exposes different *services* which can be called when necessary from other nodes. Services can also be called sending additional arguments, if required, perform the activity object of the service and return feedback or the result.

Finally, this middleware has been used in Chapter 3 to create different processes, working on the same machine, able to communicate to each other easily. As in robotics, IMPACT used nodes to read sensors' value and publish them to be used by other processes, for instance, in charge of converting these raw data into a high-level planning domain or executing actions in the simulated environment.

B

EXPERIMENTS OF CHAPTER 3

Being calculated in an automated fashion, expectedly, the obtained PDDL domain is not immediately readable to the human eye. In order to attempt to associate a meaning to the produced operators, all the automatically produced symbols, together with their related grounding classifiers, are listed in section B.2 and B.4. As the reader can observe, each symbol is associated with a list of intervals for each low-level variable representing the symbol's classifier. The meaning of each variable can be read in section 3.2.1. Symbol's classifier is expressed in terms of a set of intervals whose bounds are [0, 50], [240, 255] or [0, 255]. The [0, 50] interval is equivalent to the *false* range, the [240, 255] interval is equivalent to the *true* range, while the [0, 255] interval is equivalent to the *don't care condition*. The last case means that the variable is not considered to calculate the value of the classifier.

For instance, *Symbol*_0 is represented with all the variables' values at [0, 255] (don't care) and variable 3 at [240, 255], meaning true. As described in Figure 3.5, variable 3 means for the agent to see an object in front of him. Consequently, the semantics of *Symbol*_0 is *"Object in sight"*.

In addition, for all the symbols it is mentioned the options whose effects have played a role in producing that symbol and its semantics in a non-formal language.

B.1 ROBOTIC ARM SCENARIO: THE INITIAL PDDL DOMAIN

```
(define (domain roverDomain)
(:requirements :strips)

(:predicates
(Symbol_0)
```

```
(Symbol_1)
(Symbol_2)
(Symbol_3)
(Symbol_4)
(Symbol_5)
(Symbol_6)
(Symbol_7)
(Symbol_8)
(Symbol_9)
)

(:action opt_0
:parameters ()
:precondition (and (Symbol_3) (Symbol_4))
:effect (and (Symbol_0) (Symbol_1) (not (Symbol_3))
            (not (Symbol_6)))
)

(:action opt_1
:parameters ()
:precondition (and (Symbol_3) (Symbol_4))
:effect (and (Symbol_0) (Symbol_2) (not (Symbol_4))
            (not (Symbol_6)))
)

(:action opt_2
:parameters ()
:precondition (and (Symbol_1) (Symbol_4) (Symbol_7))
:effect (and (Symbol_3) (not (Symbol_1)))
)

(:action opt_3
:parameters ()
```

```
:precondition (and (Symbol_2) (Symbol_3) (Symbol_7))
:effect (and (Symbol_4) (not (Symbol_2)))
)

(:action opt_4
:parameters ()
:precondition (and (Symbol_0) (Symbol_7) (Symbol_9))
:effect (and (Symbol_5) (not (Symbol_7)))
)

(:action opt_5
:parameters ()
:precondition (and (Symbol_0) (Symbol_5) (Symbol_9))
:effect (and (Symbol_8) (Symbol_6) (Symbol_7)
            (not (Symbol_0)) (not (Symbol_5)) (not (Symbol_9)))
)

(:action opt_6
:parameters ()
:precondition (and (Symbol_8) (Symbol_7))
:effect (and (Symbol_9) (not (Symbol_8)))
))
```

B.2 ROBOTIC ARM SCENARIO: THE INITIAL SET OF SYMBOLS

```
Symbol s_0 (Symbol_0) Ref. Option = o_[0, 1]
Semantics: Object in sight
[0, 255]
[0, 255]
[0, 255]
[240, 255]
[0, 255]
[0, 255]
[0, 255]
[0, 255]
[0, 255]
[0, 255]

Symbol s_1 (Symbol_1) Ref. Option = o_[0]
Semantics: Rover is in zone_1
[0, 255]
[0, 255]
[0, 255]
[0, 255]
[0, 255]
[0, 255]
[0, 255]
[0, 255]
[240, 255]
[0, 255]

Symbol s_2 (Symbol_2) Ref. Option = o_[1]
Semantics: Rover is in zone_2
[0, 255]
[0, 255]
```

```
[0, 255]
[0, 255]
[0, 255]
[0, 255]
[0, 255]
[0, 255]
[0, 255]
[240, 255]

Symbol s_3 (Symbol_3) Ref. Option = o_[2]
Semantics: Rover is NOT in zone_1
[0, 255]
[0, 255]
[0, 255]
[0, 255]
[0, 255]
[0, 255]
[0, 255]
[0, 255]
[0, 50]
[0, 255]

Symbol s_4 (Symbol_4) Ref. Option = o_[3]
Semantics: Rover is NOT in zone_2
[0, 255]
[0, 255]
[0, 255]
[0, 255]
[0, 255]
[0, 255]
[0, 255]
[0, 255]
[0, 255]
```

```
[0, 50]

Symbol s_5 (Symbol_5) Ref. Option = o_[4]
Semantics: Object grasped
[0, 255]
[0, 255]
[0, 255]
[0, 255]
[0, 255]
[0, 255]
[240, 255]
[0, 255]
[0, 255]
[0, 255]

Symbol s_6 (Symbol_6) Ref. Option = o_[5]
Semantics: Object NOT in sight
[0, 255]
[0, 255]
[0, 255]
[0, 50]
[0, 255]
[0, 255]
[0, 255]
[0, 255]
[0, 255]
[0, 255]

Symbol s_7 (Symbol_7) Ref. Option = o_[5]
Semantics: Object NOT grasped
[0, 255]
[0, 255]
[0, 255]
```

```
[0, 255]
[0, 255]
[0, 255]
[0, 50]
[0, 255]
[0, 255]
[0, 255]

Symbol s_8 (Symbol_8) Ref. Option = o_[5]
Semantics: Object stowed
[0, 255]
[0, 255]
[0, 255]
[0, 255]
[0, 255]
[0, 255]
[0, 255]
[240, 255]
[0, 255]
[0, 255]

Symbol s_9 (Symbol_9) Ref. Option = o_[6]
Semantics: Object NOT stowed
[0, 255]
[0, 255]
[0, 255]
[0, 255]
[0, 255]
[0, 255]
[0, 255]
[0, 50]
[0, 255]
[0, 255]
```

B.3 ROBOTIC ARM SCENARIO: THE NEW PDDL DOMAIN

```
[tabsize=4]
(define (domain roverDomain)
(:requirements :strips)

(:predicates
(Symbol_0)
(Symbol_1)
(Symbol_2)
(Symbol_3)
(Symbol_4)
(Symbol_5)
(Symbol_6)
(Symbol_7)
(Symbol_8)
(Symbol_9)
(Symbol_10)
)

(:action opt_0
:parameters ()
:precondition (and (Symbol_4) (Symbol_5))
:effect (and (Symbol_0) (Symbol_1) (not (Symbol_4))
            (not (Symbol_7)))
)

(:action opt_1
:parameters ()
:precondition (and (Symbol_4) (Symbol_5))
:effect (and (Symbol_0) (Symbol_2) (Symbol_3) (not (Symbol_5))
            (not (Symbol_7)))
```

```
)

(:action opt_2
:parameters ()
:precondition (and (Symbol_1) (Symbol_5) (Symbol_8))
:effect (and (Symbol_4) (not (Symbol_1)))
)

(:action opt_3
:parameters ()
:precondition (and (Symbol_3) (Symbol_4) (Symbol_8))
:effect (and (Symbol_5) (not (Symbol_3)))
)

(:action opt_4
:parameters ()
:precondition (and (Symbol_0) (Symbol_5) (Symbol_8)
                   (Symbol_10))
:effect (and (Symbol_6) (not (Symbol_8)))
)

(:action opt_5
:parameters ()
:precondition (and (Symbol_0) (Symbol_2) (Symbol_8)
                   (Symbol_10))
:effect (and (Symbol_6) (not (Symbol_8)))
)

(:action opt_6
:parameters ()
:precondition (and (Symbol_0) (Symbol_6) (Symbol_10))
:effect (and (Symbol_9) (Symbol_7) (Symbol_8)
             (not (Symbol_0)) (not (Symbol_6))
```

```
(not (Symbol_10)))
)


(:action opt_7
:parameters ()
:precondition (and (Symbol_9) (Symbol_8))
:effect (and (Symbol_10) (not (Symbol_9)))
))
```

B.4 ROBOTIC ARM SCENARIO: THE NEW SET OF SYMBOLS

```
Symbol s_0 (Symbol_0) Ref. Option = o_[0, 1]
Semantics: Object in sight
[0, 255]
[0, 255]
[0, 255]
[240, 255]
[0, 255]
[0, 255]
[0, 255]
[0, 255]
[0, 255]
[0, 255]

Symbol s_1 (Symbol_1) Ref. Option = o_[0]
Semantics: Rover is in zone_1
[0, 255]
[0, 255]
[0, 255]
[0, 255]
[0, 255]
[0, 255]
[0, 255]
[0, 255]
[240, 255]
[0, 255]

Symbol s_2 (Symbol_2) Ref. Option = o_[1]
Semantics: Object is VASE
[0, 255]
[0, 255]
```

```
[240, 255]
[0, 255]
[0, 255]
[0, 255]
[0, 255]
[0, 255]
[0, 255]
[0, 255]

Symbol s_3 (Symbol_3) Ref. Option = o_[1]
Semantics: Rover is in zone_2
[0, 255]
[0, 255]
[0, 255]
[0, 255]
[0, 255]
[0, 255]
[0, 255]
[0, 255]
[0, 255]
[240, 255]

Symbol s_4 (Symbol_4) Ref. Option = o_[2]
Semantics: Rover is NOT in zone_1
[0, 255]
[0, 255]
[0, 255]
[0, 255]
[0, 255]
[0, 255]
[0, 255]
[0, 255]
[0, 50]
```

```
[0, 255]

Symbol s_5 (Symbol_5) Ref. Option = o_[3]
Semantics: Rover is NOT in zone_2
[0, 255]
[0, 255]
[0, 255]
[0, 255]
[0, 255]
[0, 255]
[0, 255]
[0, 255]
[0, 255]
[0, 50]

Symbol s_6 (Symbol_6) Ref. Option = o_[4, 5]
Semantics: Object is grasped
[0, 255]
[0, 255]
[0, 255]
[0, 255]
[0, 255]
[0, 255]
[240, 255]
[0, 255]
[0, 255]
[0, 255]

Symbol s_7 (Symbol_7) Ref. Option = o_[6]
Semantics: Object NOT in sight
[0, 255]
[0, 255]
[0, 255]
```

```
[0, 50]
[0, 255]
[0, 255]
[0, 255]
[0, 255]
[0, 255]
[0, 255]

Symbol s_8 (Symbol_8) Ref. Option = o_[6]
Semantics: Object is NOT grasped
[0, 255]
[0, 255]
[0, 255]
[0, 255]
[0, 255]
[0, 255]
[0, 50]
[0, 255]
[0, 255]
[0, 255]

Symbol s_9 (Symbol_9) Ref. Option = o_[6]
Semantics: Object is stowed
[0, 255]
[0, 255]
[0, 255]
[0, 255]
[0, 255]
[0, 255]
[0, 255]
[240, 255]
[0, 255]
[0, 255]
```

```
Symbol s_10 (Symbol_10) Ref. Option = o_[7]
Semantics: Object is NOT stowed
[0, 255]
[0, 255]
[0, 255]
[0, 255]
[0, 255]
[0, 255]
[0, 255]
[0, 50]
[0, 255]
[0, 255]
```

C

EXPERIMENTS OF CHAPTER 4

In this chapter, additional details of the experiments of Chapter 4 are included. The tables included below show supplementary details on the results of each domain in terms of the number of symbols, operators and time.

It is worth noticing that this information is included for completeness. Still, it just marginally influences the results, given that the number of symbols and operators does not describe the quality of the representation. In our case, only the success of planning can say whether the representation is valid, which has been shown in Chapter 4.

	ACTION BABBLING – DOMAIN 1																															
	SYMBOLS										OPERATORS										TIME (in seconds)									
Cycle \ Trial	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	10
1	15	16	17	17	16	18	16	14	16	21	123	148	201	162	139	222	152	105	154	270	13	15	18	15	15	18	18	15	14	23
2	15	17	16	16	15	17	19	15	15	15	141	168	288	164	145	180	413	145	177	184	16	16	18	17	17	17	21	17	17	17
3	17	19	17	16	15	17	19	17	17	15	187	227	188	167	143	174	365	215	203	158	18	18	18	18	18	18	26	19	18	19
4	15	17	15	17	17	15	15	17	15	17	146	205	177	196	200	146	184	204	165	209	19	22	20	19	19	18	20	20	19	22
5	17	15	16	15	16	16	15	17	17	17	232	158	184	152	186	159	170	208	193	212	21	21	21	19	19	19	20	21	21	21
6	17	15	17	17	16	16	17	15	17	17	220	165	234	204	184	159	218	177	192	219	21	22	24	21	21	20	21	22	22	22
7	17	18	17	17	17	17	15	15	15	16	222	219	239	213	223	206	166	177	164	193	23	24	25	24	24	22	22	22	23	23
8	17	15	17	17	19	16	15	15	15	16	197	169	243	208	332	172	158	162	161	198	24	24	27	27	27	22	23	24	24	24
9	16	15	17	17	17	15	17	15	16	17	163	165	238	213	261	151	212	161	191	238	24	25	28	27	27	25	25	26	26	26
10	15	17	16	16	15	15	17	17	18	15	154	217	206	195	189	155	211	213	241	167	26	30	29	31	31	26	27	26	28	27
11	17	16	17	15	17	17	15	16	17	16	220	187	238	167	236	204	156	192	205	200	27	31	31	31	31	28	27	30	30	29
12	17	17	17	15	15	16	16	17	17	16	201	200	211	171	167	174	188	214	212	214	29	31	32	34	34	29	29	32	30	33
13	17	17	17	15	17	17	16	19	17	16	207	213	238	160	234	198	184	257	205	194	33	32	34	34	34	32	31	33	31	33
14	15	17	15	15	16	16	16	17	17	15	155	215	171	161	195	185	187	204	227	166	31	35	36	36	36	34	32	35	32	35
15	15	16	16	17	17	15	16	15	15	17	159	187	204	214	223	163	199	159	173	218	33	36	39	39	39	36	33	37	34	37

GOAL BABBLING – DOMAIN 1

	SYMBOLS										OPERATORS										TIME (in seconds)									
Cycle \ Trial	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	10
1	17	17	17	19	17	19	17	18	16	17	146	142	170	130	213	258	158	257	174	189	17	23	22	31	21	21	23	22	22	22
2	15	16	16	16	17	15	18	15	15	17	188	164	173	141	277	193	212	171	157	204	31	27	23	17	21	25	22	21	25	31
3	15	17	17	17	16	15	15	17	17	15	173	187	213	175	202	165	151	229	236	149	24	21	21	20	22	21	21	23	23	22
4	15	16	16	15	15	17	17	15	17	17	238	171	177	135	184	163	186	170	233	209	24	23	54	30	31	23	32	22	30	23
5	15	16	16	15	17	16	15	17	17	17	192	156	189	155	229	151	153	238	212	215	27	23	23	27	25	32	23	31	24	25
6	15	17	17	17	17	17	16	16	17	17	180	224	213	211	221	200	183	186	206	209	27	26	47	25	27	26	33	25	26	26
7	17	15	17	17	18	16	15	17	17	15	229	155	238	194	253	191	182	220	209	157	33	25	35	35	37	27	32	28	27	35
8	17	17	17	18	17	17	16	16	17	17	219	211	234	217	222	219	199	170	210	227	30	32	31	39	52	29	32	35	28	37
9	17	15	15	15	17	15	17	17	17	15	219	166	181	165	229	167	235	187	212	155	54	39	35	62	31	75	30	31	30	34
10	17	17	16	17	16	17	16	17	17	15	229	239	197	225	210	238	207	201	210	155	43	31	44	32	53	31	35	37	38	32
11	15	16	16	16	17	17	17	17	16	15	175	199	203	191	227	219	216	192	191	162	45	32	47	33	36	41	36	39	38	32
12	17	17	17	15	17	17	15	17	15	17	219	224	203	157	195	205	170	195	167	200	66	44	93	33	43	56	44	35	33	35
13	16	15	17	17	15	17	15	17	15	15	192	166	202	198	156	209	163	209	149	165	38	62	38	37	40	39	35	35	49	35
14	15	15	15	17	17	17	15	17	17	17	172	167	152	202	216	230	162	229	210	199	55	41	56	38	46	53	35	49	69	47
15	15	15	15	17	17	17	15	16	15	18	165	167	154	205	212	215	172	195	149	231	48	37	55	54	68	61	39	67	65	91

DISTANCE GOAL BABBLING – DOMAIN 1

	SYMBOLS										OPERATORS										TIME (in seconds)									
Cycle \ Trial	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	10
1	16	16	15	18	16	17	15	17	15	17	177	153	145	204	136	140	155	132	133	185	20	23	53	34	27	27	19	27	23	20
2	18	17	15	16	17	17	18	15	20	17	292	196	137	135	194	191	213	146	273	181	23	25	42	19	21	20	29	21	39	20
3	17	17	18	17	17	17	17	17	17	16	251	198	261	194	163	221	185	228	249	174	22	22	22	40	20	22	22	23	27	70
4	18	17	19	17	17	18	17	17	19	17	233	204	255	199	201	264	220	198	304	253	23	56	23	22	23	24	32	25	26	23
5	19	16	16	16	17	17	17	17	18	15	292	178	187	192	217	190	224	191	264	169	28	27	23	23	25	34	26	39	31	24
6	17	15	19	17	17	15	17	15	16	17	245	155	253	228	208	156	220	176	197	191	27	24	43	38	27	24	26	25	26	25
7	17	17	17	17	15	17	17	16	17	17	229	208	284	222	162	223	239	176	222	199	33	28	27	32	27	27	29	26	28	32
8	17	17	17	15	16	17	17	17	16	17	221	204	198	166	179	216	190	197	181	204	29	28	43	27	28	28	29	45	28	30
9	16	17	17	16	15	16	17	17	17	15	195	213	210	182	150	192	190	196	205	141	43	30	35	35	30	29	30	30	32	30
10	15	15	18	15	15	15	17	17	15	16	153	170	226	140	152	159	184	189	156	155	31	30	32	29	42	36	31	31	31	32
11	17	16	17	17	15	15	16	16	16	15	199	184	251	186	140	154	162	164	177	141	49	37	35	36	32	38	32	32	46	31
12	16	17	15	17	17	16	17	15	15	16	167	208	152	184	192	181	187	142	157	167	35	46	34	33	35	33	34	32	34	36
13	17	16	19	17	16	17	16	17	17	15	185	179	243	183	167	200	163	193	197	139	36	34	36	41	43	44	35	36	36	35
14	17	17	17	17	17	17	17	15	17	17	193	204	214	182	194	202	188	136	191	189	40	36	38	44	37	42	36	35	38	38
15	15	15	17	15	15	17	17	17	17	15	147	155	216	141	144	206	191	179	192	140	39	37	39	36	46	47	38	38	39	40

ACTION BABBLING – DOMAIN 2

	SYMBOLS										OPERATORS										TIME (in seconds)									
Cycle \ Trial	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	10
1	22	22	21	21	20	21	22	21	21	21	447	270	176	315	206	410	260	263	181	235	23	21	19	20	19	20	15	21	13	18
2	21	22	21	22	21	22	22	20	21	21	421	414	244	385	314	302	297	300	309	400	23	24	20	25	21	21	18	21	23	24
3	22	22	21	22	21	22	22	21	21	22	321	356	330	394	268	347	369	325	342	409	23	23	22	24	22	23	19	24	25	22
4	22	22	20	22	21	22	22	22	22	22	333	367	262	376	377	396	360	387	403	349	25	26	22	27	23	25	24	25	25	23
5	22	22	21	22	22	22	22	21	22	22	344	413	330	386	388	374	388	393	429	355	26	28	27	26	26	26	26	26	26	23
6	22	22	22	22	22	22	22	22	22	22	423	439	436	434	379	436	392	423	441	476	26	29	29	31	27	28	30	28	27	28
7	22	22	22	22	21	22	22	21	22	22	451	446	397	461	365	420	395	416	418	436	28	31	27	34	29	29	31	30	29	29
8	22	21	22	22	22	22	22	22	22	22	443	390	401	432	409	389	401	430	409	449	30	30	30	35	29	30	33	31	31	30
9	22	22	22	22	21	21	22	22	22	22	439	431	433	408	428	365	413	429	427	452	32	33	34	35	31	30	33	34	32	32
10	22	22	21	22	22	22	22	22	22	22	416	432	388	412	375	416	435	426	429	453	33	34	36	36	35	32	34	35	33	34
11	21	22	21	22	22	22	22	22	22	22	371	424	351	438	411	412	420	423	440	483	38	36	36	42	37	35	36	35	34	34
12	22	22	22	22	22	21	22	22	22	22	424	389	391	429	416	423	423	421	416	419	37	38	39	42	37	36	38	37	37	36
13	22	22	22	22	22	22	22	22	22	22	411	384	398	411	417	400	422	406	429	435	39	37	40	44	40	36	42	38	41	36
14	22	22	21	22	22	22	22	22	22	22	400	402	360	409	431	410	413	394	405	413	40	41	42	45	42	39	42	41	42	39
15	22	22	22	22	22	22	21	22	22	22	411	414	410	450	452	477	380	395	413	403	44	40	43	48	44	41	45	44	44	42

GOAL BABBLING – DOMAIN 2

	SYMBOLS										OPERATORS										TIME (in seconds)									
Cycle \ Trial	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	10
1	21	21	21	21	21	22	21	22	22	22	359	299	275	266	242	395	308	295	255	325	28	23	23	27	22	22	22	33	28	25
2	21	22	22	22	21	22	21	22	22	22	349	343	411	284	262	318	290	330	350	259	25	28	33	31	22	26	34	25	29	25
3	21	21	22	22	21	20	21	22	22	22	357	315	359	299	291	401	341	372	353	326	37	24	27	29	24	35	24	27	27	24
4	22	22	22	22	21	22	22	22	22	22	409	380	407	321	330	442	391	339	465	346	28	29	29	33	26	30	30	28	34	32
5	22	22	22	22	21	22	21	22	22	22	369	438	415	311	330	463	345	391	383	414	29	35	40	26	36	31	37	35	31	38
6	22	21	22	22	22	22	22	22	22	22	382	373	467	374	352	425	350	407	507	413	40	30	31	46	37	36	36	35	34	34
7	22	22	22	22	22	22	22	22	22	22	357	406	408	450	395	429	373	417	422	377	38	34	45	44	31	34	30	32	41	32
8	22	22	22	22	22	22	22	22	22	22	365	374	438	442	370	400	390	402	393	413	34	33	40	33	45	36	50	33	33	39
9	22	22	22	21	22	22	22	22	22	22	367	383	379	363	361	426	377	413	396	404	35	40	47	43	48	42	36	34	35	36
10	22	22	22	22	22	21	22	22	21	22	355	446	417	440	402	427	385	416	410	405	64	37	43	45	39	39	49	37	56	36
11	21	22	21	22	22	22	22	22	22	22	326	424	379	440	392	463	423	418	471	377	62	55	52	53	39	41	39	40	60	38
12	22	22	22	22	22	22	21	21	22	22	385	430	425	392	389	434	369	391	407	378	41	49	57	39	65	40	39	40	37	39
13	22	22	22	22	22	22	22	22	22	22	379	407	375	433	436	433	386	417	418	381	40	41	47	45	44	51	50	42	52	48
14	22	22	22	22	22	21	22	22	22	22	388	427	404	449	425	405	390	416	395	387	42	50	52	42	46	47	42	44	47	50
15	22	22	22	22	22	22	22	22	22	22	425	429	396	392	436	427	386	438	394	387	73	49	45	41	47	54	47	45	47	43

DISTANCE GOAL BABBLING – DOMAIN 2

	SYMBOLS										OPERATORS										TIME (in seconds)									
Cycle \ Trial	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	10
1	21	21	21	21	21	21	22	21	21	21	284	290	219	365	256	205	371	222	315	225	62	29	23	24	25	21	26	19	18	24
2	21	22	21	21	22	22	22	21	21	21	334	280	251	406	247	336	340	265	322	283	25	25	24	28	24	26	26	24	23	26
3	22	22	21	21	22	22	22	21	22	22	345	316	313	295	378	335	271	287	351	339	33	28	36	26	29	29	27	24	28	28
4	22	22	21	21	22	22	22	21	22	22	385	358	290	302	330	301	289	337	450	362	29	32	27	26	28	30	28	26	31	30
5	22	22	21	21	22	22	22	21	21	22	463	351	277	329	322	338	300	353	315	343	31	31	28	31	29	31	30	34	29	29
6	22	22	20	21	22	22	22	21	21	22	419	289	287	397	285	394	292	343	326	345	32	33	31	30	36	33	31	30	36	35
7	22	22	21	21	22	22	22	21	22	22	432	325	314	389	318	430	334	350	354	414	35	34	31	32	32	42	33	43	32	32
8	22	22	21	21	22	22	22	21	22	22	426	361	327	386	323	357	354	333	389	376	44	34	31	33	33	37	34	31	34	34
9	22	22	21	20	22	22	22	21	22	22	393	350	347	353	349	357	348	346	441	363	37	34	40	33	35	36	36	39	37	34
10	22	22	21	21	21	22	22	20	22	22	386	360	353	393	287	387	347	312	423	366	37	37	176	36	35	38	36	33	40	45
11	22	22	21	21	21	22	22	20	22	22	378	333	357	385	286	386	358	333	408	375	39	41	35	36	43	43	38	34	39	41
12	21	22	21	21	22	22	22	21	22	21	359	344	341	325	284	380	366	310	437	326	41	39	37	36	39	42	52	37	42	38
13	22	22	21	21	22	22	22	20	22	21	427	337	341	325	283	382	379	294	368	344	42	41	52	37	39	43	41	37	43	41
14	22	22	21	21	22	22	22	21	22	22	427	336	346	322	309	376	347	337	371	376	43	42	39	39	41	44	44	40	48	42
15	21	21	21	21	22	22	20	21	21	22	331	295	324	313	327	381	316	330	345	384	43	43	41	41	41	45	43	42	62	42

ACTION BABBLING – DOMAIN 3

	SYMBOLS										OPERATORS										TIME (in seconds)									
Cycle \ Trial	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	10
1	26	25	24	27	26	24	23	26	25	25	1024	451	602	537	683	348	371	510	417	444	54	59	47	66	90	37	59	42	38	43
2	26	28	25	25	26	24	26	25	25	26	1314	824	715	651	555	456	439	919	405	521	77	63	55	59	63	42	49	78	42	45
3	25	25	26	29	26	26	25	25	25	25	559	637	830	874	1142	602	495	693	504	529	64	72	73	78	80	62	65	67	63	60
4	26	27	28	28	26	26	26	26	26	26	728	707	980	922	637	869	1051	835	599	761	77	78	86	95	77	67	86	77	63	78
5	26	27	26	26	25	27	26	26	26	26	802	738	609	881	630	851	847	685	1257	933	88	87	83	93	65	87	95	80	85	94
6	29	27	24	25	26	26	26	26	27	25	1127	1209	1124	819	898	777	853	772	838	867	105	92	104	107	108	93	103	95	94	96
7	26	26	26	26	28	26	26	27	26	26	903	1036	1124	949	1029	971	1038	1008	1160	919	114	116	103	120	114	112	117	120	103	113
8	25	26	26	28	26	25	27	26	26	27	1115	1067	1150	1093	824	837	899	832	853	979	113	139	119	128	122	115	123	100	107	128
9	25	26	26	26	26	26	27	26	26	26	808	1029	866	963	706	877	872	878	963	1318	130	144	129	126	130	120	122	127	122	151
10	26	26	26	26	26	26	27	27	27	26	1195	994	1092	990	745	811	944	978	977	988	142	140	144	151	134	135	135	127	137	163
11	26	26	26	27	26	25	27	26	26	25	857	1004	886	1043	1034	750	1054	929	945	1221	149	170	148	158	145	142	155	134	148	162
12	26	27	26	26	26	26	25	26	26	26	1219	946	886	996	1014	852	894	1011	860	1337	168	179	168	172	160	153	162	147	154	195
13	26	25	27	26	28	26	26	26	27	26	1142	751	1034	830	1114	875	909	956	871	947	181	173	188	198	170	174	175	157	168	190
14	27	27	26	26	25	27	26	27	26	26	878	875	969	873	1134	935	817	1088	806	891	196	210	198	227	182	182	189	194	180	204
15	27	26	26	26	26	24	26	25	26	27	930	811	977	881	1210	730	826	1092	807	999	212	214	201	228	195	191	216	184	194	216

GOAL BABBLING – DOMAIN 3

	SYMBOLS										OPERATORS										TIME (in seconds)									
Cycle \ Trial	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	10
1	24	24	25	21	26	25	25	25	24	26	456	495	824	401	1025	575	432	422	662	625	169	51	70	54	120	70	117	69	63	78
2	26	26	24	26	26	25	25	26	26	26	809	524	626	764	599	635	731	621	730	585	74	67	110	87	72	75	93	85	90	54
3	26	26	25	23	25	26	25	27	25	26	809	515	698	457	463	691	689	546	623	634	99	88	81	70	102	88	111	86	97	105
4	26	26	25	26	26	26	25	27	25	26	891	1155	726	687	521	734	613	780	664	587	97	107	86	84	85	99	124	96	109	86
5	26	27	25	26	27	26	27	27	27	26	1152	785	668	847	904	643	918	733	1016	886	119	102	97	107	118	82	88	95	131	127
6	27	26	24	25	26	26	26	27	27	26	1138	633	664	694	1161	635	748	938	1059	1327	104	93	100	107	127	116	112	146	150	121
7	26	28	25	25	28	27	24	28	28	26	769	725	718	829	1042	758	638	1056	1107	1210	180	114	138	134	112	104	115	129	157	200
8	27	26	25	25	27	27	26	26	26	26	764	628	691	681	1355	681	827	955	944	822	142	123	137	133	129	114	137	121	160	178
9	27	26	26	26	26	26	25	25	25	25	765	649	751	754	824	681	1224	734	837	757	157	144	149	153	166	142	160	177	140	183
10	26	24	25	26	26	26	27	26	26	28	716	624	712	783	917	720	1094	885	966	1035	136	138	164	174	159	140	164	155	160	224
11	26	26	26	26	26	26	26	26	27	26	703	726	723	822	819	763	950	850	943	868	177	182	163	165	149	163	173	206	171	198
12	26	26	26	26	26	26	26	26	26	26	914	785	722	910	969	881	915	826	964	824	193	209	214	307	221	163	182	216	179	195
13	27	26	26	26	26	26	26	27	26	26	975	734	740	853	917	1262	918	892	850	838	348	204	180	252	187	301	219	208	219	217
14	26	28	25	26	26	28	26	25	26	24	825	853	756	890	845	1475	949	772	846	681	193	172	294	262	231	262	224	227	223	202
15	26	26	25	25	26	25	25	27	26	26	821	733	904	785	839	1198	864	886	1170	819	204	213	284	277	198	228	258	243	208	331

DISTANCE GOAL BABBLING – DOMAIN 3

	SYMBOLS										OPERATORS										TIME (in seconds)									
Cycle \ Trial	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	10
1	25	24	25	27	25	23	25	24	25	19	485	407	511	550	442	518	911	440	482	371	84	60	70	77	114	105	215	66	71	42
2	26	25	26	27	25	25	25	25	27	19	707	514	463	728	531	440	527	502	592	353	129	72	88	94	338	122	85	84	142	55
3	24	26	26	28	26	26	26	26	27	26	698	734	524	662	969	606	595	538	600	685	88	96	104	97	129	153	93	84	136	88
4	26	26	27	26	26	26	26	26	27	25	628	631	477	820	766	632	842	562	651	614	87	107	92	113	116	149	81	91	105	95
5	25	26	26	26	25	25	26	26	26	26	607	640	694	852	609	685	808	564	1048	663	131	105	101	180	116	99	103	102	181	107
6	25	28	26	26	27	27	26	26	26	26	574	986	745	926	715	1043	808	781	747	597	103	127	120	133	107	110	106	119	139	113
7	26	24	26	26	27	27	26	26	27	26	634	805	779	852	915	1002	788	752	777	752	119	132	329	137	129	112	175	129	152	125
8	26	25	26	25	26	25	26	26	25	26	586	830	869	1075	707	748	781	760	650	795	138	182	229	144	143	116	143	134	120	305
9	26	26	26	26	27	27	26	26	25	26	635	889	887	863	916	960	731	848	737	745	139	161	142	295	226	134	133	179	152	134
10	26	27	26	25	26	26	26	26	26	26	568	928	844	774	905	896	747	874	738	773	128	155	168	167	192	142	150	196	163	158
11	24	28	26	26	27	28	26	26	25	26	507	974	990	935	974	1109	758	841	669	761	150	173	162	183	188	159	155	179	189	157
12	25	26	26	27	27	26	26	27	25	28	547	921	979	914	894	972	766	866	713	961	142	175	212	232	201	195	185	207	156	268
13	26	26	26	26	28	27	26	26	24	26	587	873	962	872	1046	1044	755	882	716	820	191	183	205	263	194	173	260	249	177	172
14	28	25	26	26	27	27	26	26	25	26	791	911	934	875	956	1102	732	863	1012	824	231	209	201	277	204	190	206	226	320	236
15	27	26	26	28	26	27	26	26	26	27	728	881	983	1002	902	985	820	961	832	782	233	282	240	304	240	251	259	234	253	229

ACTION BABBLING – DOMAIN 4

	SYMBOLS										OPERATORS										TIME (in seconds)									
Cycle \ Trial	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	10
1	23	26	25	24	27	25	26	25	21	25	305	391	418	394	560	241	435	452	572	390	34	47	32	31	35	34	32	35	33	54
2	23	26	27	24	28	25	25	28	26	25	364	602	629	357	837	481	437	784	1113	502	34	64	42	32	53	47	39	76	65	46
3	23	26	26	24	26	25	25	24	26	27	341	426	769	570	723	473	743	740	612	1657	47	54	61	41	54	48	41	56	48	69
4	25	27	28	26	26	26	27	26	27	28	540	866	1473	787	1016	408	1110	695	859	831	53	79	88	66	78	55	63	74	78	73
5	26	27	26	26	27	26	27	27	27	28	681	694	1373	789	1458	550	652	483	953	986	66	84	94	71	96	62	68	80	76	94
6	26	27	28	27	27	28	27	27	28	28	592	577	1605	1048	1191	1119	672	966	812	1057	70	79	114	109	90	102	78	92	75	105
7	27	27	29	28	29	27	26	27	27	28	869	1228	1573	1111	1939	1170	845	747	929	1344	99	99	128	128	119	101	91	89	94	113
8	27	27	29	27	29	29	27	27	29	28	801	1246	1660	1078	1461	1353	1108	711	898	1192	87	109	125	133	112	108	94	99	99	113
9	28	27	29	27	28	27	27	28	27	28	1471	1408	1762	1012	1506	1352	1070	982	1091	1177	126	126	148	134	125	113	125	98	102	120
10	27	27	29	27	29	27	28	28	27	28	1134	1253	1793	1304	2305	1219	1418	1173	978	1241	148	134	136	157	163	112	124	121	112	118
11	28	27	28	27	28	27	28	28	27	28	859	1085	1688	1388	2107	1157	1517	1152	1347	1710	140	126	145	146	170	124	150	128	140	160
12	28	27	29	27	31	27	28	29	30	28	1139	1237	1786	1111	3033	1210	1580	1108	1533	1297	161	140	158	150	178	126	158	131	137	143
13	28	27	29	27	30	28	30	30	28	26	1173	1232	1601	1229	2399	1214	2244	1363	1312	1167	173	164	171	168	175	132	199	154	141	154
14	27	26	30	28	30	28	29	27	28	29	1124	1161	1727	1383	2444	1442	2171	1230	1302	1964	181	157	182	171	187	148	194	154	169	200
15	28	27	29	28	30	29	29	27	28	28	1131	1305	1578	1393	2346	1482	2244	970	986	1612	193	180	180	185	193	154	210	151	161	185

GOAL BABBLING – DOMAIN 4

	SYMBOLS										OPERATORS										TIME (in seconds)									
Cycle \ Trial	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	10
1	24	22	30	27	26	26	24	24	26	27	518	407	811	596	711	386	362	373	372	420	87	34	105	64	85	57	44	44	49	51
2	29	24	27	28	27	25	24	27	25	27	727	388	859	816	629	388	375	673	517	437	80	50	98	81	110	76	47	79	61	70
3	24	26	26	29	27	27	25	24	26	28	702	597	1038	913	793	926	599	439	620	552	62	68	137	209	86	71	109	49	66	68
4	27	24	25	29	29	27	27	27	26	27	758	408	982	740	1339	1073	649	768	710	550	83	61	81	99	101	81	85	96	94	70
5	28	27	26	29	28	28	28	28	28	28	915	712	1146	1305	1357	1030	985	1599	812	1149	109	104	83	125	100	106	130	112	112	107
6	29	27	27	29	28	28	26	28	28	27	1165	672	905	1277	970	952	838	1472	1170	698	110	141	114	139	123	145	163	128	91	109
7	29	27	27	29	28	28	28	28	28	27	964	562	1148	1438	1182	1401	1362	1449	758	799	106	95	119	166	174	142	147	132	159	170
8	30	27	29	30	29	28	30	28	29	27	1147	843	1478	1681	1206	1394	1839	1600	1202	713	140	110	156	203	134	141	140	160	144	141
9	29	26	27	30	28	28	29	28	29	27	1626	960	1318	1743	1157	1453	1563	1432	1405	1040	191	124	136	170	128	160	184	146	155	185
10	27	27	27	30	28	30	28	28	29	27	1688	977	1521	1768	955	1404	1782	1599	1528	1161	144	161	142	166	134	159	168	163	228	143
11	29	26	28	30	28	30	30	28	29	27	2232	942	1907	1782	962	1498	1895	1549	1588	1075	171	157	187	185	190	165	166	156	175	150
12	29	27	28	30	28	28	32	29	29	27	2067	1045	1447	1892	1105	1435	1674	1531	1622	1045	239	183	215	254	161	161	347	214	187	166
13	29	27	28	32	30	32	29	29	27	27	2215	1171	1428	2045	1223	1982	1620	1413	1216	1035	196	225	233	213	166	191	197	170	261	284
14	29	27	28	30	27	30	30	29	29	27	1785	1196	1324	1972	1022	1708	1693	1500	1624	1049	194	191	267	272	202	256	250	206	262	223
15	30	27	28	30	28	29	29	29	29	27	1981	1152	1299	1912	1155	1494	1322	1176	1581	1033	192	256	236	366	264	202	190	204	302	187

DISTANCE GOAL BABBLING – DOMAIN 4

	SYMBOLS										OPERATORS										TIME (in seconds)									
Cycle \ Trial	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	10
1	27	24	25	28	25	27	25	23	26	26	1229	1040	538	741	532	1487	629	338	293	665	72	215	163	64	108	174	61	44	43	78
2	28	27	28	27	26	27	27	27	26	26	843	800	918	566	641	878	1046	1389	696	408	115	309	203	240	271	269	108	72	60	79
3	28	28	27	27	27	27	28	26	27	27	738	643	770	1130	995	1050	887	713	932	730	179	196	296	150	385	75	237	229	128	180
4	28	29	28	27	28	27	29	26	26	27	756	737	935	1436	1716	1029	1370	703	1101	889	142	80	100	451	584	89	391	99	83	321
5	28	29	30	29	28	27	29	26	27	27	834	1410	1194	1139	1440	1006	1105	825	901	1071	136	521	170	205	545	93	416	316	101	98
6	28	28	28	28	27	28	27	26	27	26	830	1650	1064	803	1582	962	1179	786	1568	1185	395	127	115	131	530	124	147	308	118	101
7	28	29	28	29	29	27	29	26	28	28	1103	1327	1134	975	1361	935	1587	1061	1614	1498	125	174	145	388	586	118	156	178	125	149
8	28	29	28	28	28	27	29	27	29	27	1107	1425	1421	827	1352	1222	1523	952	1973	1338	133	155	161	369	615	481	173	279	167	119
9	28	29	28	28	28	27	29	28	30	27	1172	1422	1405	821	1312	1098	1400	940	2172	1289	147	510	153	189	607	481	193	143	163	146
10	28	29	28	28	30	27	28	28	29	27	1059	1410	1449	1064	1221	1085	1636	785	1664	1187	305	201	195	128	668	155	172	330	171	130
11	28	29	28	29	30	27	29	29	29	27	993	1634	1425	1119	1061	1341	1545	885	1661	1257	166	189	244	141	218	151	179	468	167	163
12	28	29	28	29	30	28	29	28	30	25	1007	1902	1426	1101	1812	1115	1572	804	1739	939	176	203	237	151	173	550	210	357	187	181
13	29	30	28	29	30	27	29	29	30	27	1243	1969	1309	1178	1472	960	1477	897	1542	1156	199	227	201	303	202	556	212	534	228	218
14	28	31	30	29	30	28	29	28	28	27	1084	1727	1496	1092	1419	1441	2221	938	1347	1341	225	259	267	173	213	770	244	502	203	170
15	27	29	28	29	30	28	29	29	30	26	1221	1458	1297	1535	1505	1542	1342	1148	2007	1046	252	227	259	204	228	203	233	293	216	185

ACTION BABBLING – DOMAIN 5

	SYMBOLS										OPERATORS										TIME (in seconds)									
Cycle \ Trial	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	10
1	31	32	29	33	30	34	31	30	35	33	1685	890	1394	1518	1063	1754	1526	1110	1636	2073	234	198	207	224	182	265	225	235	174	140
2	33	34	31	34	34	35	34	32	32	32	1585	1226	1855	1077	1398	1638	1772	1614	1099	1565	333	342	307	336	301	454	380	379	328	286
3	31	32	33	31	34	35	32	31	35	35	1363	1276	1930	1403	1178	1812	1229	1615	1662	1810	471	441	404	489	404	621	377	531	540	365
4	33	32	33	33	32	33	34	33	33	33	1921	1054	1478	1601	1492	1034	1163	1569	1408	1435	705	619	559	794	563	701	514	706	709	490
5	34	32	36	34	34	34	33	36	33	34	1752	1268	1609	2016	1737	1296	1092	1698	1610	1520	1063	923	674	1207	749	932	744	928	914	631
6	33	34	33	33	33	33	34	32	32	33	1467	1421	1443	2278	1238	1760	2278	1105	1474	1967	1290	1237	872	1363	912	1320	1040	1183	1195	960
7	34	34	32	32	32	31	36	35	34	34	1265	1523	1108	2185	1510	1371	2377	2034	2740	1803	1743	1581	1073	1697	1149	1502	1245	1506	1541	1174
8	32	32	33	33	34	33	32	34	34	32	1269	1433	1210	2273	2000	1495	1981	3164	2394	1929	2152	1975	1312	2036	1440	1834	1470	1912	1877	1501
9	32	31	33	32	32	32	34	34	34	36	1833	1640	1742	1957	1956	1495	2216	2358	2079	2605	2649	2160	1668	2432	1677	2182	1746	2278	2161	1765
10	33	32	33	33	32	34	34	34	34	32	1803	1792	2435	2261	1371	2066	2242	2383	2384	1991	3049	2901	1977	2838	2009	3127	2235	2705	2501	2037
11	34	31	32	35	33	33	32	33	34	33	1907	1751	1676	2664	1654	1753	2112	2037	2164	2140	3516	3030	2255	3291	2307	3614	2491	2864	2701	2370
12	34	32	34	33	34	33	34	33	33	33	2058	1955	2246	2081	2311	1913	2449	2004	2025	2511	3998	3421	2643	3699	2605	4211	2897	3544	3403	2857
13	34	32	32	33	32	32	32	36	34	33	2039	2015	1843	1945	1971	1602	2221	2731	2109	2412	4549	3966	3050	4215	3048	4633	3329	4233	3806	3231
14	33	35	33	33	33	32	32	35	33	33	1741	2230	2127	2126	2015	1820	2341	2608	2060	2267	4832	4384	3298	5060	3435	5381	3775	4907	4209	3600
15	31	32	33	33	35	32	34	33	32	32	1607	1929	2057	2172	2645	1830	2398	2091	2000	2001	5798	4837	3640	5593	3802	5920	4102	5493	4629	3971

GOAL BABBLING – DOMAIN 5

	SYMBOLS										OPERATORS										TIME (in seconds)									
Cycle \ Trial	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	10
1	32	30	34	32	32	33	35	33	34	34	1672	1240	1446	1317	1255	1743	1498	1569	1178	1443	289	178	1005	239	306	951	320	378	1030	1008
2	33	32	32	32	35	33	33	31	32	34	1967	1698	1302	1408	1675	1822	1699	1536	1379	1664	908	341	405	446	330	459	387	481	659	507
3	32	29	35	33	34	31	32	35	35	32	2456	1101	1793	1680	1866	1587	1546	1804	1819	2628	585	410	712	617	508	603	573	742	1104	511
4	33	34	34	34	31	33	34	33	32	34	2525	1748	1270	1423	1465	1747	2066	1423	1491	2729	1031	740	908	831	668	994	812	977	711	675
5	33	32	32	34	35	31	34	33	35	35	2144	1836	1686	1954	2242	1267	1611	2301	1971	2718	1403	946	742	1421	1131	1443	935	1400	1015	911
6	32	31	33	34	33	32	33	35	32	34	1950	974	1446	1715	2134	1350	1828	2615	1557	2431	1443	1304	1121	1471	1310	1626	1225	1752	1093	1209
7	33	32	33	34	32	33	35	33	34	31	2479	1357	1764	1414	1672	2281	2190	2093	1807	1915	1934	1618	1611	1929	1414	2151	1513	2365	1306	1493
8	33	32	34	34	32	33	32	32	32	33	2174	1385	1432	2170	1426	1506	2043	2042	1205	2083	2385	1906	1746	2630	1660	2461	1991	2815	1428	1785
9	32	32	34	32	36	32	35	32	33	32	2004	985	2677	2639	1615	1365	2603	2030	1377	1943	2739	2248	2404	3094	2443	2935	2508	3479	1825	2526
10	33	32	33	32	34	33	35	32	34	32	2022	1199	2512	1714	2045	1909	2674	2302	1870	2101	2909	2838	2342	3360	2559	3971	3217	3786	2445	2297
11	34	35	34	32	35	34	33	35	34	32	2202	1693	2190	1816	1942	2042	2084	2429	2105	2061	3371	3135	2553	4003	2767	4333	3996	4782	3509	2811
12	33	34	31	31	33	32	33	35	35	33	1946	2014	1633	1883	1770	1800	2095	2496	2248	2148	4246	3455	3263	4448	3243	5252	3432	5283	3636	3958
13	34	33	35	36	33	35	33	33	31	34	2364	1907	2330	2242	1871	2261	2036	2030	1880	2305	4653	4669	3863	5438	4403	5452	3736	4785	3471	3505
14	32	35	33	33	33	34	35	34	31	32	1854	2650	2050	1755	1811	1830	2471	2172	1945	2008	7763	4722	5190	5863	3827	6077	5730	5451	4468	4754
15	34	33	34	32	32	32	35	35	33	35	2247	2134	2254	1987	1388	1666	2569	2453	1852	2544	6581	5709	4410	6326	4547	7229	5228	7621	4174	7039

DISTANCE GOAL BABBLING – DOMAIN 5

	SYMBOLS										OPERATORS										TIME (in seconds)									
Cycle \ Trial	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	10
1	34	33	32	35	35	32	32	33	33	32	1289	1640	810	1663	1603	2588	1686	923	1020	1235	401	453	946	1078	998	343	296	1058	1049	561
2	36	33	34	32	33	32	33	33	33	34	2055	1241	2572	1762	1321	1880	1779	1458	1139	1354	661	393	702	1374	751	544	414	1286	1173	483
3	34	32	34	34	35	34	32	32	33	32	1513	1539	2607	2194	1585	1273	1434	2025	1566	1151	651	465	1119	806	459	1735	1061	1569	1119	529
4	33	34	33	32	35	35	34	35	33	33	1748	1786	1849	1642	2125	1544	1579	2674	1887	1643	1283	859	741	901	764	893	754	922	723	878
5	33	33	34	33	35	34	33	31	35	32	2131	1322	2342	2050	2016	1275	1529	2005	1479	2271	1433	763	1058	1209	922	1021	1101	1272	959	1866
6	33	34	32	33	33	32	33	32	33	33	2328	2129	1735	2043	2221	1166	1474	2227	1238	1384	1909	1152	1370	1560	1210	1368	1302	1357	1701	1155
7	32	33	33	35	32	32	34	32	35	34	2086	1986	1770	2367	2206	1299	1905	2199	1575	1292	2088	1199	1797	2081	1550	1741	2037	1619	1689	1663
8	33	33	33	32	33	34	33	32	31	32	2057	2165	1874	1969	2172	1782	1739	2107	1212	1143	2142	1487	2060	2455	2049	1982	1825	1910	1742	1719
9	31	32	33	34	32	32	33	32	32	33	1468	2051	2026	2240	1948	1725	2261	2248	1863	1209	2658	1834	2470	3105	2171	2738	2285	2230	2313	2055
10	34	34	33	33	32	34	32	33	32	33	1773	2556	1766	2106	2131	2369	2161	2461	1446	1469	3395	2316	2889	3239	2526	3149	2756	2642	2269	2498
11	32	32	32	33	33	32	34	35	32	34	1635	2039	1840	2136	2192	2098	2390	2715	1477	1797	4010	2042	3381	3787	2923	3412	2972	3324	2995	2855
12	34	34	35	33	32	32	34	34	33	34	2533	2226	2360	2078	2126	2124	2324	2524	1677	2052	3904	2398	3803	4185	3736	4368	4355	3723	3421	3284
13	32	34	34	32	34	33	31	33	32	34	1907	2236	2641	2162	2323	2279	1809	2310	1694	2379	4701	2575	3881	4613	3793	4556	3665	3725	3765	3906
14	33	35	33	35	33	33	32	33	32	33	2009	2350	2560	2728	2031	2286	2048	2250	1530	2363	4919	3406	4767	4584	4087	5992	4197	4070	4138	4495
15	32	32	34	33	34	34	32	31	32	32	2189	1879	2398	2180	2383	2350	2196	1989	1910	2011	5188	3587	5049	5718	4440	5818	4924	4644	4333	4762

BIBLIOGRAPHY

[1] Leonardo Amado, Ramon Fraga Pereira, Joao Aires, Mauricio Magnaguagno, Roger Granada, and Felipe Meneguzzi. "Goal recognition in latent space." In: *2018 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2018, pp. 1–8.

[2] Masataro Asai. "Unsupervised grounding of plannable first-order logic representation from images." In: *Proceedings of the International Conference on Automated Planning and Scheduling*. Vol. 29. 2019, pp. 583–591.

[3] Masataro Asai and Alex Fukunaga. "Classical Planning in Deep Latent Space: Bridging the Subsymbolic-Symbolic Boundary." In: *Proceedings of the AAAI Conference on Artificial Intelligence* 32.1 (2018). DOI: 10.1609/aaai.v32i1.12077. URL: https://ojs.aaai.org/index.php/AAAI/article/view/12077.

[4] Akhil Bagaria, Jason K Senthil, and George Konidaris. "Skill Discovery for Exploration and Planning using Deep Skill Graphs." In: *Proceedings of the 38th International Conference on Machine Learning*. Ed. by Marina Meila and Tong Zhang. Vol. 139. Proceedings of Machine Learning Research. PMLR, 2021, pp. 521–531. URL: https://proceedings.mlr.press/v139/bagaria21a.html.

[5] Gianluca Baldassarre and Marco Mirolli. *Intrinsically Motivated Learning in Natural and Artificial Systems*. Springer Science & Business Media, 2013.

[6] Andrew G Barto and Sridhar Mahadevan. "Recent advances in hierarchical reinforcement learning." In: *Discrete event dynamic systems* 13.1 (2003), pp. 41–77.

[7] Andrew Barto, Marco Mirolli, and Gianluca Baldassarre. "Novelty or surprise?" In: *Frontiers in psychology* 4 (2013), p. 907.

[8] Marc G Bellemare, Salvatore Candido, Pablo Samuel Castro, Jun Gong, Marlos C Machado, Subhodeep Moitra, Sameera S Ponda, and Ziyu Wang. "Autonomous navigation of stratospheric balloons using reinforcement learning." In: *Nature* 588.7836 (2020), pp. 77–82.

[9] Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. "Unifying count-based exploration and intrinsic motivation." In: *Advances in neural information processing systems* 29 (2016).

[10] Yoshua Bengio, Ian Goodfellow, and Aaron Courville. *Deep learning*. Vol. 1. MIT press Cambridge, MA, USA, 2017.

[11] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. "Curriculum learning." In: *Proceedings of the 26th annual international conference on machine learning*. 2009, pp. 41–48.

[12] Daniel E Berlyne. "Novelty and curiosity as determinants of exploratory behaviour." In: *British journal of psychology* 41.1 (1950), p. 68.

[13] Daniel E Berlyne. "Curiosity and Exploration: Animals spend much of their time seeking stimuli whose significance raises problems for psychology." In: *Science* 153.3731 (1966), pp. 25–33.

[14] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. "Dota 2 with large scale deep reinforcement learning." In: *arXiv preprint arXiv:1912.06680* (2019).

[15] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*. Vol. 4. 4. Springer, 2006.

[16] Sebastian Blaes, Marin Vlastelica Pogančić, Jiajie Zhu, and Georg Martius. "Control what you can: Intrinsically motivated task-planning agent." In: *Advances in Neural Information Processing Systems* 32 (2019).

[17] R. Peter Bonasso, R. James Firby, Erann Gat, David Kortenkamp, David P. Miller, and Mark G. Slack. "Experiences with an architecture for intelligent, reactive agents." In: *Journal of Experimental & Theoretical Artificial Intelligence* 9.2-3 (1997), pp. 237–256. DOI: 10.1080/095281397147103.

eprint: https://doi.org/10.1080/095281397147103. URL: https://doi.org/10.1080/095281397147103.

[18] Blai Bonet and Héctor Geffner. "mGPT: A probabilistic planner based on heuristic search." In: *Journal of Artificial Intelligence Research* 24 (2005), pp. 933–944.

[19] Robert A Butler. "Discrimination learning by rhesus monkeys to visual-exploration motivation." In: *Journal of Comparative and Physiological Psychology* 46.2 (1953), p. 95.

[20] Murray Campbell, A.Joseph Hoane, and Feng hsiung Hsu. "Deep Blue." In: *Artificial Intelligence* 134.1 (2002), pp. 57–83. ISSN: 0004-3702. DOI: https://doi.org/10.1016/S0004-3702(01)00129-1. URL: https://www.sciencedirect.com/science/article/pii/S0004370201001291.

[21] Michael Cashmore, Maria Fox, Derek Long, Daniele Magazzeni, Bram Ridder, Arnau Carrera, Narcis Palomeras, Natalia Hurtos, and Marc Carreras. "ROSPlan: Planning in the Robot Operating System." In: *ICAPS 2015, the 25th International Conference on Automated Planning and Scheduling*. 2015. URL: https://www.aaai.org/ocs/index.php/ICAPS/ICAPS15/paper/view/10619.

[22] Yanqing Chen. "Mechanisms of winner-take-all and group selection in neuronal spiking networks." In: *Frontiers in computational neuroscience* 11 (2017), p. 20.

[23] Nuttapong Chentanez, Andrew G Barto, and Satinder P Singh. "Intrinsically motivated reinforcement learning." In: *Advances in neural information processing systems*. 2005, pp. 1281–1288.

[24] Rohan Chitnis, Tom Silver, Joshua B Tenenbaum, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. "Glib: Efficient exploration for relational model-based reinforcement learning via goal-literal babbling." In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 13. 2021, pp. 11782–11791.

[25] Cédric Colas, Pierre Fournier, Mohamed Chetouani, Olivier Sigaud, and Pierre-Yves Oudeyer. "CURIOUS: intrinsically motivated modular

multi-goal reinforcement learning." In: *International conference on machine learning*. PMLR. 2019, pp. 1331–1340.

[26] Corinna Cortes and Vladimir Vapnik. "Support-Vector Networks." In: *Mach. Learn.* 20.3 (Sept. 1995), 273–297. ISSN: 0885-6125.

[27] S. Doncieux, D. Filliat, N. Díaz-Rodríguez, T. Hospedales, R. Duro, A. Coninx, D. M. Roijers, B. Girard, N. Perrin, and O. Sigaud. "Open-ended learning: a conceptual framework based on representational redescription." In: *Frontiers in neurorobotics* 12 (2018), p. 59.

[28] Emrah Düzel, Nico Bunzeck, Marc Guitart-Masip, and Sandra Düzel. "NOvelty-related motivation of anticipation and exploration by dopamine (NOMAD): implications for healthy aging." In: *Neuroscience & Biobehavioral Reviews* 34.5 (2010), pp. 660–669.

[29] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise." In: *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*. KDD'96. Portland, Oregon: AAAI Press, 1996, 226–231.

[30] Sébastien Forestier, Rémy Portelas, Yoan Mollard, and Pierre-Yves Oudeyer. "Intrinsically motivated goal exploration processes with automatic curriculum learning." In: *arXiv preprint arXiv:1708.02190* (2017).

[31] Mikhail Frank, Jürgen Leitner, Marijn Stollenga, Alexander Förster, and Jürgen Schmidhuber. "Curiosity driven reinforcement learning for motion planning on humanoids." In: *Frontiers in neurorobotics* 7 (2014), p. 25.

[32] Marianna B Ganapini, Murray Campbell, Francesco Fabiano, Lior Horesh, Jon Lenchner, Andrea Loreggia, Nicholas Mattei, Taher Rahgooy, Francesca Rossi, Biplav Srivastava, et al. "Combining Fast and Slow Thinking for Human-like and Efficient Navigation in Constrained Environments." In: *arXiv preprint arXiv:2201.07050* (2022).

[33] M. Garnelo and M. Shanahan. "Reconciling deep learning with symbolic artificial intelligence: representing objects and relations." In: *Current Opinion in Behavioral Sciences* 29 (2019). SI: 29: Artificial Intelligence (2019),

pp. 17 –23. ISSN: 2352-1546. DOI: https://doi.org/10.1016/j.cobeha.2018.12.010. URL: http://www.sciencedirect.com/science/article/pii/S2352154618301943.

[34] Marta Garnelo, Kai Arulkumaran, and Murray Shanahan. "Towards deep symbolic reinforcement learning." In: *arXiv preprint arXiv:1609.05518* (2016).

[35] Erann Gat. "Three-layer Architectures." In: *Artificial Intelligence and Mobile Robots*. Ed. by David Kortenkamp, R. Peter Bonasso, and Robin Murphy. Cambridge, MA, USA: MIT Press, 1998, pp. 195–210. ISBN: 0-262-61137-6. URL: http://dl.acm.org/citation.cfm?id=292092.292130.

[36] Michael R Genesereth and Nils J Nilsson. *Logical foundations of artificial intelligence*. Morgan Kaufmann, 2012.

[37] M. Ghallab, A. Howe, C. Knoblock, D. Mcdermott, A. Ram, M. Veloso, D. Weld, and D. Wilkins. *PDDL—The Planning Domain Definition Language*. 1998. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.37.212.

[38] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning: theory and practice*. Elsevier, 2004.

[39] Ben Goertzel. "Artificial general intelligence: concept, state of the art, and future prospects." In: *Journal of Artificial General Intelligence* 5.1 (2014), p. 1.

[40] Harry F Harlow. "Learning and satiation of response in intrinsically motivated complex puzzle performance by monkeys." In: *Journal of comparative and physiological psychology* 43.4 (1950), p. 289.

[41] Todd Hester and Peter Stone. "Intrinsically motivated model learning for developing curious robots." In: *Artificial Intelligence* 247 (2017). Special Issue on AI and Robotics, pp. 170 –186. ISSN: 0004-3702. DOI: https://doi.org/10.1016/j.artint.2015.05.002. URL: http://www.sciencedirect.com/science/article/pii/S0004370215000764.

[42] Andreas Holzinger, Anna Saranti, Christoph Molnar, Przemyslaw Biecek, and Wojciech Samek. "Explainable AI methods-a brief overview." In: *xxAI-Beyond Explainable AI: International Workshop, Held in Conjunction with ICML 2020, July 18, 2020, Vienna, Austria, Revised and Extended Papers*. Springer. 2022, pp. 13–38.

[43] Timothy Hospedales, Antreas Antoniou, Paul Micaelli, and Amos Storkey. "Meta-learning in neural networks: A survey." In: *IEEE transactions on pattern analysis and machine intelligence* 44.9 (2021), pp. 5149–5169.

[44] Julian Ibarz, Jie Tan, Chelsea Finn, Mrinal Kalakrishnan, Peter Pastor, and Sergey Levine. "How to train your robot with deep reinforcement learning: lessons we have learned." In: *The International Journal of Robotics Research* 40.4-5 (2021), pp. 698–721.

[45] A. J. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor, and S. Schaal. "Dynamical Movement Primitives: Learning Attractor Models for Motor Behaviors." In: *Neural Computation* 25.2 (2013), pp. 328–373. DOI: 10.1162/NECO_a_00393.

[46] Félix Ingrand, Simon Lacroix, Solange Lemai-Chenevier, and Frederic Py. "Decisional autonomy of planetary rovers." In: *Journal of Field Robotics* 24.7 (2007), pp. 559–580. DOI: 10.1002/rob.20206. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/rob.20206. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.20206.

[47] William Koch, Renato Mancuso, Richard West, and Azer Bestavros. "Reinforcement learning for UAV attitude control." In: *ACM Transactions on Cyber-Physical Systems* 3.2 (2019), pp. 1–21.

[48] N. Koenig and A. Howard. "Design and use paradigms for Gazebo, an open-source multi-robot simulator." In: *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*. Vol. 3. 2004, 2149–2154 vol.3. DOI: 10.1109/IROS.2004.1389727.

[49] George Konidaris. "On the necessity of abstraction." In: *Current Opinion in Behavioral Sciences* 29 (2019). Artificial Intelligence, pp. 1–7. ISSN: 2352-1546. DOI: https://doi.org/10.1016/j.cobeha.2018.11.

005. URL: https://www.sciencedirect.com/science/article/pii/S2352154618302080.

[50] George Konidaris and Andrew G Barto. "Skill discovery in continuous reinforcement learning domains using skill chaining." In: *Advances in neural information processing systems*. 2009, pp. 1015–1023.

[51] George Konidaris, Leslie Pack Kaelbling, and Tomas Lozano-Perez. "From skills to symbols: Learning symbolic representations for abstract high-level planning." In: *Journal of Artificial Intelligence Research* 61 (2018), pp. 215–289.

[52] L. Kunze, N. Hawes, T. Duckett, M. Hanheide, and T. Krajník. "Artificial Intelligence for Long-Term Robot Autonomy: A Survey." In: *IEEE Robotics and Automation Letters* 3.4 (2018), pp. 4023–4030. ISSN: 2377-3766. DOI: 10.1109/LRA.2018.2860628.

[53] Ray Kurzweil. *The singularity is near*. Springer, 2014.

[54] John E Laird, Allen Newell, and Paul S Rosenbloom. "Soar: An architecture for general intelligence." In: *Artificial intelligence* 33.1 (1987), pp. 1–64.

[55] Christian Lebiere and John R Anderson. "A connectionist implementation of the ACT-R production system." In: (1993).

[56] Marlos C Machado, Marc G Bellemare, and Michael Bowling. "A laplacian framework for option discovery in reinforcement learning." In: *arXiv preprint arXiv:1703.00956* (2017).

[57] Drew Mcdermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. *PDDL - The Planning Domain Definition Language*. Tech. rep. CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, 1998. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.37.212.

[58] Hiroyuki Mikita, Haseru Azuma, Yohei Kakiuchi, Kei Okada, and Masayuki Inaba. "Interactive symbol generation of task planning for daily assistive robot." In: *2012 12th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2012)*. IEEE. 2012, pp. 698–703.

[59] Marco Mirolli and Gianluca Baldassarre. "Functions and mechanisms of intrinsic motivations: The knowledge versus competence distinction." In: *Intrinsically motivated learning in natural and artificial systems* (2013), pp. 49–72.

[60] Kay C Montgomery. "The role of the exploratory drive in learning." In: *Journal of Comparative and Physiological Psychology* 47.1 (1954), p. 60.

[61] Dana Nau, Malik Ghallab, and Paolo Traverso. *Automated Planning: Theory & Practice*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2004. ISBN: 1558608567.

[62] Hai Nguyen and Hung La. "Review of deep reinforcement learning for robot manipulation." In: *2019 Third IEEE International Conference on Robotic Computing (IRC)*. IEEE. 2019, pp. 590–595.

[63] Remi Niel and Marco A Wiering. "Hierarchical Reinforcement Learning for Playing a Dynamic Dungeon Crawler Game." In: *2018 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE. 2018, pp. 1159–1166.

[64] Nils J Nilsson. *Principles of artificial intelligence*. Springer Science & Business Media, 1982.

[65] Angelo Oddi, Riccardo Rasconi, Emilio Cartoni, Gabriele Sartor, Gianluca Baldassarre, and Vieri Giuliano Santucci. *Learning High-Level Planning Symbols from Intrinsically Motivated Experience*. arXiv:1907.08313. 2019.

[66] Angelo Oddi, Riccardo Rasconi, Vieri Giuliano Santucci, Gabriele Sartor, Emilio Cartoni, Francesco Mannella, and Gianluca Baldassarre. "Integrating open-ended learning in the sense-plan-act robot control paradigm." In: *ECAI 2020*. IOS Press, 2020, pp. 2417–2424.

[67] Junhyuk Oh, Xiaoxiao Guo, Honglak Lee, Richard L Lewis, and Satinder Singh. "Action-conditional video prediction using deep networks in atari games." In: *Advances in neural information processing systems* 28 (2015).

[68] Pierre-Yves Oudeyer and Frederic Kaplan. "What is intrinsic motivation? A typology of computational approaches." In: *Frontiers in neurorobotics* (2009), p. 6.

[69] Pierre-Yves Oudeyer, Frédéric Kaplan, and Verena Hafner. "Intrinsic motivation systems for autonomous mental development." In: *IEEE transactions on evolutionary computation* 11.2 (2007), pp. 265–286.

[70] Simone Parisi, Victoria Dean, Deepak Pathak, and Abhinav Gupta. "Interesting object, curious agent: Learning task-agnostic exploration." In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 20516–20530.

[71] Emanuel Parzen. "On Estimation of a Probability Density Function and Mode." In: *The Annals of Mathematical Statistics* 33.3 (1962), pp. 1065–1076. ISSN: 00034851.

[72] Fabian Pedregosa et al. "Scikit-Learn: Machine Learning in Python." In: *J. Mach. Learn. Res.* 12.null (Nov. 2011), 2825–2830. ISSN: 1532-4435.

[73] Morgan Quigley, Ken Conley, Brian P. Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. "ROS: an open-source Robot Operating System." In: *ICRA Workshop on Open Source Software*. 2009.

[74] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993. ISBN: 1-55860-238-0.

[75] Charan Ranganath and Gregor Rainer. "Neural mechanisms for detecting and remembering novel events." In: *Nature Reviews Neuroscience* 4.3 (2003), pp. 193–202.

[76] Peter Redgrave and Kevin Gurney. "The short-latency dopamine signal: a role in discovering novel actions?" In: *Nature reviews neuroscience* 7.12 (2006), pp. 967–975.

[77] Francisco J Rodríguez-Lera, Francisco Martín-Rico, and Vicente Matellán-Olivera. "Generating symbolic representation from sensor data: inferring knowledge in robotics competitions." In: *2018 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*. IEEE. 2018, pp. 261–266.

[78] Alejandro Romero, Gianluca Baldassarre, Richard J Duro, and Vieri Giuliano Santucci. "Analysing autonomous open-ended learning of skills with different interdependent subgoals in robots." In: *2021 20th International Conference on Advanced Robotics (ICAR)*. IEEE. 2021, pp. 646–651.

[79] Alejandro Romero, Francisco Bellas, and Richard J Duro. "A Perspective on Lifelong Open-Ended Learning Autonomy for Robotics through Cognitive Architectures." In: *Sensors* 23.3 (2023), p. 1611.

[80] Murray Rosenblatt. "Remarks on Some Nonparametric Estimates of a Density Function." In: *The Annals of Mathematical Statistics* 27.3 (1956), pp. 832–837. ISSN: 00034851.

[81] Stuart J Russell. *Artificial intelligence a modern approach*. Pearson Education, Inc., 2010.

[82] Richard M. Ryan and Edward L. Deci. "Intrinsic and Extrinsic Motivations: Classic Definitions and New Directions." In: *Contemporary Educational Psychology* 25.1 (2000), pp. 54 –67. ISSN: 0361-476X. DOI: https://doi.org/10.1006/ceps.1999.1020. URL: http://www.sciencedirect.com/science/article/pii/S0361476X99910202.

[83] Vieri G Santucci, Gianluca Baldassarre, and Marco Mirolli. "Autonomous selection of the "what" and the "how" of learning: an intrinsically motivated system tested with a two armed robot." In: *Development and Learning and Epigenetic Robotics (ICDL-Epirob), 2014 Joint IEEE International Conferences on*. IEEE. 2014, pp. 434–439.

[84] Vieri Giuliano Santucci, Gianluca Baldassarre, and Emilio Cartoni. "Autonomous Reinforcement Learning of Multiple Interrelated Tasks." In: *Joint IEEE 9th International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob2019)*. 2019, pp. 221–227.

[85] Vieri Giuliano Santucci, Gianluca Baldassarre, and Marco Mirolli. "Intrinsic motivation signals for driving the acquisition of multiple tasks: a simulated robotic study." In: *Proceedings of the 12th International Conference on Cognitive Modelling (ICCM)*. 2013.

[86] Vieri Giuliano Santucci, Gianluca Baldassarre, and Marco Mirolli. "Which is the best intrinsic motivation signal for learning multiple skills?" In: *Frontiers in neurorobotics* 7 (2013), p. 22.

[87] Vieri Giuliano Santucci, Gianluca Baldassarre, and Marco Mirolli. "GRAIL: A Goal-Discovering Robotic Architecture for Intrinsically-Motivated Learning." In: *IEEE Transactions on Cognitive and Developmental Systems* 8.3 (2016), pp. 214–231. DOI: 10.1109/TCDS.2016.2538961.

[88] Vieri Giuliano Santucci, Pierre-Yves Oudeyer, Andrew Barto, and Gianluca Baldassarre. "Intrinsically motivated open-ended learning in autonomous robots." In: *Frontiers in neurorobotics* 13 (2020), p. 115.

[89] Gabriele Sartor, Davide Zollo, Marta Cialdea Mayer, Angelo Oddi, Riccardo Rasconi, and Vieri Giuliano Santucci. "Option Discovery for Autonomous Generation of Symbolic Knowledge." In: *International Conference of the Italian Association for Artificial Intelligence*. Springer. 2022, pp. 153–167.

[90] Gabriele Sartor, Davide Zollo, Marta Cialdea Mayer, Angelo Oddi, Riccardo Rasconi, and Vieri Giuliano Santucci. "Autonomous Generation of Symbolic Knowledge via Option Discovery." In: *Proceedings of the 9th Italian workshop on Planning and Scheduling (IPS'21) and the 28th International Workshop on "Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion" (RCRA'21)*. Vol. 3065. CEUR Workshop Proceedings. CEUR-WS.org, 2021.

[91] Stefan Schaal. "Dynamic Movement Primitives -A Framework for Motor Control in Humans and Humanoid Robotics." In: *Adaptive Motion of Animals and Machines*. Ed. by Hiroshi Kimura, Kazuo Tsuchiya, Akio Ishiguro, and Hartmut Witte. Tokyo: Springer Tokyo, 2006, pp. 261–280. ISBN: 978-4-431-31381-6. DOI: 10.1007/4-431-31381-8_23. URL: https://doi.org/10.1007/4-431-31381-8_23.

[92] Jürgen Schmidhuber. "Formal theory of creativity, fun, and intrinsic motivation (1990–2010)." In: *IEEE transactions on autonomous mental development* 2.3 (2010), pp. 230–247.

[93] Kristsana Seepanomwan, Vieri Giuliano Santucci, and Gianluca Baldassarre. "Intrinsically Motivated Discovered Outcomes Boost User's Goals Achievement in a Humanoid Robot." In: *2017 Joint IEEE International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob)*. 2017, pp. 178–183.

[94] Bruno Castro da Silva, George Konidaris, and Andrew G Barto. "Learning Parameterized Skills." In: (2012).

[95] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. "Mastering the game of Go with deep neural networks and tree search." In: *nature* 529.7587 (2016), pp. 484–489.

[96] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play." In: *Science* 362.6419 (2018), pp. 1140–1144.

[97] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. "Mastering the game of go without human knowledge." In: *nature* 550.7676 (2017), pp. 354–359.

[98] David Silver, Satinder Singh, Doina Precup, and Richard S. Sutton. "Reward is enough." In: *Artificial Intelligence* 299 (2021), p. 103535. ISSN: 0004-3702. DOI: https://doi.org/10.1016/j.artint.2021.103535. URL: https://www.sciencedirect.com/science/article/pii/S0004370221000862.

[99] Satinder Singh, Andrew G. Barto, and Nuttapong Chentanez. "Intrinsically Motivated Reinforcement Learning." In: *Proceedings of the 17th International Conference on Neural Information Processing Systems*. NIPS'04. Vancouver, British Columbia, Canada: MIT Press, 2004, 1281–1288.

[100] Freek Stulp, Gennaro Raiola, Antoine Hoarau, Serena Ivaldi, and Olivier Sigaud. "Learning compact parameterized skills with a single regression." In: (2013), pp. 417–422.

[101] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[102] Richard S. Sutton, Doina Precup, and Satinder Singh. "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning." In: *Artificial Intelligence* 112.1 (1999), pp. 181–211. ISSN: 0004-3702.

[103] Vivek Veeriah, Tom Zahavy, Matteo Hessel, Zhongwen Xu, Junhyuk Oh, Iurii Kemaev, Hado P van Hasselt, David Silver, and Satinder Singh. "Discovery of options via meta-learned subgoals." In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 29861–29873.

[104] Jeannette M. Wing. "Trustworthy AI." In: *Commun. ACM* 64.10 (2021), 64–71. ISSN: 0001-0782. DOI: 10.1145/3448248. URL: https://doi.org/10.1145/3448248.

[105] Håkan LS Younes and Michael L Littman. "PPDDL1. 0: An extension to PDDL for expressing planning domains with probabilistic effects." In: *Techn. Rep. CMU-CS-04-162* 2 (2004), p. 99.

[106] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. "A Survey of Large Language Models." In: *arXiv preprint arXiv:2303.18223* (2023).