IMPERIAL COLLEGE LONDON

DEPARTMENT OF COMPUTING

Action Grammars: A Grammar Induction-Based Method for Learning Temporally-Extended Actions

Author:
Robert Tjarko Lange

Supervisor:
Dr Aldo Faisal

Submitted in partial fulfillment of the requirements for the MSc degree in Computing with Specialism in Machine Learning of Imperial College London

September 2018

Abstract

Hierarchical Reinforcement Learning algorithms have been successfully applied to large-scale problems with sparse reward signals. By operating at multiple time scales, the Reinforcement Learning agent is able to overcome difficulties in exploration and value information propagation. However, all of these algorithms face one of three unsatisfying properties: They either require manual specification of hierarchical structures, lack clear interpretability or can hardly be justified in a comparative fashion.
This research project combats all of these shortcomings in a fully automated and end-to-end fashion. By treating an on-policy trajectory as a sentence sampled from the policy-conditioned language of the environment, we are able to apply powerful ideas from computational linguistics to the substructure discovery problem. We identify hierarchical constituents with the help of grammatical inference and the theory of language parsing. Afterwards, the agent is endowed with this information in order to accelerate and structure her learning process.
Our contributions are the following: First, we develop an algorithmic framework which constructs macro-actions and options from stochastic and deterministic context-free grammars. Second, one can easily interpret the resulting parse trees as subgoal achievements which convey semantic meaning. Thereby, we are able to partially answer questions of interpretability in Reinforcement Learning. Finally, a generalization of syntactic surprisal, replay surprisal, can be utilized to perform grammar comparison and to choose the grammar that does not overfit noise in sampled sentences. Thereby, we allow for efficient exploration. We demonstrate the efficiency and strong performance of our framework in the context of multiple environments which suffer severe long-term credit assignment problems.

We call the resulting framework *action grammars* following Pastra and Aloimonos (2012).[1]

[1]This project builds on an independent study option (ISO, Lange (2018a,b)) conducted under the supervision of Dr Faisal. During the ISO we discovered major shortcomings of the current state of Hierarchical Reinforcement Learning. All theoretical contributions, replications and experiments displayed in this thesis were conducted independently of the ISO. In order to maintain clarity and completeness there remains overlap in the background provided in chapter 2 and 3 of this report. Still, these sections are tailored towards our application and stand conditionally independent of the ISO. E.g. we have extended the literature review to include recent developments from the 35th International Conference on Machine Learning 2018, added completely new sections on macro-actions as well as syntactic surprisal and probabilistic context-free grammars. There exist clear differences and substantial contributions which remain distinct to this project (for more details see Appendix B). We explicitly state and cite when parallels remain.

Acknowledgements

I am grateful to my supervisor, Aldo Faisal for making this year as productive and exciting. His enthusiasm, confidence and motivation made me explore and exploit the unknown unknowns.

Thank you!

Contents

List of Figures

List of Algorithms

List of Tables

Symbol directory

Symbols and abbreviations refer to the following, except where stated differently.

Abbreviations

EM	Expectation-Maximization	MDP	Markov Decision Process
RL	Reinforcement Learning	SMDP	Semi-Markov Decision Process
DRL	Deep Reinforcement Learning	HAMs	Hierarchies of Abstract Machines
HRL	Hierarchical Reinforcement Learning	TD	Temporal Difference
CFG	Context-Free Grammar	PCFG	Probabilistic CFG
CFAG	Context-Free Action Grammar	PAG	Probabilistic Action Grammar
IGGI	Information Greedy Grammar Inference	DAG	Directed Acyclic Graph
HMM	Hidden Markov Model	PR	Production Rules
RNN	Recurrent Neural Network	MLP	Multilayer Perceptron
LSTM	Long Short-Term Memory	GRU	Gated Recurrent United
AIC	Akaike Information Criterion	BIC	Bayesian Information Criterion

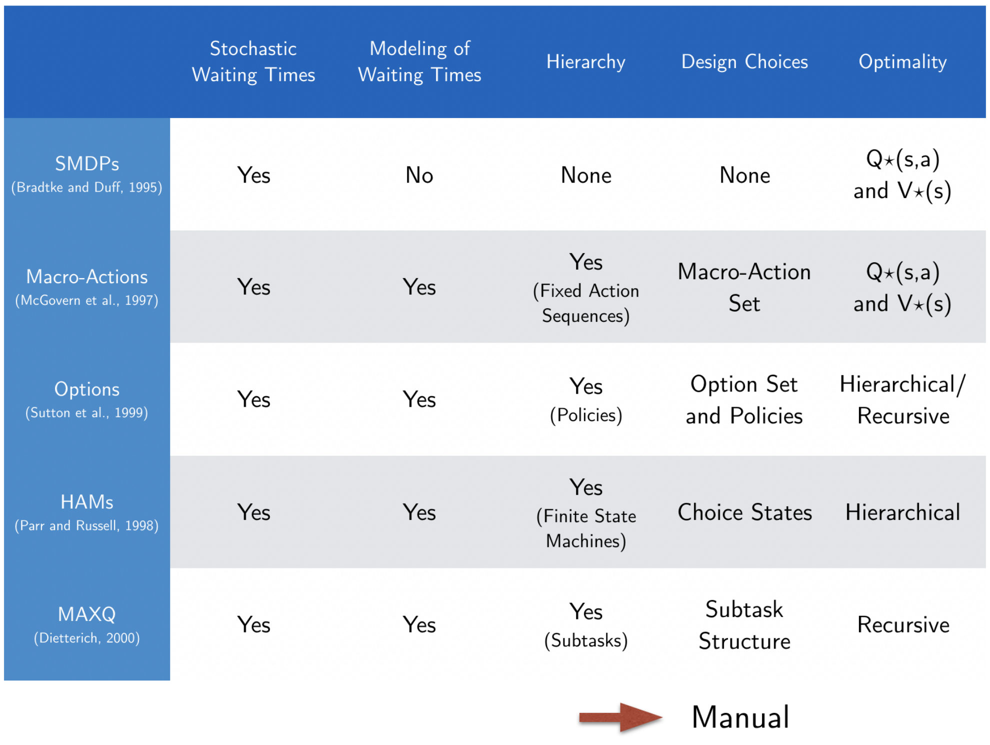
Essential Symbols

$s \in S$	State s in state space S	ϑ	Trajectories
$\pi(s,a)$	Policy prob. assigned to action a in s	$\mathcal{A}^{\hat{G}}$	Grammaraaugmented action space
$r(s,a)$	Exp. immediate reward for taking a in s	N_a	Action Learning episodes
$V^{\pi}(s)$	Value assigned to state s	N_g	Grammar Learning episodes
$Q^{\pi}(s,a)$	Value assigned to state-action (s,a)	N_{init}	Initial Learning episodes
γ	Discount factor	$\hat{G}(\vartheta;\theta)$	Estimated grammar
α_k	Learning rate at iteration k	$\mathcal{M}^{\hat{G}}$	Grammar-estimated macros
$\tau \in \mathbb{Z}^+$	Waiting time between decisions in SMDP	$\mathcal{O}_{\hat{G}(a\|_1^{N_g})}$	Grammar-estimated options
$\mathcal{E}(a,s,\tau)$	Event of taking a in s with waiting time τ	$R^a_{ss'}$	Expected Transition Reward
$U(s,\omega)$	Option value function upon arrival	$P^a_{ss'}$	Transition Probability
ω	Semi-Markov/Markov Option	$R^a_{ss'}(\tau)$	Expected Transition Reward after τ
$\mathcal{I}_\omega$	Option initiation set	$P^a_{ss'\tau}$	Exp. disc. acc. transition prob.
β_ω	Option termination condition	$e(s,m)$	Eligibility value
$h_{tt+\tau}$	Option history from t to τ	$a \in \mathcal{A}$	Action a in action space $\mathcal{A}$
Ω	Set of all possible histories		
$\mathcal{O}$	Set of all available options in S		
$\mu(s,\omega)$	Policy prob. assigned to ω in s		
$\mathcal{E}(a,s,\tau)$	Taking a in s with τ waiting time		
$m \in \mathcal{M}$	Macro-action		
λ	Eligibility parameter		
δ	TD error		
$\pi_{\mathcal{O}}(\omega\|s)$	Policy over options		
$\tilde{\pi}_{\mathcal{O}}(\omega_t\|\omega_{t-1},s_t)$	Option-to-option policy		
Σ	Terminal vocabulary		
$\mathbb{N}$	Non-terminal vocabulary		
$V^\star$	Set of all possible string		
V^+	$V^\star$ without null-string		
$\mathcal{P}$	Production rules		
p	Probability vector associated with PCFG		
θ	Hyperparameters		
z	Hiddens State HMM		

Chapter 1

Introduction

The Reinforcement Learning (RL, e.g. Bertsekas and Tsitsiklis (1995); Sutton and Barto (1998)) agent faces a structural credit assignment problem in order to maximize her expected cumulated and discounted reward. As the dimensionality of both the action and state space grows linearly, the computational complexity of classical learning algorithms grows at an exponential rate. This phenomenon is known as the curse of dimensionality in RL. Furthermore, sparse rewards can hinder reward-based learning and exploration of the state space. In order to overcome these challenges of complex real-world domains, it has been proposed to make use of hierarchical substructures, intrinsic motivation as well as the reusability of low-level skills. Classical Hierarchical Reinforcement Learning (HRL, see Barto and Mahadevan (2003) and figure 1.1 for an overview) algorithms are based on learning sequential solutions at different temporal levels of decision making. Instead of choosing a strategy over primitive actions, the HRL agent optimizes a policy over high-level structures. These structures in turn execute substructures lower in the hierarchy. Finally, this leads to the execution of potentially multiple primitive actions at the bottom of the control hierarchy. Learning a policy over temporally-extended actions thereby allows the agent to combat the uncertainty induced by single time-step decision making. The agent overcomes exploration problems, by restricting her decision process in a syntactically meaningful way. Hierarchies can be defined over fixed sequences of primitive actions (macro-actions; see (McGovern et al., 1997)), sub-policies (options; see (Sutton et al., 1999)), finite state machines (HAMs; see (Parr and Russell, 1998)) or sub-tasks (MAXQ; see (Dietterich, 2000)). The biggest challenge of HRL is the actual identification of a meaningful substructure specification. As of yet, this challenge has not been successfully addressed. Graph theoretic, diverse density and visitation-based approaches try to identify bottlenecks which represent meaningful goal states (e.g. McGovern and Barto (2001); Hengst (2002); Menache et al. (2002)). Parametric gradient theorems (Bacon et al., 2017) can help to construct meaningful sub-policies and required termination conditions. Deep HRL lets the agent simultaneously generalize the state space information using deep architectures to identify subgoals while planing the achievement of such (e.g. Kulkarni et al. (2016a,b); Vezhnevets et al. (2016, 2017)).



	Stochastic Waiting Times	Modeling of Waiting Times	Hierarchy	Design Choices	Optimality
SMDPs (Bradtke and Duff, 1995)	Yes	No	None	None	$Q_\star(s,a)$ and $V_\star(s)$
Macro-Actions (McGovern et al., 1997)	Yes	Yes	Yes (Fixed Action Sequences)	Macro-Action Set	$Q_\star(s,a)$ and $V_\star(s)$
Options (Sutton et al., 1999)	Yes	Yes	Yes (Policies)	Option Set and Policies	Hierarchical/ Recursive
HAMs (Parr and Russell, 1998)	Yes	Yes	Yes (Finite State Machines)	Choice States	Hierarchical
MAXQ (Dietterich, 2000)	Yes	Yes	Yes (Subtasks)	Subtask Structure	Recursive

Figure 1.1: Overview of Classical HRL Algorithms. Altered from Lange (2018a).

Still, all of the above require the choice of many hyperparameters (e.g. the number of desired options, network architecture) and make fairly strong assumptions regarding the initiation set of the subtask (e.g. the complete state space). This easily leads to misspecification and can result in a significant slow-down of learning and exploration. While providing a fully end-to-end approach, "deep" attempts lack interpretability and often times require severe amounts of pre-training.
Human infants, on the other hand, learn seemingly unstructured patterns in nature and from observing role models. They are incredibly well equipped to infer hierarchical rule-based structures from language, visual input as as well as auditory stimuli (Marcus et al., 1999; Frank et al., 2009; Marcus et al., 2007). By observing an expert, they get a head-start in their learning process and are able to learn over higher level sequences of low level control elements. Furthermore, there is convincing evidence from several MEG and fMRI studies that indicates a form of hierarchical language comprehension in the brain (Ding et al., 2017; Frank and Christiansen, 2018; Brennan et al., 2016; Nelson et al., 2017) and a parallelism to motor control (Pastra and Aloimonos, 2012; Stout et al., 2018). Inspired by such observations this research project overcomes the identified weaknesses by merging HRL with the field of computational linguistics. More specifically, we propose the usage of grammatical inference algorithms to extract hierarchical structures from trajectory sentences with the ultimate aim to deploy them in the Hierarchical Reinforcement Learning process. Thereby, the original RL problem is split into two alternating stages:

1. **Grammar Learning**: Given some sequential experience (either state transitions or action traces) we treat the time-series as a sentence sampled from the language of the policy-conditioned environment. The language in turn was generated by the grammar

induced by the current policy. Using grammar induction the agent extracts hierarchical constituents of the current policy. Based on this estimate she constructs temporally-extended actions which convey hierarchical syntactic meaning. Afterwards, we augment the agent's action space with such actions.

2. **Action Learning**: Using the grammar-augmented action space, the agent acquires new value information from interacting with the environment and refines his action-value estimates using Semi-Markov-Decision-Process-Q-Learning (Bradtke and Duff, 1995). Afterwards, we sample simulated "sentences" from the improved policy by rolling out observations in the environment and repeat step 1.

The overall learning procedure consists of alternating updates of the grammar estimate and a refinement of the corresponding "action grammar"-value estimates. We implement two conceptual HRL pipelines (see figure 1.2):

1. **Context-Free Action Grammar (CFAG)**: Given sequences of primitive action execution we infer a context-free grammar (CFG) using classical algorithms (e.g. Sequitur (Nevill-Manning and Witten, 1997), G-Lexis (Siyari et al., 2016) or IGGI (Schoenhense et al., 2017)). The resulting production rules can then be flattened into sequences of primitive actions. We use such to either construct deterministic macro-actions or options based on state-specific termination. The resulting temporally-extended action have a fixed termination determined either by a termination state or the length of the macro.

2. **Probabilistic Action Grammar (PAG)**: Given sequences of state transitions we fit probabilistic grammars using Hidden Markov Models (HMMs) and Recurrent Neural Networks (RNNs). The action space of the agent is then augmented by a complex new action, an *action grammar option*. The agent learns the value of choosing to follow actions sampled from the PCFAG conditioned on the previous sequence of actions and the current state. We stop the execution based on a syntactic information-theoretical complexity measure (Hale, 2001; Levy, 2008). Thereby, our approach is deeply rooted in the Bayesian doctrine of reasoning about uncertainty. As the option becomes more uncertain about its state transition it returns control back to the agent.

Still, this leaves us with the task of choosing the *best* grammar. Similar to model comparison problems in supervised learning, we solve this problem by estimating a generalization error. We learn a grammar on the train set and compute the value of a loss function for both train and test traces. This loss function is defined as the cumulative surprisal (Hale, 2001; Levy, 2008) or self-information from observing a particular action as the successor in a sequence. While parsing through an expert trace we expect the next state based on our previously extracted hierarchy. We seek to minimize surprising encounters and prefer grammars which have identified the hierarchical nature of the policy. We call the cross-validated estimate of this loss *replay surprisal*. Finally, we display powerful results on several environments with hierarchical structure and severe long-term credit assignment problems (e.g. Towers of Hanoi, the Four Room Problem, OpenAI environments). The main contributions of this project are the following:

- We introduce a computational framework that synthesizes Computational Linguistics and HRL (see section 4.1).
- We propose novel end-to-end macro-action and option discovery (see sections 4.2 and 4.3) algorithms for deterministic/stochastic environments which use grammar induction.

- We propose a surprisal-based metric to compare grammars in the context of HRL (see section 4.4).

Furthermore, minor contributions include the following:

- We implement and formalize an eligibility trace (Watkins and Dayan, 1992) algorithm for the SMDP-Q-Learning (Bradtke and Duff, 1995) framework (see section 2.3.1).
- The interpretability of the substructures is enhanced due to the properties of CFGs and the resulting parse trees.

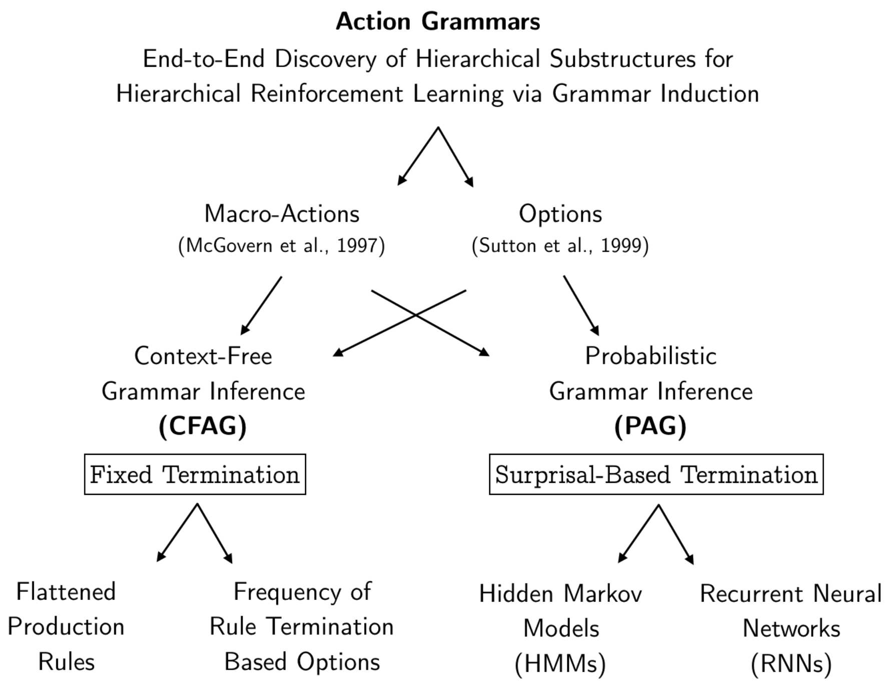

Action Grammars
End-to-End Discovery of Hierarchical Substructures for Hierarchical Reinforcement Learning via Grammar Induction
Macro-Actions (McGovern et al., 1997)
Options (Sutton et al., 1999)
Context-Free Grammar Inference (CFAG)
Probabilistic Grammar Inference (PAG)
Fixed Termination
Surprisal-Based Termination
Flattened Production Rules
Frequency of Rule Termination Based Options
Hidden Markov Models (HMMs)
Recurrent Neural Networks (RNNs)

Figure 1.2: Action Grammars: A Conceptual Overview

We proceed in the following way: Chapter 2 reviews the required Reinforcement Learning background. We start by outlining the temporal generalization of Markov Decision Processes (MDP), namely Semi-Markov Decision Processes (SMDP). Afterwards, we review two specific approaches to hierarchical task solving: Macro-actions (McGovern et al., 1997; McGovern and Sutton, 1998) and options (Sutton et al., 1999). Finally, we discuss current approaches from the literature to infer meaningful substructures in hierarchically structured environments. Chapter 3 introduces notation and intuition for grammatical inference of both CFGs and PCFGs. Furthermore, we review the notion of syntactic surprisal. Chapter 4 provides the key theoretic contributions of this project: We introduce an algorithmic framework which automatically and fully end-to-end identifies hierarchical substructures using grammar induction. Furthermore, we introduce the replay surprisal approach to solve the grammar comparison problem. Chapter 5 implements and proves the effectiveness in multiple "hard" RL environments. Chapter 6 summarizes our main results and discusses future endeavors. Finally, the appendix provides experimental details, summarizes distinct contributions and discusses legal and ethical considerations of this project.

Chapter 2

Time Uncertainty

The following chapter introduces the required background knowledge in Hierarchical Reinforcement Learning. Markov Decision Processes define a simple RL framework in which agents execute actions which lead to a transition in the following period. In discrete-time MDPs the time in which the agent waits for a transition is fixed to one time period. Semi-Markov Decision Processes extend MDPs to actions with variable waiting times. Macro-actions and options are both formalized within the SMDP framework and provide explicit descriptions of the waiting time distributions between decisions. Thereby, they impose explicit temporal hierarchies on the behavior of the agent. Identifying suitable hierarchical structures is not easy. We review the current literature.

2.1 Markov Decision Processes

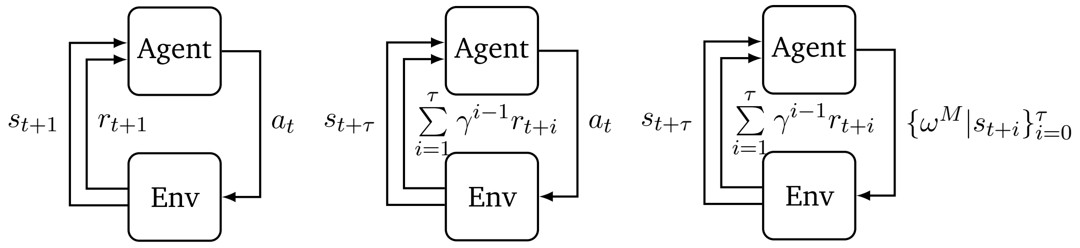
The classical Reinforcement Learning problem (see e.g. (Sutton and Barto, 1998)) is based upon the iterative optimization of a behavioral sequential strategy in Markov Decision Processes. MDPs provide an elegant account of a simple agent-environment closed loop (see figure 2.1a). A discrete MDP is defined by the tuple $(S, \mathcal{A}_s, P_{ss'}^a, R_s^a, \gamma)$ where S denotes a finite non-empty set of environment states, $\mathcal{A}_s$ denotes a finite non-empty set of actions which are admissible given the agent's state $s \in S$. Let $R_s^a = \mathbb{E}[r_{t+1}|s, a]$ denote the expected reward of transitioning from state s in period t to a new state in period $t+1$ given the execution of action a. The probability of transitioning to a specific s' is given by $P_{ss'}^a = P(s_{t+1} = s'|s_t = s, a_t = a) \in [0, 1]$ and models the uncertainty innate to the environment. It is assumed that waiting times between transitions are constant at one period. When the agent decides to execute a primitive action she is assured to experience some from of transition in the next period - no matter what. Hence, there is no notion of temporal abstraction and the waiting time distribution is a Delta-Dirac distribution with center of mass at 1. We know the frequency of control with certainty. Given the model of the MDP described by $P_{ss'}^a$ and R_s^a the problem can easily be solved using the following Bellman Equation (see Barto and Mahadevan (2003, p. 346) and Lange (2018b, p. 2f.)):

$$Q^\star(s, a) = \max_\pi Q^\pi(s, a) = R_s^a + \gamma \sum_{s' \in S} P(s'|s, a) \max_{a' \in A_{s'}} Q^\star(s', a')$$

As the dimensionality of state and action space grows, full-width Bellman backups become impractical. Instead the MDP is solved using either value-based or policy gradient learning methods. In this project we focus on model-free value-based learning as compared to model-based learning. A simple model-free and off-policy Q-learning RL agent updates his action value estimates according to the bootstrapped value estimate of the next action value according to the behavioral policy (see Watkins and Dayan (1992, p. 281) and Lange (2018b, p. 3)):

$$Q(s,a)_{k+1} = (1-\alpha)Q(s,a)_k + \alpha\left(r_{t+1} + \gamma \max_{a' \in \mathcal{A}} Q(s',a')_k\right).$$

The agent iteratively adapts her value estimate based on the temporal difference (TD) error. The optimal policy, $\pi^\star : S \times \cup_s \mathcal{A}_s \to [0,1]$ of the agent is then obtained by maximizing the value of the converged Q-function with respect to the action space: $\pi^\star = arg\max_{a'} Q^\star(s,a')$.


Agent
Env
s_{t+1}
r_{t+1}
a_t
Agent
Env
$s_{t+\tau}$
$\sum_{i=1}^{\tau} \gamma^{i-1} r_{t+i}$
a_t
Agent
Env
$s_{t+\tau}$
$\sum_{i=1}^{\tau} \gamma^{i-1} r_{t+i}$
$\{\omega^M | s_{t+i}\}_{i=0}^{\tau}$

(a) MDPs (b) SMDPs (c) Markov Options

Figure 2.1: Closed Loops: MDPs, SMDPs and Options. Altered from Lange (2018b, p. 8)

2.2 Semi-Markov Decision Processes

Semi-Markov Decision Processes (see figure 2.1b; Bradtke and Duff (1995)), on the other hand, allow us to not only model epistemic uncertainty but also time uncertainty. They are defined as the tuple $(S, A_s, P^a_{ss'\tau}, R^a_s, \gamma)$. The time between actions is modeled as a positive-valued discrete random variable, $\tau \in \mathbb{Z}^+$.[1] The transition probability distribution now accounts for the joint event of transitioning into the new state s' after τ time steps, $P^a_{ss'\tau} = P(s',\tau|s,a)$ (Sutton et al., 1999, p. 189). Marginalizing over the waiting time yields the distribution which measures the collected and discounted likelihood of *eventually* transitioning to state s', $\sum_{\tau=1}^{\infty} \gamma^\tau P(s',\tau|s,a)$. Let $\mathcal{E}(a,s,\tau)$ denote the event of taking action a in state s and afterwards having to wait for τ periods. $R^a_s(\tau) = \mathbb{E}[\sum_{i=1}^{\tau} \gamma^{i-1} r_{t+i} | \mathcal{E}(a,s,\tau)]$ then denotes the expected discounted accumulated reward sequence obtained from staying in state s for τ time points, having chosen action a. Only after the waiting period has finished, the agent transitions into the next state s'. Compared to traditional MDPs this allows us to model complex waiting time distributions. Again, the Bellman optimality equation is given by (see Bradtke and Duff (1995, p. 395f.) and also see ISO report (Lange, 2018b, p. 4)):

$$Q^\star(s,a) = \max_\pi Q^\pi(s,a) = R^a_s(\tau) + \sum_{s' \in S} \sum_{\tau=1}^{\infty} \gamma^\tau P(s',\tau|s,a) \max_{a' \in \mathcal{A}_{s'}} Q^\star(s',a')$$

The standard model-free Q-learning algorithm easily translates to the SMDP setting. The updating rule (see Bradtke and Duff (1995, p. 396) and also see ISO report (Lange, 2018b, p. 5)) at iteration $k+1$ is given by

[1]Real-valued waiting time and real-valued action space SMDP theory exists. For simplicity and applicability to HRL we focus on the discrete-time discrete-event SMDP. For more detail we refer to Bradtke and Duff (1995) and the ISO report (Lange, 2018b).

$$Q_{k+1}(s,a) = (1-\alpha_k)Q_k(s,a) + \alpha_k \left(\sum_{i=1}^{\tau} \gamma^{i-1} r_{t+i} + \gamma^{\tau} \max_{a' \in \mathcal{A}_{s'}} Q_k(s',a') \right).$$

Since the agent has access to the unrestricted set of primitive actions in A_s, the SMDP Q-learning algorithm converges to the flat optimal solution (for a proof see Parr (1998)). By abstracting away the need for decision making at every time-step, SMDPs provide an entry point for temporally-extended actions and decision making at different time scales.

2.3 Temporal Abstractions

As the sparsity of the reward signals provided by the environment increases, the agent receives fewer information about which parts of the state space she should care about. Therefore, we face both a credit assignment problem as well as an exploration problem. Temporal difference-based learning algorithms which use one-step backups have a hard time propagating value information back to critical decision points. Hierarchical Reinforcement Learning introduces a hierarchy of time scales at which decisions are made. Thereby, one is able to propagate learning signals further back in time and to regularize the learning problem into meaningful sub-problems. In this section we review two algorithms which can both be embedded into the SMDP framework: Macro-actions (McGovern et al., 1997; McGovern and Sutton, 1998) and Options (Sutton et al., 1999).

2.3.1 Macro-Actions

Macro-actions (McGovern et al., 1997; McGovern and Sutton, 1998) were first introduced in robotics and help to circumvent the curse of dimensionality in large state spaces. Instead of only working with a restricted action space consisting of primitive actions, we allow for the sequential execution of multiple primitive actions as one entity. The agent can now operate at multiple time-scales. While a primitive action leads to a waiting time of one period, macro-actions are executed over multiple time periods ("a_1" vs. "$a_2a_1a_3$"). Only after the execution of the macro has finished (or the goal state has been reached), the agent obtains back control and updates her value estimate. Metaphorically speaking, the agent puts herself into handcuffs in order to avoid additional behavioral uncertainty. Macro-actions define a Semi-Markov Decision Process for which convergence guarantees have been proven by Parr (1998, p. 39). They introduce a deterministic waiting time distribution which is a Dirac-Delta with center of mass at the length of the specific macro-action (e.g. "$a_2a_1a_3$" almost surely terminates after three time steps).
Given a macro-action, $m \in \mathcal{M}$, the agent updates the value of such according to the following SMDP-Q-Learning rule (Bradtke and Duff, 1995, p. 396):

$$Q(s,m)_{k+1} = (1-\alpha)Q(s,m)_k + \alpha \left(\sum_{i=1}^{\tau_m} \gamma^{i-1} r_{t+i} + \gamma^{\tau_m} \max_{a' \in \mathcal{A}} Q(s',a')_k \right)$$

The intuition is straight forward: While simple Q-Learning propagates value information only from one step at a time, SMDP-Q-Learning allows the agent to propagate information over τ_m time steps (McGovern et al., 1997, p. 2f.), where τ_m denotes the respective execution time of macro m. This is one of the two properties which allows the agent to be more sample efficient.

There is a clear relationship to a concept that connects Monte Carlo Methods and TD learning, namely eligibility traces: Both, macros and eligibility traces provide a mechanism for accelerating value propagation. The concept of eligibility provides a form of exponentially decaying memory which attributes credit to actions which lie in the past. As long as the agent stays on-policy, credit is assigned proportionately to the exponential of a $\lambda \in [0, 1]$ factor. Setting $\lambda = 0$ recovers TD learning while $\lambda = 1$ yields Monte Carlo updates (Sutton et al., 1999). Macro-actions, on the other hand, group together pre-defined primitive actions. While executing a macro, there is no possibility for exploration and thereby, they can be interpreted as a form of hard eligibility. More specifically, McGovern and Sutton (1998, p. 10) have shown that SMDP-Q-Learning with macros is more efficient at finding the right policy, whereas $Q(\lambda)$ (Watkins and Dayan, 1992) cares more about absolute action values and not only about the relative action values needed to obtain a good policy. Hence, macro-actions are faster at finding the optimal policy.

Algorithm 1 SMDP-$Q(\lambda)$-Learning. Adapted from Sutton and Barto (1998)

```
Input: Initial $Q(s,m)$, Learning rate $\alpha$. Value propagation parameter $\lambda$.
Output: Optimal state-macro value table.
1: repeat
2:     Initialize $e(s,m) = 0\ \forall s \in \mathcal{S}, m \in \mathcal{M}$ and $s, m$
3:     for Each step of the episode do
4:         Take macro-action $m$, observe $\sum_{i=1}^{\tau_m} \gamma^{i-1} r_i$ and $s'$
5:         $m^\star \leftarrow arg\max_b Q(s', b)$
6:         $\delta \leftarrow \sum_{i=1}^{\tau_m} \gamma^{i-1} r_i + \gamma^{\tau_m} Q(s', m^\star)$
7:         $e(s,m) \leftarrow e(s,m) + 1$
8:         for all $s, m$ do
9:             $Q(s,m) \leftarrow Q(s,m) + \alpha\delta e(s,m)$
10:            if $m' = m^\star$ (exploitation) then $e(s,m) \leftarrow \gamma\lambda e(s,m)$ else $e(s,m) \leftarrow 0$
11:        end for
12:    end for
13:    Set $s \leftarrow s'$ and $m = m'$
14: until Convergence
```

Furthermore, the agent alters her exploration behavior when using macro-actions. It can trivially be shown that she is biased towards towards spending more time in parts of the state space where the macros lead her (McGovern et al., 1997, p. 2). The effect on learning of macro-actions is only negative when the macro-actions alone cannot lead the agent to the goal position (McGovern et al., 1997, p. 5). Following an initial idea dating back to McGovern and Sutton (1998, p. 10), we propose a simple adaptation of the original eligibility trace algorithm to temporally-extended actions. Algorithm 1 extends the classical $Q(\lambda)$ algorithm by altering the equations of the temporal difference (TD) error to the temporally-extended version. By treating a macro-action as a single entity, we are able to combine both the value propagation properties of the eligibility traces as well as the changes in exploration induced by macro-actions.

We note that this formalism might also provide an automated mechanism for extending macro-actions. If an action is assigned with eligibility, irregardless of the state ($e(., a)$ large) and previous to executing a macro, this might indicate that the macro should be extended to include the previous action, $a||m$.

2.3.2 Options

Options (see figure 2.1c, Sutton et al. (1999)) are arguably the most intuitive HRL framework. They capture the notion of locally-constrained sub-policies and allow the agent to learn a policy over an option set instead of the primitive actions. This enhances the action set of the agent while at the same time restricting the number of decisions, she has to make due to the stochastic waiting time. The overall combinatorial complexity of decision points reduces significantly. Ultimately, this leads to improvements in computation and faster learning. A Markov option (Sutton et al., 1999, p. 186f.) is defined as the tuple $\omega^M =< \mathcal{I}_\omega, \pi_\omega, \beta_\omega >$, where $\mathcal{I}_\omega \subseteq S$ denotes the initiation set of states in which the option is allowed to start execution, $\pi_\omega : S \times A \rightarrow [0,1]$ represents the intra-option policy defined over the Cartesian product of the state-action space and $\beta_\omega : S^+ \rightarrow [0,1]$ is the termination condition which stochastically terminates the execution of the option given the current state of the agent. In practice, often times the termination condition is constructed in a deterministic fashion such that it takes value 1 for a specific state and 0 otherwise. Hence, also an option and its termination condition define a stochastic waiting time distribution within the SMDP framework. It differs from classical macro-actions (Hauskrecht et al., 1998) since the sequence of action executions within the option is not deterministic but state dependent. Thereby, the option execution adapts to the environment. If an action execution fails, the agent is able to adapt his sequential behavior according to the state dependent intra-option policy. For a macro-action this is not the case. Instead, the agent has to follow the previously selected sequence of actions.

The option is called Markov since the termination condition and intra-option policy solely depends upon the current state of the agent. A semi-Markov option (Barto and Mahadevan, 2003, p. 352), ω^{SM}, generalizes this by conditioning the intra-option policy termination on the complete history of states, actions and rewards obtained since the beginning of option execution (Barto and Mahadevan, 2003, p. 352). The history for an option which starts its execution at time t and finishes after τ periods is denoted by $h_{tt+\tau} = \{s_t, a_t, r_{t+1}, s_{t+1}, \dots, r_{t+\tau}, s_{t+\tau}\}$. Furthermore, let Ω denote the set of all possible histories. A semi-Markov option's policy is defined as $\pi_\omega : \Omega \times A \rightarrow [0,1]$ and the condition for termination is $\beta_\omega : \Omega \rightarrow [0,1]$. Sutton et al. (1999, p. 186) state that semi-Markov options easily can result from defining options over options, introducing an additional hierarchical layer. This is due to the observation that the choice of an option is different before and after the previous one has been executed (Sutton et al., 1999, p. 187). Hence, all convergence guarantees easily carry over. Let $\mathcal{O} = \cup\omega$ denote the option set. The top-level of the hierarchy is then given by the resulting policy over options, $\pi_{\mathcal{O}}(\omega|s)$.

$$\begin{tikzcd}
 & & & & Bern(\beta_{\omega_1}) \arrow[r] & \pi_{\mathcal{O}}(\omega|s_2) & \boxed{\text{Return Control to Option-to-Option Policy}} \\
\pi_{\mathcal{O}}(\omega|s_1) \arrow[r] & \omega_1 \arrow[r] & \pi_{\omega_1}(a|s_1) \arrow[r] & (a_1, r_2, s_2) \arrow[ur] \arrow[dr] & & & \\
 & & & & Bern(1-\beta_{\omega_1}) \arrow[r] & \pi_{\omega_1}(a|s_2) & \boxed{\text{Continue Execution Intra-Option Policy}}
\end{tikzcd}$$

Figure 2.2: Markov Options: Schema of an Execution

Finally, we define the option-to-option policy (Smith et al., 2018, p. 3) function as:

$$\tilde{\pi}_{\mathcal{O}}(\omega_t|\omega_{t-1}, s_t) = \underbrace{\left(1 - \beta_{\omega_{t-1}}(s_t)\right) \delta_{\omega_t \omega_{t-1}}}_{\text{Continuation of option}} + \underbrace{\beta_{\omega_{t-1}}(s_t) \pi_{\mathcal{O}}(\omega_t|s_t)}_{\text{Switch to new option}}$$

We highlight that the option set does not substitute but complement the set of primitive actions. The original low-level actions can be viewed as one-step options with deterministic termination and policy (Sutton et al., 1999, p. 187). Importantly, it can be easily shown that a MDP whose action space consists of options is equivalent to an SMDP (Sutton et al., 1999, p. 189). Again, we can apply SMDP Q-Learning and obtain the following updating rule (Sutton et al. (1999, p. 195) and also see ISO report (Lange, 2018b, p. 9)):

$$Q_{k+1}(s, \omega) = (1 - \alpha_k) Q_k(s, \omega) + \alpha_k \left(\sum_{i=1}^{\tau} \gamma^{i-1} r_{t+i} + \gamma^\tau \max_{\omega' \in \mathcal{O}_{s'}} Q_k(s', \omega') \right).$$

Finally, we have to differentiate between different notions of optimality as introduced by Dietterich (2000) (also see Lange (2018b, p. 5): Flat, hierarchical and recursive optimality. Flat optimality is simply global optimality achieved for example policy by a converged Q-learner. A policy is called hierarchically optimal if it is the best with respect to the prescribed hierarchical structure Barto and Mahadevan (2003, p. 362). A recursively optimal policy, on the other hand, is defined with respect to a set of pre-defined subtasks (Barto and Mahadevan, 2003, p. 362). Given the policies of the children subtasks, the policy is optimal for the current subtask. Usually, hierarchical optimality is regarded as the stronger notion of optimality (Barto and Mahadevan, 2003, p. 362). For options we have the special case that learning converges to a policy that is hierarchically and recursively optimal at the same time. This is due to the intra-option policies being fixed by the HRL algorithms designer. Hence, the hierarchy consists of sub-child policies. Furthermore, if the set of options, $\mathcal{O}$, includes all primitive actions as one-step options, then SMDP-Q-Learning is guaranteed to converge to the flat optimal policy (Sutton et al., 1999, p. 190).

Intra-option Learning and Interruption

One major concern of the options framework is that learning only happens after the option has finished execution. This can take a while. Sutton et al. (1999, p. 203f.), therefore, proposed intra-option learning. After every primitive action selection a_t in s_t, one determines all options ω that would choose the same action, $\pi_\omega(s_t) = a_t$. The agent can then generalize her experience and update the value of all such options in the following way (Sutton et al. (1999, p. 203ff.) and see ISO report (Lange, 2018b, p. 20)):

$$Q(s_t, \omega)_{k+1} = Q(s_t, \omega)_k + \alpha \left(r_{t+1} + \gamma U(s_{t+1}, \omega)_k - Q(s_t, \omega) \right)$$
$$U(s, \omega)_k = (1 - \beta(s)) Q(s, \omega)_k + \beta(s) \max_{\omega'} Q(s, \omega')_k$$

where $U(s, \omega)$ denotes the option value function upon arrival. Intuitively, it can be interpreted as a termination probability weighted average of continuing the execution of the old option and starting a new one. This approach turns out to be more sample efficient since one is able to use more of the information from the same amount of training observations (Sutton et al., 1999, p. 204). Finally, it also helps to learn in cases where options are not completely executed (e.g. when the goal state is reached before termination). Since the primitive actions are trivially consistent with the option, the agent is still able to learn from the reinforcement signal. Another useful trick that can speed up learning is the interruption of an option execution (Sutton et al., 1999, p. 196f.). Similar to a policy improvement argument, one might improve a policy over options, μ, if the value of interruption is higher than the value of continuation. More specifically, if option ω is Markov in state s_t, we compare

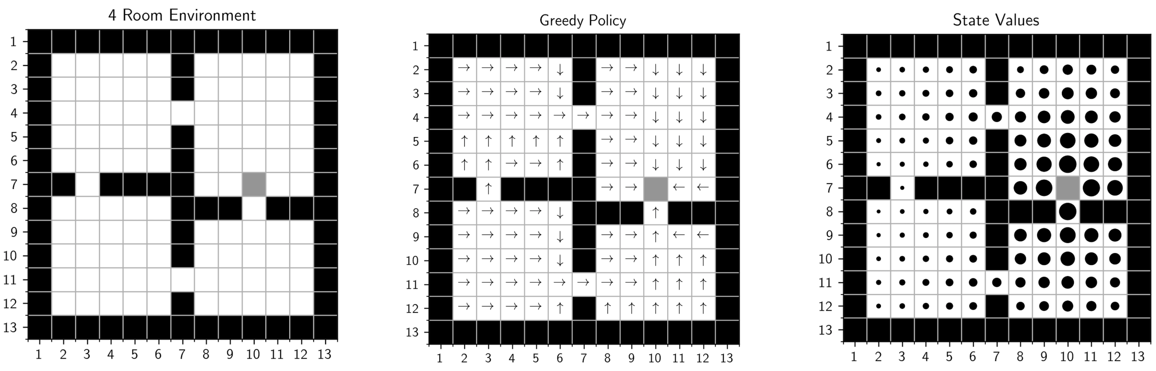


Figure 2.3: Foor Room Problem (Sutton et al., 1999). **Left.** The Game Environment. **Middle.** Q-Learning Solution after 10000 episodes. **Right.** State Values (indicated by circle size) after 10000 Q-Learning episodes.

the value of continuing its execution, $Q^{\mu}(s_t, \omega)$, with starting a new option execution in this state, $V^{\mu}(s) = \sum_q \mu(s,q)Q^{\mu}(s,q)$. If the option advantage function is negative, hence $A(s,\omega) = Q^{\mu}(s_t,\omega) - V^{\mu}(s) < 0$, we set $\beta'(s_t) = 1$ and define a new interrupted policy μ' Sutton et al. (1999, p. 197). Sutton et al. (1999, p. 197f.) prove that one can thereby improve the policy of the agent.

As a running example throughout this report we will make use of the classic four room problem outlined in Sutton et al. (1999, p. 192f.) (see figure 2.3). At the beginning of an episode an agent is randomly placed within the environment and can move in all directions (as long it does not violate the wall constraints). Upon reaching the goal position (indicated in grey) the agent receives a reward of 1. For every other transition she yields a null reward. The agent discounts her reward by γ. State transitions occur with 90 percent success rate. In the remaining 10 percent the successor state is sampled from the neighborhood of the current state.

At this point we can already glimpse at the hierarchical structure innate to this environment. It goes as simple as this: If the state of the agent is not in the room with the goal, the agent has to leave the room as fast as possible. Hence, an optimal policy lets the agent transition through the hallways. A classical manual construction of an option set can be found in figure 2.4. Each individual option leads the agent to one of the four hallways. Instead of having to learn the same overarching principle of "escaping" the room for every state, this option set reduces the problem to determining which hallway to move to. Afterwards, the exact goal location can be found using primitive actions.

Figure 2.5 shows learning results averaged across 5 agents and 10 on-policy rollouts in the four room problem. The middle column implements macro-actions with eligibility traces. We can observe that macros are not suited for noisy environments. Given an unsuccessful transition the agent is locked into executing the remaining actions of the macro, even though this might now be sub-optimal. Softmax (or Boltzmann) exploration and eligibility traces ($\lambda = 0.1$) perform best combined with the hand-crafted macros. Still, the variance in the learning process averaged over 5 learning trials is significant. Learning with the hallway options, on the other hand, is very fast and exhibits almost no variance. Furthermore, one can observe that both intra-option learning as well as interruption both enhance learning significantly.

The hallway options were constructed in a completely manual fashion and encode a significant

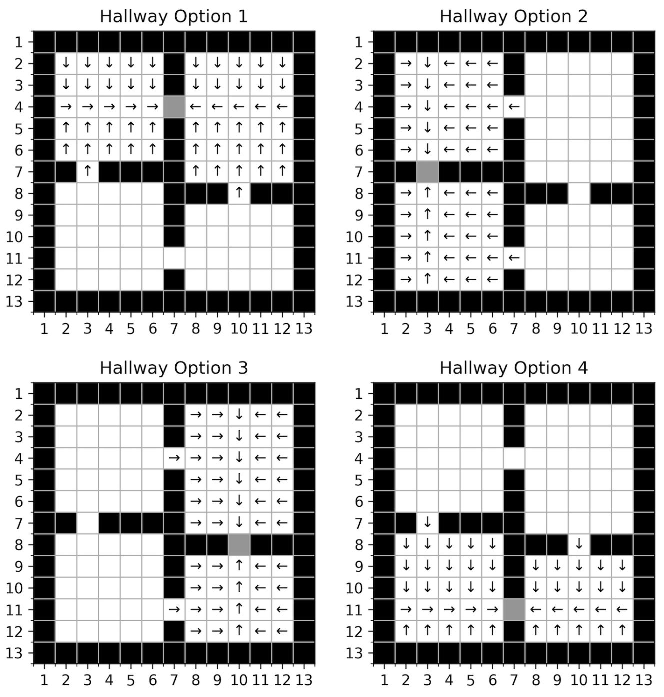


Figure 2.4: Hallway Options (Sutton et al., 1999). The initiation set is indicated by the cells which contain arrows. The intra-option policy is indicated by the direction of the arrows. The termination set is given by the state colored in grey.

amount of domain knowledge. Being able to automatically identify subroutines which exploit the hierarchical nature of the optimal policy allows the agent to learn faster and explore in a more efficient manner. This form of end-to-end learning is the ultimate goal of the project at hand. In the next section we review the current state of literature that deals with subgoal discovery in HRL.

2.4 Discovering Hierarchical Structure

We are not the first to infer hierarchical structure in subgoal achievement problems. More specifically, the option discovery problem deals with the question of how to construct an option set that captures the hierarchical structure between sub-regions of the core MDP. Often times this task is limited to defining the initiation set and the termination condition. The intra-option policy can afterwards be easily learned (e.g. by introducing a pseudo-reward for reaching the termination state).
Figure 2.6 provides an overview of the different approaches to subgoal discovery in HRL. Roughly speaking the approaches can be categorized in three main pillars: First, graph theoretic (Hengst, 2002; Menache et al., 2002; Mannor et al., 2004; Simsek et al., 2004) and visitation-based (McGovern and Barto, 2001; Stolle and Precup, 2002; Simsek et al., 2004) approaches aim to identify bottlenecks. Bottlenecks are regions in the state space which characterize successful trajectories. Gradient-based approaches, on the other hand, discover parametrized options by iteratively optimizing an objective function such as the estimated expected value of the log likelihood with respect to the latent variables in a probabilistic

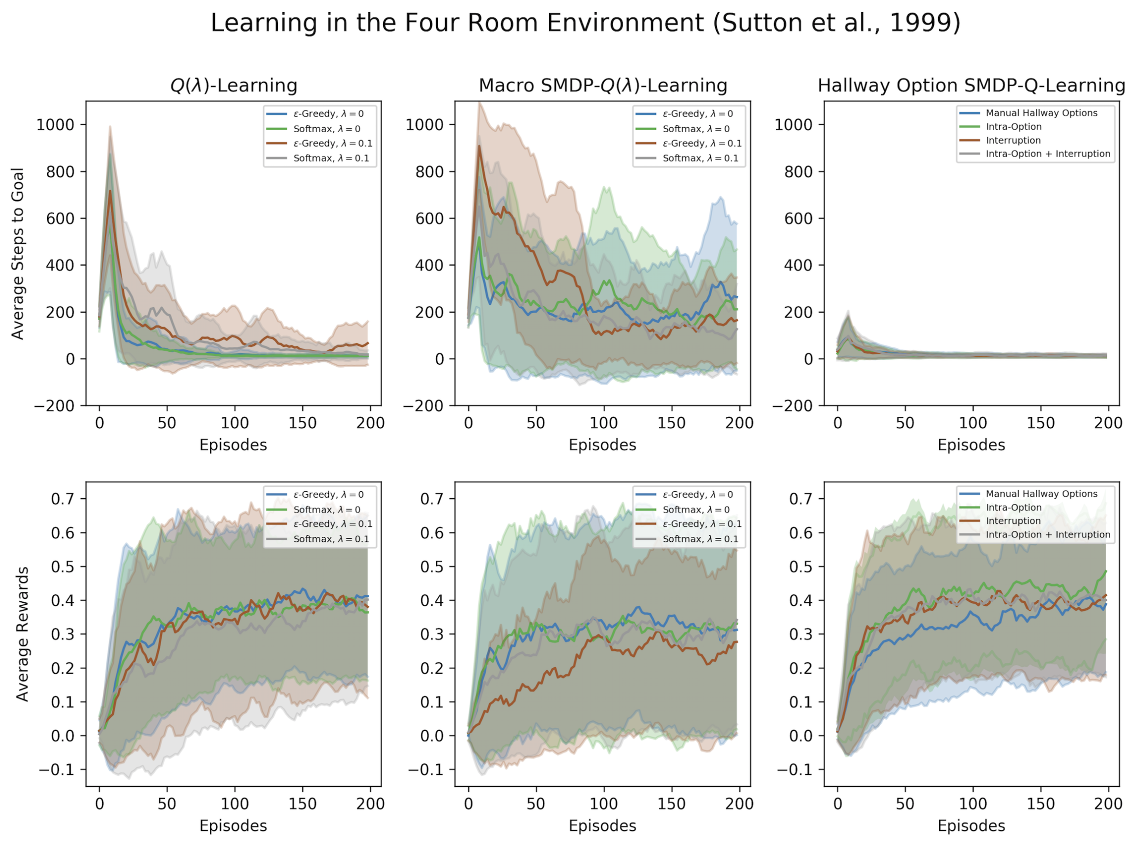


Figure 2.5: Learning with Hallway Options. **Left.** Q(λ)-Learning Agent with Eligibility Traces. **Middle.** Macro-action SMDP-Q(λ)-Learning Agent. The macros used are extracted using the approach described in section 4.2 and are the following $\{ru, drr, rr, drrrr, dd, rrru, dddd, rrr\}$. **Right.** Hallway Options SMDP-Q-Learning Agent. All results were implement by the author.

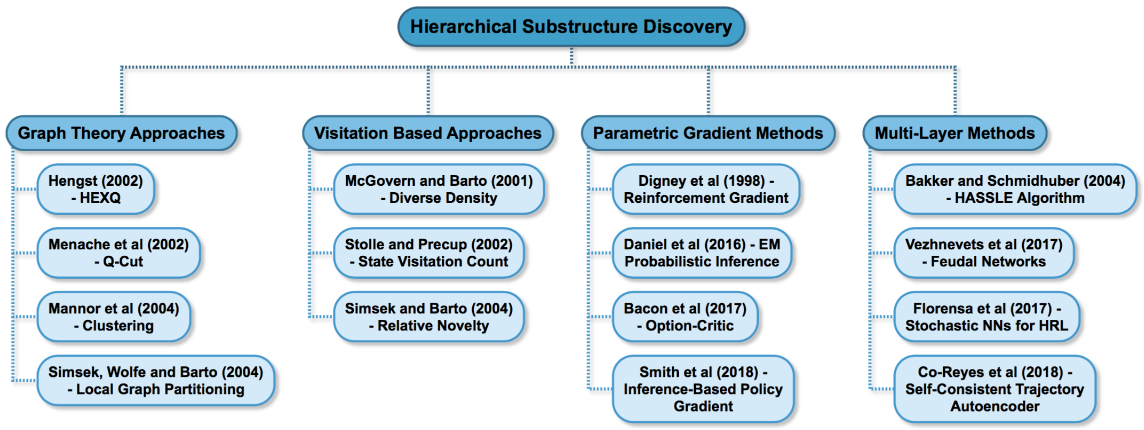

Hierarchical Substructure Discovery
Graph Theory Approaches
Hengst (2002) - HEXQ
Menache et al (2002) - Q-Cut
Mannor et al (2004) - Clustering
Simsek, Wolfe and Barto (2004) - Local Graph Partitioning
Visitation Based Approaches
McGovern and Barto (2001) - Diverse Density
Stolle and Precup (2002) - State Visitation Count
Simsek and Barto (2004) - Relative Novelty
Parametric Gradient Methods
Digney et al (1998) - Reinforcement Gradient
Daniel et al (2016) - EM Probabilistic Inference
Bacon et al (2017) - Option-Critic
Smith et al (2018) - Inference-Based Policy Gradient
Multi-Layer Methods
Bakker and Schmidhuber (2004) - HASSLE Algorithm
Vezhnevets et al (2017) - Feudal Networks
Florensa et al (2017) - Stochastic NNs for HRL
Co-Reyes et al (2018) - Self-Consistent Trajectory Autoencoder

Figure 2.6: Literature Review: Subgoal Discovery in HRL

setting (Daniel et al., 2016) or simply the expected cumulative reward in a policy gradient context (Bacon et al., 2017; Smith et al., 2018). Finally, multi-layer (Bakker and Schmidhuber, 2004; Vezhnevets et al., 2017; Florensa et al., 2017) approaches attempt to split the goal discovery and goal achievement across different stages and layers of the learning architecture. Usually, the top level of the hierarchy specifies goals in the environment while the lower levels have to achieve such.
The following subsections provide more details and highlight some short-comings.

2.4.1 Graph Theory and Visitation-based Approaches

Bottlenecks are defined as parts of the state space which are likely to occur in successful traces and unlikely to be visited in unsuccessful ones (McGovern and Barto, 2001, p. 1). Hence, a bottleneck state has to trade-off the frequency of a state occurrence with its participation in successful experiences. Generally speaking, we are interested in identifying bottlenecks early in the learning process which persist throughout learning (McGovern and Barto, 2001, p. 1). Options provide a large amount of structural information about the learning environment. Therefore, early discovery of significant regions and sub-policies improves the exploration process the most (McGovern and Barto, 2001, p. 201). That being said, one first has to accumulate a reasonable amount of information to actually identify such states. Therefore, most methods have to balance the point in time where the agent is endowed with the option set with time spent exploring the environment to construct a better set.

Graph theoretic approaches (Hengst, 2002; Menache et al., 2002; Mannor et al., 2004; Simsek et al., 2004) do so by viewing the core MDP as a flow problem. States are represented by nodes and transitions occur along the edges of the graph. A bottleneck can then be viewed as a node which accumulates a lot of traffic by the paths of the agent. Hengst (2002, p. 1) construct a set of nested sub-MDP regions from initially random runs and observe frequently visited parts of the core MDP. Thereby the state space is divided it into strongly connected components. Markov regions are then defined as union of strongly connected regions in which any exit state can be reached from any entry (Hengst, 2002, p.2). Afterwards, policies are learned over regions and one is able to obtain a recursively optimal policy. Menache et al. (2002), on the other hand, specify subgoals with the help of MinCut or MaxFlow algorithms. Again, bottlenecks represent loose connections between strongly connected components. The option set is constructed in an online fashion based on cut conditions which ensure the quality of the cut (Menache et al., 2002, p. 5). One is interested in a small number of edges (corresponding to bottlenecks) which connect balanced sets of states. Finally, Mannor et al. (2004) provide a more robust approach by the means of state clustering. Again, they balance the quality of separation and clustering.
Visitation-based and graph theoretic approaches are equivalent, except for the fact that only one of them explicitly constructs a graph. Diverse density, an approach due to McGovern and Barto (2001), identify bottleneck regions by solving a multiple-instance problem in which one tries to infer a target concept with the help of "bags" of instances (states sequences). Each trace can be regarded as either a successful bag of instances or an unsuccessful one. A subgoal necessarily appears in the successful bags while it does not in the unsuccessful (McGovern and Barto, 2001, p. 4). Finally, Stolle and Precup (2002) explicitly construct a complete set of options instead of greedy individual ones. No assumptions about the success of a trace have to be made. Instead, the agent learns the importance of specific states while achieving a set of different tasks. The terminal state is defined to be the most frequently visited one, while the initiation set consists of the set of interpolated states which were experienced more

than an average threshold.

All approaches suffer from severe thirst for observations. In order to differentiate between a regular state and a bottleneck state a lot of experience is necessary. E.g. Stolle and Precup (2002, p. 21) require a large set of experiences to infer accurate state frequencies. Having to split the exploration process and the actual learning also seems very sample inefficient. Furthermore, graph partitioning seems unlikely to be a neurologically plausible way of how humans construct hierarchical sub-structures. Our proposed approach (see chapter 4) solves this distinction by learning temporally-extended actions in an online and sample efficient fashion.

2.4.2 Parametric Gradient Approaches

The approaches presented in the previous section first explore the MDP and afterwards construct options suited for the inferred properties of the environment. This bears the limitation that one has to infer an approximately accurate model of the dynamics before the actual learning with the options can start. An approach to circumvent this is to parametrize the options framework and to optimize such parameters in an iterative fashion.

Daniel et al. (2016) first introduced a probabilistic view to the options framework. The authors view options as latent variables with activation, termination and intra-option policy functions being defined by parametrized probability distributions. A graphical model over the latent options can then describe the transition dynamics in the reinforcement learning environment (Daniel et al., 2016, p. 340). Thereby one is able to formulate a lower bound to the marginal log-likelihood of observed trajectories and to iteratively infer the parameters describing the option distributions with the help of the Expectation-Maximization (EM) algorithm (Baum et al., 1970). The authors derive updating equations and show how this can be incorporated in both imitation as well as reinforcement learning (Daniel et al., 2016, p. 341f.).
The option-critic framework (Bacon et al., 2017), on the other hand, simultaneously solves the option discovery problem and learns policies over such options. Unlike before, the objective now is the expected discounted cumulative reward which can be rewritten in terms of the option-to-option policy function. Afterwards, one is able to derive gradient expressions for both parametric intra-option policies as well as termination condition function (Bacon et al., 2017, p. 1728). This deletes the necessity to pre-specify subgoals and only leaves the designer to choose the number of desired options (Bacon et al., 2017, p. 1726). Based on a termination condition gradient expression that negatively depends on the advantage function the agent is more likely choose a new option at an earlier point in time if the current choice is sub-optimal (Bacon et al., 2017, p. 1728).
Finally, Smith et al. (2018) combine both points of view. Again, options are viewed as latent variables but now in the policy gradient context of Bacon et al. (2017). Instead of optimizing a lower bound with the help of the full EM algorithm, one can instead marginalize over the hidden options (Smith et al., 2018, p. 2). The corresponding posterior for the expectation in the policy gradient can be found via recursion and a simple forward pass in the expectation step. The main advantage lies in being able to simultaneously learn all shared parameters with the help of the inferred policy gradients. Furthermore, the authors are able to show that the degree of option specialization increases which in turn helps to interpret them as distinct functional behaviors (Smith et al., 2018, p. 6).

All of the above approaches suffer from the following two limitations: The initiation set is assumed to be the complete state space. Due to the observation that the efficiency of an option depends on the position in the state space, this can significantly slow down the learning process. The state-specialization of the option has to fully be learned by the policy over options. Second, the number of desired options has to chosen as a hyper-parameter. An interesting alternative to this would be a Bayesian non-parametric approach where the optimal number of options could be obtained by maximizing the marginal likelihood. Interestingly, Smith et al. (2018, p. 7) observe that the number of desired options might vary throughout the learning process. In the beginning, more options might be useful to stabilize sub-optimal behavior. Later on, as the behavior improves, the number of options can be reduced. Both of these short-comings have to be addressed in future work. Our approach is not directly comparable since we do not parametrize options. Instead, the option inference algorithm is parametrized and its hyper-parameters can be optimized.

2.4.3 Multi-Layer Approaches

Most of recent research efforts intend to train a hierarchical architecture in an end-to-end fashion. Combining temporal and spatial abstraction in Deep Reinforcement Learning (DRL, e.g. Mnih et al. (2013, 2015, 2016)) yields impressive results in many complicated settings. The field of Deep HRL is not directly related to the options framework. Instead, the agent simultaneously generalizes the state space information using deep architectures while planing the achievement of subgoals. The systems parameters are then learned in end-to-end fashion using backpropagation (Rumelhart et al., 1986).

Originally, Bakker and Schmidhuber (2004) introduced the HASSL (Hierarchical Assignment of Subgoals to Subpolicies Learning) algorithm. A high-level policy is tasked with the identification of meaningful intermediate achievements that predate top-level goals (Bakker and Schmidhuber, 2004, p.1). Low-level policies at the same time master subgoals which were set by the next higher level. High level goal states are concatenated with the lower level states in order to create specialized behavior which generalizes within a certain part of the state space (Bakker and Schmidhuber, 2004, p.2). Other two-layer architectures allow learning at different time-scales (Kulkarni et al., 2016a; Vezhnevets et al., 2017). A meta-controller/manager module learns subgoal signals from the external environment while the low level controller/worker module learns how to achieve those using primitive actions. Furthermore, there are multiple attempts to combine recurrent structures (Vezhnevets et al., 2016), the notion of successor representations (Kulkarni et al., 2016b), meta learning (Frans et al., 2017) as well as intrinsic motivation (Florensa et al., 2017) and soft proxy rewards. Lately, Co-Reyes et al. (2018) introduced a novel architecture which learns a latent space of behaviors. Thereby one leverages a continuous and infinite set of temporally-extended actions. A behavior parametrizes a policy and a decoder predicts a trajectory of state visitations (Co-Reyes et al., 2018, p. 2). Together with Model-Predictive Control one performs planning at a coarser timescale. Independent of this a Proximal Policy Optimization (Schulman et al., 2017) is trained to explore trajectories by maximizing the generated entropy.

While providing a fully algorithmic approach, such attempts lack interpretability and often times need severe amounts of pre-training. They loose the functional interpretability of options as sub-policies and are able to overfit abstract behaviors. Our approach, on the other hand, directly leverages the hierarchical structure captured by sequences of behavior or state transitions. Thereby, we obtain an encoding which decomposes different levels of abstraction.

Furthermore, in chapter 4 we introduce an information-theoretic mechanism which prevents overfitting of noisy sequences. Hence, we maintain the important property of interpretability and reduce the computational load.

This chapter has introduced the formal HRL framework in which we develop our grammar-based approach for automating the discovery of temporally-extended actions. We covered the general MDP setup and its temporal generalization in SMDPs. Furthermore, we discussed how macro-actions as well as options constitute SMDPs when defined over the core MDP. Eligibility traces provide another form of extended value information propagation and we outlined an algorithm that combines macro-actions with eligibility traces. Finally, we argued that the current state of HRL research lacks a sufficient answer to the hierarchical substructure discovery problem.
Language, on the other hand, is hierarchically structured and its constituents can be inferred using computational tools based on formal grammars. In the following section we now turn to the foundations of grammatical inference and numerical measures which quantify the information-theoretical complexity of parsing a sequence of symbols.

Chapter 3

Language and Behavior

The previous section has introduced several different approaches to subgoal discovery. We have seen that many of them require a large amount of pre-training and cannot be integrated in to the learning process itself. Thereby, computational inefficiencies were introduced. Our approach, on the other hand, leverages the hierarchical structure innate to human language and its comprehension. The following chapter introduces key concepts from grammatical inference and formal grammars. We first motivate a strong parallelism between language comprehension and motor control. Afterwards, we introduce notation and compare context-free and probabilistic context-free grammars. Such hierarchical structures can be discovered using tools from grammatical inference. Finally, we review syntactic surprisal which is going to have a key role in discovering temporally-extended actions.

3.1 Motor Control and Language

Lashley (1951) was first to hypothesize that grammatical structure applies to both language and goal-driven behavior. Since then multiple studies have provided clear evidence that Broca's area is not only concerned with language-specific hierarchical computations (Brennan et al., 2016; Ding et al., 2017; Nelson et al., 2017) but also with action-to-goal related processes (Fazio et al., 2009; Fadiga et al., 2009). Despite this and to the best of our knowledge, only a few studies have made the effort to intelligently link computational models of behavior with the structural paradigm of language (e.g Aloimonos (2008); Aloimonos et al. (2010); Pastra and Aloimonos (2012); Stout et al. (2018)). Furthermore, most of these have been centered around the usage of tools as extensions to specific body parts (Pastra and Aloimonos, 2012, p. 106). In the following we intend to give a simple example of how a grammatical structure might help a RL agent to solve problems efficiently.

Pastra and Aloimonos (2012, p. 104) formulate a minimalist programme (Chomsky, 1995) conceptual class whose general principles apply to both language as well as other biological systems. Syntax provides a parsimonious modeling framework which allows for efficient computation for both generation and parsing of time-dependent sequences. In their specification they disregard the notion of sub-goals (Pastra and Aloimonos, 2012, p. 107f.). By citing neuroscientific evidence, they argue that the final and global goal influences all actions and that the first action already expresses clear signs of intention. Instead, minimalist programme augment a CFG with the following transformations: recursion, merge and move (Pastra and Aloimonos, 2012, p. 105f.). The action grammar then consists of primitive actions, action strings or phrases and action features such as tool/affected-object complement and goal

(Pastra and Aloimonos, 2012, p. 106.). Figure 3.1 describes one such action grammar and the corresponding parse tree for a specific goal-driven sequential behavior, namely inserting a key in order to open a door. We follow Pastra and Aloimonos (2012, p. 108) and differentiate between primitive actions (a), entities (e), non-terminals ($\tilde{A}$) and the maximal projection (also known as goal) of the action structure (A_{goal}). In this specific case all sub-actions are related by sequential temporal conjunctions. Entity references (E), on the other hand, are related by either action-object or action-tool complements.

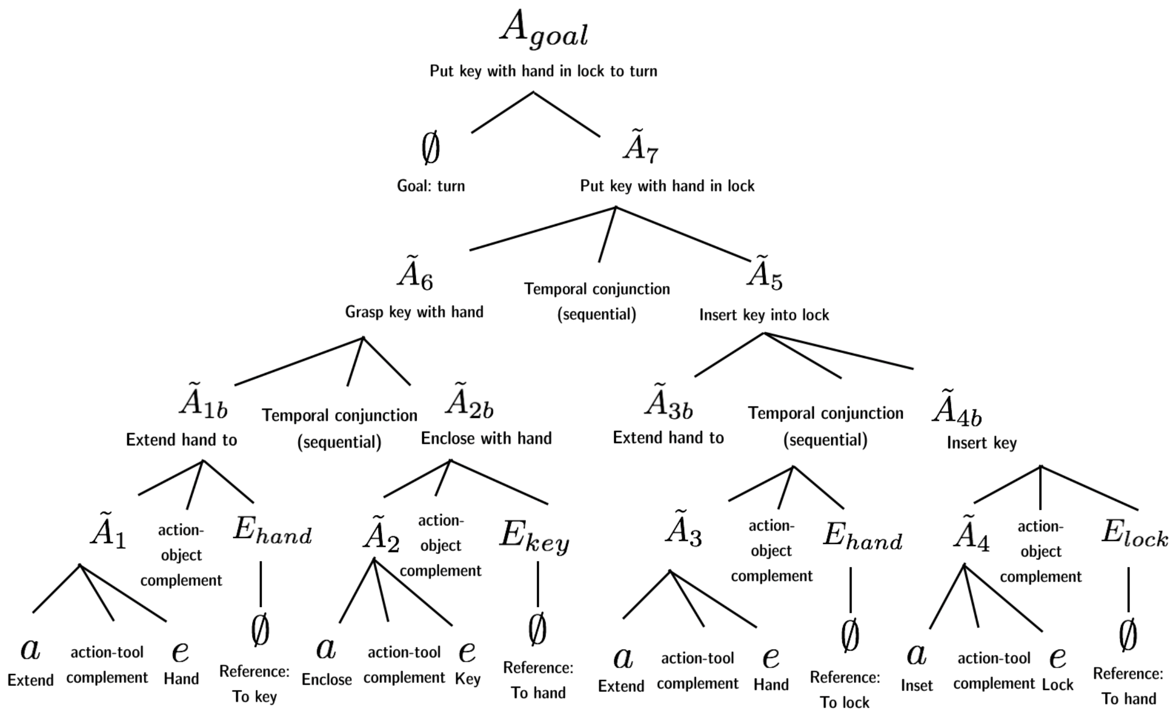


Figure 3.1: Action Grammars Parse Tree. Extended from Pastra and Aloimonos (2012, p. 108)

Now imagine a RL agent trying to solve the task of Montezuma's Revenge. In order to sufficiently solve the task, the agent has to first safely pickup a key and afterwards deliver it to a goal location. This goal achievement procedure can be seamlessly retrieved and executed by simply parsing/sequentially executing actions provided by the grammar encoded in the parse tree above. Hence, the parse tree acts as a form of fast thinking system which automatically solves a sub-task of the overall task. This reformulation of sub-goal achievement as following a grammar-prescribed sequence of terminal vocabulary (primitive actions) motivates the action grammar framework outlined in chapter 4. Ultimately, we are interested in extending this school of thought to the Reinforcement Learning world in which an agent episodically interacts with a stimulus and reward providing environment. By doing so, we will be able to both better understand what makes hierarchical decomposition so efficient and to improve the performance of HRL agents.

Now that we have seen how action grammars are useful to solve hierarchically structured tasks, the next sections introduce the required background and notation in formal grammars and how to obtain estimates of such grammars.

3.2 Formal Grammars

Formal grammars and the theory of computational linguistics study both generating and accepting systems that underlie a language. Given a start symbol S a formal grammar $(\Sigma, \mathbb{N}, S, \mathcal{P})$ produces an output which is a string of words. They are made up of multiple components (see e.g. Levelt (2008)):

- Σ: A terminal vocabulary which is a set of terminal elements used to construct the sentences of a language. For convenience we stick with the convention that individual elements are denoted by lower case latin letters.
- $\mathbb{N}$: A non-terminal vocabulary (or variables denoted by upper case latin letters), on the other hand, is a set of elements which are only used in the process of deriving a sentence.
- $V^\star$: The set of all possible strings of possible elements in the vocabulary.
- V^+: The set of all possible strings of possible elements except for the null-string in the vocabulary.
- $\mathcal{P}$: Production rules are then defined as ordered pairs of strings such that $\alpha \to \beta, \alpha \in V^+, \beta \in V^\star$.

3.2.1 Context-Free Grammars

Following Chomsky's hierarchy of grammars (see figure 3.2; Chomsky (1959a,b)) a type-2 grammar, also known as context-free grammar is such that the production rules have the following form (Levelt, 2008, p. 11):

$$A \to \beta \text{ , where } \beta \neq \lambda \text{ or } |\beta| \neq 0$$

Since production rules either map from one-to-one, one-to-none or one-to-many they are called context-free. The context of a non-terminal symbol does not influence the production rule (Pastra and Aloimonos, 2012, p. 105). A context-free grammar that is non-branching and loop-free is called a straight-line grammar (Siyari and Gallé, 2016, p. 81). Such grammars are very restrictive since they are only able to generate a single sentence.

3.2.2 Probabilistic Context-Free Grammars

A probabilistic CFG $(\Sigma, \mathbb{N}, S, \mathcal{P}, p)$ generalizes the notion of a CFG by adding a probability vector $p : \mathcal{P} \to [0, 1]$ (Levelt, 2008, p. 33f.). The vector is a proper probability distribution, hence for all $A \in \mathbb{N}$ we have

$$\sum_{A \to \beta \in \mathcal{P}(A)} p(A \to \beta) = 1 \text{ ,where } \mathcal{P}(A) = \{A \to \alpha : A \to \alpha \in \mathcal{P}\}.$$

Intuitively, p specifies the likelihood of rewriting a non-terminal string. The probability of a specific derivation is simply the product of its productions. Hence, PCFGs capture the frequency with which certain types of sentences occur in a language.

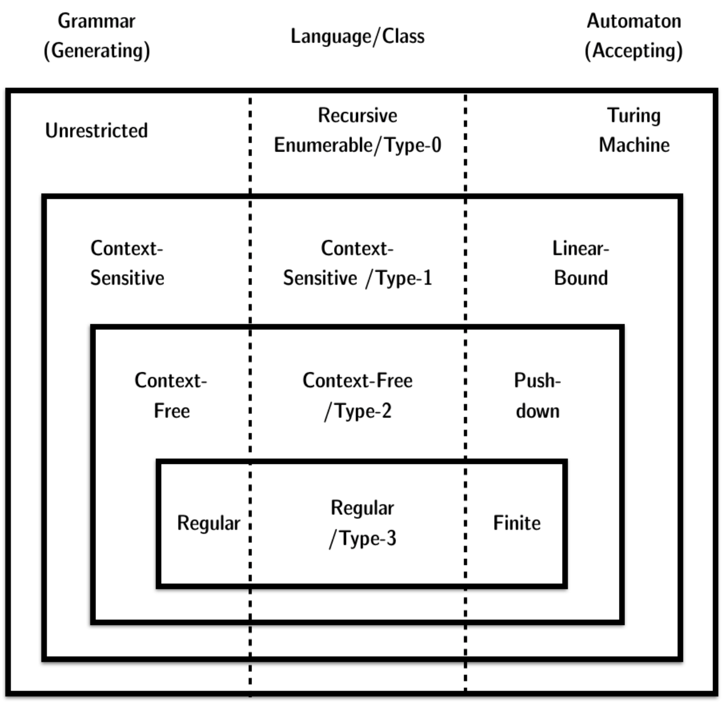


Figure 3.2: The Chomsky Hierarchy. Altered from Levelt (2008, p. 12)

3.3 Grammatical Inference

The process of inferring a grammar for a language that is consistent with a given sample of sentences is called grammatical inference or grammar induction (Levelt, 2008, p. 109f.).

3.3.1 Grammar Induction for Context-Free Grammars

The smallest grammar problem (Charikar et al., 2005; Siyari and Gallé, 2016) formalizes the problem of finding the smallest CFG which compresses a string generated by a straight-line grammar. The size of a grammar is defined by the length of concatenated symbols needed to decode the encoded string. It is well known that this problem is NP-hard (Charikar et al., 2005, p. 2). Two algorithms that greedily approximate the smallest CFG are Sequitur (Nevill-Manning and Witten, 1997) and Lexis (Siyari et al., 2016).

The Sequitur algorithm infers a CFG in a greedy fashion. Given a single sentence of the language, Sequitur sequentially reads in all symbols and collects repeating subsequences of symbols into a production rule. The construction imposes two constraints at all times. First, the final encoded string is only allowed to have unique bigrams (*Digram Uniqueness*, Nevill-Manning and Witten (1997, p. 69)). If this was not the case, the algorithm could replace the bigram by a non-terminal symbol and add a corresponding production rule. Second, production rules have to be used more than once in the derivation of the string (*Rule Uniqueness*, Nevill-Manning and Witten (1997, p. 70)). Otherwise, we could simply replace it by the corresponding terminal vocabulary and thereby reduce the size of the grammar. A simple example with $\Sigma = \{a, b, c, d\}$ can be found below:

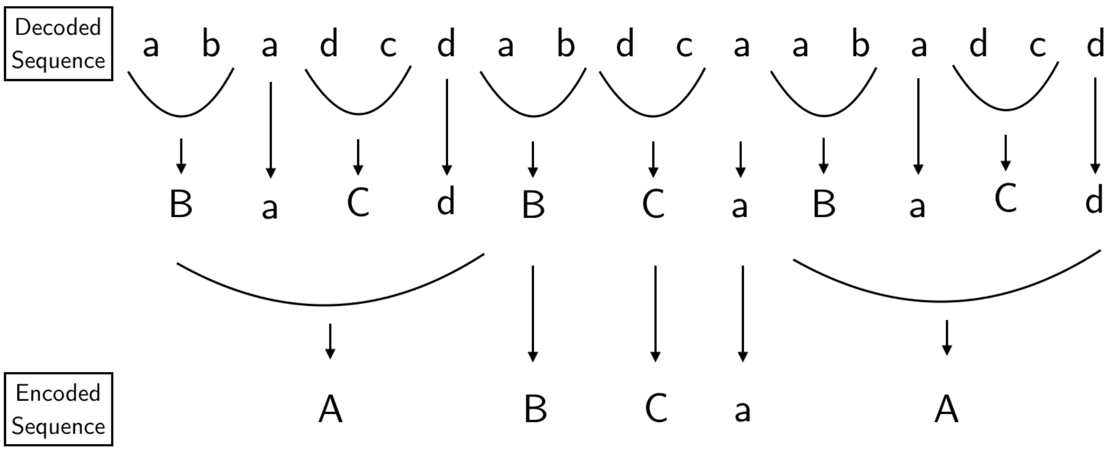


Table 3.1: A 2-Sequitur Encoding Example

It is well known that Sequitur tends to overfit noise in observed sentences. As the length of the sequence increases, Sequitur greedily encodes the sequence by adding non-terminal symbols as soon as a sub-sequence is scanned twice. Similar to classical supervised learning, one might thereby not be able to accurately capture the true underlying data generating process. Multiple alternatives have been proposed to combat this problem.
k-Sequitur (Stout et al., 2018, p. 5) for example generalizes this approach by replacing a bigram with a rule only if the bigram occurs at least k times. As k increases the discovered CFG grammar becomes less and less sensitive to overfitting noise and the resulting grammar is more parsimonious in terms of productions.
Lexis (Siyari et al., 2016), on the other hand, provides an optimization-based alternative which iteratively constructs a directed acyclic graph (DAG), the so-called Lexis-DAG. The graph consists of alphabet, intermediate and target nodes and has to satisfy three properties (Siyari et al., 2016, p. 2): First, every node in the DAG represents a string. Second, every intermediate or target node represents string which is generated by the concatenation of substrings and represent hierarchical substring concatenations. Finally, intermediate nodes have at least two outgoing edges. Hence, the substrings which they constitute are used in at least two following concatenations which is reminiscent of the rule utility principle of the Sequitur algorithm. Starting from a trivial graph which connects a set of target sentences with the set of elements in the terminal vocabulary, we seek to construct the Lexis-DAG by adding intermediate nodes (Siyari et al., 2016, p. 2). The indirect objective thereby is to minimize a cost function (e.g. number of concatenations or DAG edges - equivalent algorithms) while imposing that the constructed graph satisfies all three Lexis-DAG properties (Siyari et al., 2016, p. 2f.). Again, this problem by itself is NP-hard. G-Lexis, the greedy algorithmic implementation, searches for substrings that will lead to a maximal reduction in the cost, when added as new intermediate node (Siyari et al., 2016, p. 3f.). It does not account for longer dependencies.
IGGI (Information Greedy Grammar Inference, Schoenhense et al. (2017)) sits conceptually on top of G-Lexis and combats the phenomenon of "overfitting" by introducing a decision rule based on the decrease in Shannon entropy gained from introducing the production. IGGI thereby provides a mechanism to regularize the DAG constructed by G-Lexis and one can avoid fitting noise in non-hierarchical string sequences (Schoenhense et al., 2017, p. 7). Furthermore, IGGI captures the Bayesian notion of Occam's Razor and performs very data-efficient.

3.3.2 Grammar Induction for Probabilistic Context-Free Grammars

PCFGs, on the other hand, can be inferred from visible data by the means of the inside-outside algorithm (Lari and Young, 1990). The inside-outside algorithm is an Expectation-Maximization type algorithm which makes use of dynamic programming and iteratively computes expected frequencies of production rules. Afterwards, it constructs their probabilities. It thereby provides a point estimate of the probabilities associated with a specific parse tree. As always, Bayesian approaches can provide a sophisticated notion of uncertainty. For example, Hidden Markov Models naturally lead themselves to filtering distributions which we will later see to relevant for our notion of replay surprisal. Furthermore, we can also employ a Bayesian Multinomial-Dirichlet model (see for example (Stolcke, 1994)).
Hidden Markov Models are generative probabilistic models. Given a hidden variable z with θ hidden states, a HMM consists of a Markov Chain described by the transition matrix $p(z_t|z_{t-1}) \in \mathbb{R}^{\theta\times\theta}$, an emission distribution $p(w_t|z_t) \in \mathbb{R}^{|\Sigma|\times\theta}$ and an initial distribution for the first hidden state $p(z_1) \in \mathbb{R}^{\theta}$.

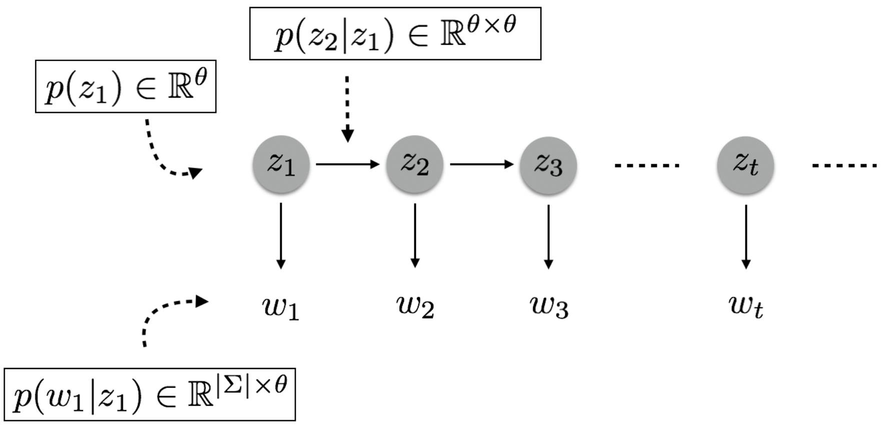

$p(z_2|z_1) \in \mathbb{R}^{\theta\times\theta}$
$p(z_1) \in \mathbb{R}^{\theta}$
z_1
z_2
z_3
z_t
w_1
w_2
w_3
w_t
$p(w_1|z_1) \in \mathbb{R}^{|\Sigma|\times\theta}$

Figure 3.3: Hidden Markov Model

They are trained by repeating alternating expectation and maximization steps until convergence. In practice this is achieved by the Baum-Welch algorithm which consists of a backward pass which computes the joint distribution $p(w_1, \ldots, w_t, z_t)$ and a forward pass which computes $p(w_{t+1}, \ldots, w_T|z_t)$. These two distributions together together allow us to compute posteriors and to perform the expectation step. Afterwards, one easily updates the parametrized transition, emission and initial distributions by maximizing the expected data log-likelihood.

3.4 Information-Theoretical Complexity

Language comprehension is an active field of computational linguistics which intends to study the sequential parsing of strings while inferring global meaning of the sentence. It combines the static and generative notion of formal grammars with the dynamic and accepting elements of automata (Hale, 2014). Probabilistic grammars formalize prior beliefs about the distribution of sub-structures in human language (Hale, 2014, p. 84). They associate probabilities with specific derivations and thereby formalize a notion of anticipation. Given a pre-string how likely is it to observe a specific one at the next time step? Both, syntactic surprisal (Hale, 2001; Levy, 2008) and entropy reduction (Hale, 2003, 2006) transform this conditional distribution in order to quantify the amount of information-processing required to process the successor. A strongly unexpected string leads to a large amount of necessary belief updating. Intuitively, syntactic surprisal (Hale, 2001; Levy, 2008) measures the difficulty generated by replacing a previous conditional distribution with the revelation of a new element in the sequence. Equivalently it measures the amount of self-information provided by revealing the next element in the sequence of strings of a sentence. Mathematically, it is the logarithm with base 2 of the reciprocal of the conditional distribution. Hence, it is measured in bits and can be regarded as the amount of information revealed due to observing *this* particular element instead of any other. Given a grammar G and sentence i let w_j denote the word at position j. Syntactic surprisal (Levy, 2008, p. 1130f.) is then defined as

$$Surprisal_t^{i,G} \propto -\log_2 p^G(w_{t+1}^i | w_{1,\dots,t}^i) = -\log_2 p_t^G(w_{t+1}^i) \geq 0$$

It can be easily constructed for any form of PCFG that provides some a filtering distribution. E.g. a Hidden Markov Model with k hidden states lets us compute the conditional in the following way:

$$p(w_{t+1}|w_1, \dots w_t) = \sum_{z_{t+1}=1}^{k} \underbrace{p(w_{t+1}|z_{t+1})}_{\text{Emissions}} \underbrace{p(z_{t+1}|z_t)}_{\text{Transitions}} \underbrace{p(z_t|s_1, \dots, s_t)}_{\text{Filtering}}$$

Surprisal is able to capture many phenomena in human language comprehension such as *Garden Path Sentences* and object relative clauses (Hale, 2014, p. 35).
Entropy reduction, on the other hand, quantifies changes in the structure associated with the conditional distribution (Hale, 2014, p. 88). The entropy is simply the expectation of the surprisal:

$$\mathcal{H}(i;G) \propto -\mathbb{E}[\log_2 p^G(w_{t+1}^i|w_{1,\dots,t}^i)] = -\sum_{w_{t+1}} p^G(w_{t+1}^i|w_{1,\dots,t}^i) \log_2 p^G(w_{t+1}^i|w_{1,\dots,t}^i)$$

If the entropy is reduced after a new word has been processed ($\Delta_{t,t+1}\mathbb{E}[Surprisal^{i,G}] < 0$) the conditional distribution becomes more peaked around specific values. This represents certainty in what the parser expects next. The conditional distribution has become more structured. Following Hale (2003, p. 105), the entropy reduction hypothesis states that a strong reduction reflects large processing efforts. Similarly to syntactic surprisal, a strong reduction in entropy is associated with "mental work".
Compared with syntactic surprisal, entropy reduction captures the notion of relative frequency of the derivations. Hale (2014, p. 89) states that this leads to better empirical performance when profiling human language comprehension.
In later sections we will make use of the surprisal measure to stop an option execution or the generative process of macro-actions if a certain threshold is exceeded. Intuitively, a HRL agent which has learned some structural aspects of the environment will want to circumvent surprising transitions in state space. Surprisal thereby measures a form of uncertainty within the executed sub-policy. Instead of continuing the execution of the current grammar-based option, control will be returned to the next higher level.
This chapter has introduced the required formal knowledge necessary to merge grammatical inference with Hierarchical Reinforcement Learning. First, we motivated a parallel point-of-view between hierarchically structured behavior execution and language comprehension. Afterwards, we defined the technical terms and conventions required to reason about grammatical structures. The process of discovering such structures was reviewed and we

highlighted several difficulties that are associated with capturing the true underlying generating process. Finally, we discussed information-theoretic measures of effort in sentence parsing.

In the next chapter we will now combine both fields and derive algorithms which are able to leverage innate advantages: Efficient learning with hierarchical substructures within HRL and efficient discovery of such substructures with the help of computational linguistics.

Chapter 4

Action Grammars

The following section introduces the core contributions of this project. Section 2.4 concluded that current Hierarchical Reinforcement Learning approaches are in need for a sample efficient and interpretable end-to-end algorithm that extracts hierarchical constituents. Furthermore, we saw that grammar induction provides such a sample efficient mechanism in the context of language. Such algorithms infer the generating process conditioned by the formal grammar. It comes only natural to combine both.
We will now make a crucial assumption that connects linguistics with eager behavior:

Assumption 1. *Observed episodic behavior (with trajectory $\vartheta = \{\vartheta_1, \dots, \vartheta_T\}$ where $\vartheta_t = \{s_t, a_t\}$) can be equivalently viewed as sentences sampled from the language, $L(G)$ with $G \sim \pi|E$.*

Action sequences as well as communication by the means of words both convey meaning and are goal-directed. Furthermore, both consist of hierarchical structures and are conditioned by the environment in which they are uttered in. By treating sequential behavior as sentences, we are able to extract temporally-extended actions by applying grammatical inference techniques (see section 3.3). In general, the language of actions depends on two factors. First, trajectories of a RL agent are generated by a policy π resulting from the current value estimates. Second, the environment E (i.e. the start and goal location as well as the transition success rate) dictates the length of the sentences as well as the allowed sub-sequences. Hence, the sampled sentences or trajectories encode valuable information. Not only do they convey self-information but they also encode structures within the environment.
Therefore, we propose to apply grammar induction to an observed history of state-action transitions of an agent that interacts in a goal-directed fashion within a structured environment. The overall learning process can thereby be split into two alternating steps: *Action learning* and *grammar learning*. Grammar Learning describes the process by which the agent extracts information from her observations. Given sampled state-action sentences from previous agent-environment interactions the agent extracts hierarchical constituents of his current policy. These constituents can then be transformed to alter the action space of the agent. We interpret the augmented action space as an estimate of the production rules underlying the true action grammar. During action learning, on the other hand, the agent refines his "action grammar"-value estimates using SMDP-Q-Learning. The overall learning procedure then consists of alternating updates of the grammar estimate and a refinement of the corresponding value estimates based on reinforcement signals. Hence, the agent switches between refining his structural assumptions of the environment and updating her value estimates of inferred temporally-extended actions.
This bilateral paradigm introduces another exploration-exploitation trade-off within the overall learning problem. Not only does the agent have to balance the expected value and

uncertainty associated with state-action pairs. She also has to choose whether to exploit the hierarchical sub-structures previously discovered or to update her action space and further explore and extend her experience buffer. A graphical illustration can be seen in figure 4.1:

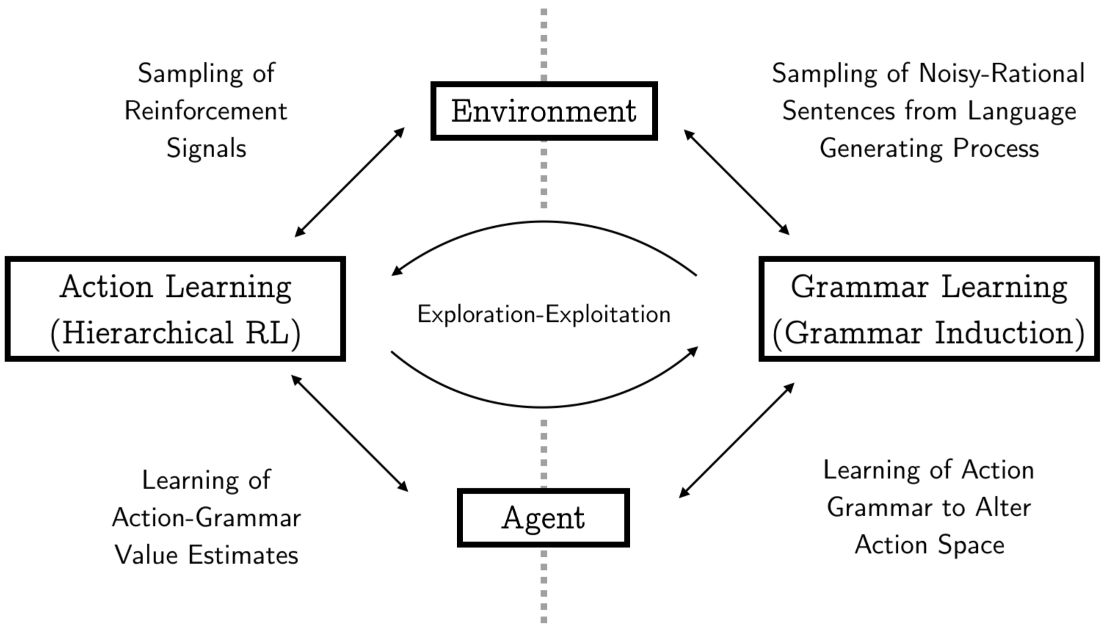

Sampling of Reinforcement Signals
Environment
Sampling of Noisy-Rational Sentences from Language Generating Process
Action Learning (Hierarchical RL)
Exploration-Exploitation
Grammar Learning (Grammar Induction)
Learning of Action-Grammar Value Estimates
Agent
Learning of Action Grammar to Alter Action Space

Figure 4.1: A New Exploration-Exploitation Trade-Off: Grammar and Action Learning

Instead of handcrafting sub-policies or learning sub-structures in a non-interpretable way, we derive an approach which extracts temporally-extended actions with the help of grammatical inference. By separating structure discovery in grammar learning from policy optimization in action learning, the agent is able to leverage powerful ideas from both worlds of computational linguistics as well as Hierarchical Reinforcement Learning.
In the following, we outline different frameworks which incorporate the bilateral relationship between structure discovery and structure application. We explore different grammar definitions in the context of HRL and compare results in several learning environments. While doing so, we will distinguish between an imitation learning task and a online reinforcement learning task. During the imitation learning task the agent is going to observe an expert. She will infer latent structures from the expert behavior and adapt her action space accordingly. Afterwards, the action learning process starts. We will show how action grammars provide an automatic way of discovering optimal sub-policies. The online reinforcement learning agent, on the other hand, will learn a grammar of his own behavior. Here grammar induction provides a form of reflection process. The agent reviews his previous actions and extracts patterns which generalize throughout the state space. This approach has clear parallels to the classical basal ganglia-inspired Actor-Critic (AC; e.g. Takahashi et al. (2008)) paradigm in which a critic is concerned with learning a state-value function while the actor learns to execute actions. Here, similarly a grammar critic module learns to structure the temporally-extended action space. This chapter proceeds as follows. First, we formally state the problem and the general approach. Afterwards, we show how one can incorporate this general line of thought into the automatic discovery of both macro-actions and options. Finally, we will discuss an efficient syntactic surprisal-based metrics to compare different grammatical substructures in the context of HRL.

4.1 Problem Formulation

Let us assume that the optimal policy of a Reinforcement Learning agent is hierarchically structured for a specific environment E. The optimal policy $\pi^\star$ then consists of a hierarchy of subgoal achievements which increase in sequential difficulty when moving up the hierarchy. We define the terminal vocabulary Σ to consist of the primitive action space $\mathcal{A}$, hence $\Sigma = \mathcal{A}$. A trajectory obtained from traversing the current policy π is viewed as a sample from the language generated by the grammar $L(\pi|E)$. We write $\vartheta^i \sim L(\pi|E)$ for $i = 1, \dots N_g$ trajectories. Since we assume an episodic reinforcement learning task in which an episode ends with the achievement of the goal, each trajectory has an individual length denoted by T_i. Given a set of trajectories, $\vartheta^1, \dots \vartheta^{N_g}$, we construct a grammar training set from which we infer a (probabilisitic) context-free grammar using one of the algorithms outlined in section 3.3. Thereby we obtain a grammar estimate $\hat{G}$. Afterwards, we transform this grammar into temporally-extended actions such as macros ($\mathcal{M}^{\hat{G}}$) or options ($\mathcal{O}^{\hat{G}}$). We augment the action space of the HRL agent, e.g. $\mathcal{A}^{\hat{G}} = \mathcal{A} \cup \mathcal{M}^{\hat{G}}$. The HRL agent can then use this new action space in his further learning. A graphical illustration of the **grammar learning** phase is shown in figure 4.2:

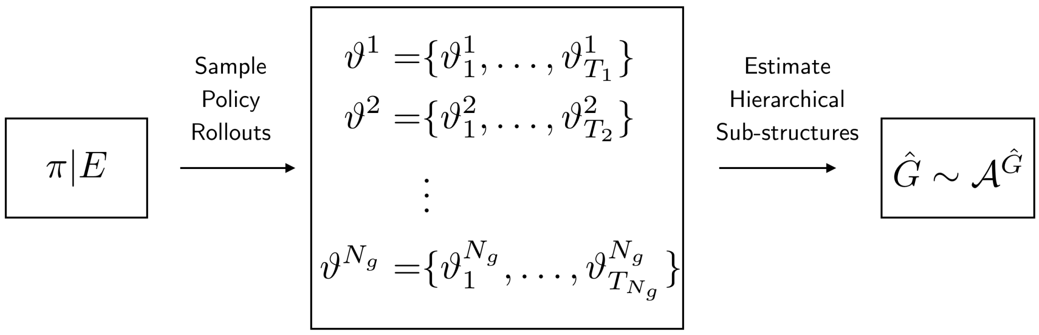

$\pi|E$
Sample Policy Rollouts
$\vartheta^1 = \{\vartheta^1_1, \dots, \vartheta^1_{T_1}\}$
$\vartheta^2 = \{\vartheta^2_1, \dots, \vartheta^2_{T_2}\}$
$\vdots$
$\vartheta^{N_g} = \{\vartheta^{N_g}_1, \dots, \vartheta^{N_g}_{T_{N_g}}\}$
Estimate Hierarchical Sub-structures
$\hat{G} \sim \mathcal{A}^{\hat{G}}$

Figure 4.2: Exploring Past Experiences: Grammar Learning

Grammar learning thereby introduces a reflective period in which the agent takes a bird's eye-view on the observed behavior. The grammar inference process identifies repeating patterns that led to successful goal achieving experiences. By extracting these patterns and redefining them as temporally-extended actions, we additionally *save* the progress made not only in the value estimate but also in the augmented action space. During the action learning phase, on the other hand, the agent interacts with the environment and receives reinforcement signals from the environment. She updates her value estimates and refines her behavior. More specifically, the temporal difference error for episode i at time j with waiting time τ is denoted by

$$\delta^i_j(s_j, a_j) = \sum_{k=1}^{\tau} \gamma^{k-1} r^1_{t+k} + \gamma^\tau \max_{a' \in A^{\hat{G}}} Q(s', a') - Q(s_j, a_j) \; \forall i = 1, \dots, N_a \text{ and } j = 1, \dots, \tau, \dots T_i.$$

Together they shape the action-grammar value estimate $Q(s, a^{\hat{G}})$ and during an action training phase of N_a episodes we are able to update the estimate to $Q'(s, a^{\hat{G}})$.[1] Depending on whether

[1]In the next sections we will outline different ways of transferring learned value estimates between action grammar updates such that we do not loose any progress between different phases.

we model the agent under the AC umbrella we separately update the policy or can extract it using the $\max_a$ operator on $Q(.,a)$. **Action learning** is illustrated in figure 4.3:

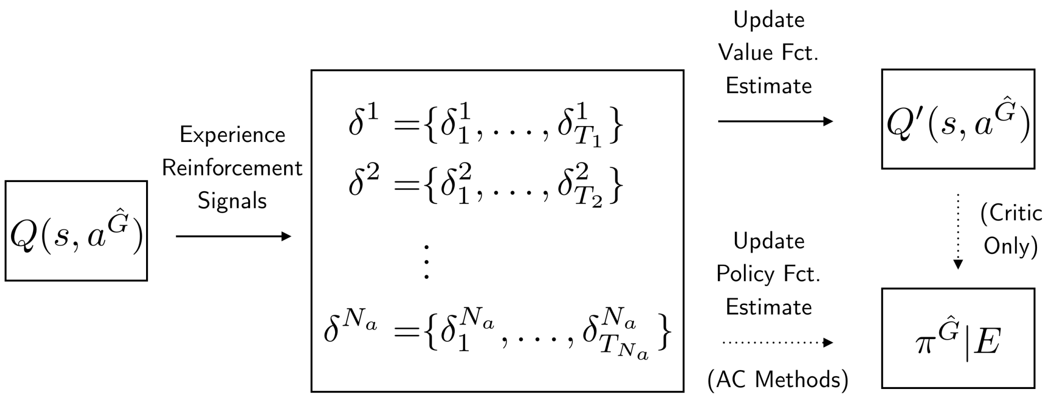

$Q(s, a^{\hat{G}})$
Experience Reinforcement Signals
$\delta^1 = \{\delta_1^1, \dots, \delta_{T_1}^1\}$
$\delta^2 = \{\delta_1^2, \dots, \delta_{T_2}^2\}$
$\vdots$
$\delta^{N_a} = \{\delta_1^{N_a}, \dots, \delta_{T_{N_a}}^{N_a}\}$
Update Value Fct. Estimate
Update Policy Fct. Estimate
(AC Methods)
$Q'(s, a^{\hat{G}})$
(Critic Only)
$\pi^{\hat{G}}|E$

Figure 4.3: Exploiting New Experiences: Action Learning

Ultimately, we are interested in learning $\pi^\star$. By alternating between grammar updates and value updates the agent is able to iteratively update his action space and to learn how to put those actions to use. Since temporally-extended actions are especially powerful in the first stages of learning, the frequency of grammar estimate updates can be decreasing. During the beginning of the overall learning process we update often. As learning becomes more stable, the need to refine the underlying grammar decreases as well. Furthermore, as the agent learns, the length of the sampled sentences decreases. Since the agent reaches the goal earlier and since the grammar inference algorithms are usually not robust to the length of the input sequence, the hyperparameters of the grammar induction algorithm also have to be adjusted. The complete learning process can be viewed below.

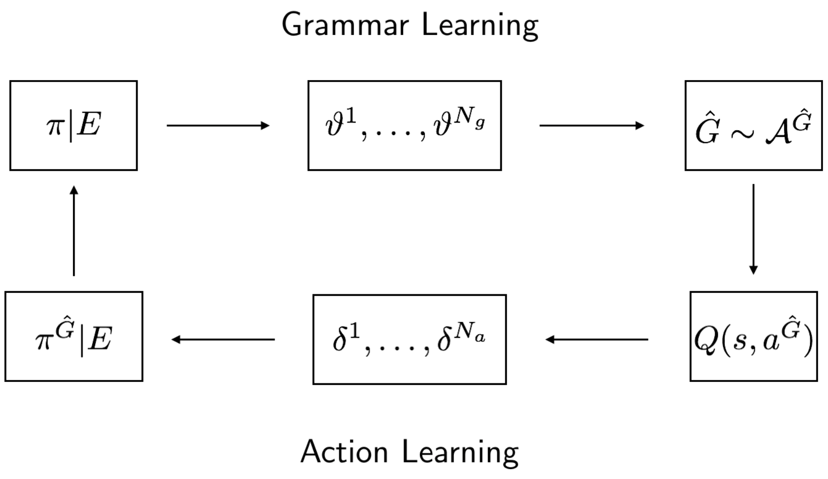

Grammar Learning
$\pi|E$
$\vartheta^1, \dots, \vartheta^{N_g}$
$\hat{G} \sim \mathcal{A}^{\hat{G}}$
$Q(s, a^{\hat{G}})$
$\delta^1, \dots, \delta^{N_a}$
$\pi^{\hat{G}}|E$
Action Learning

Figure 4.4: Closed Loop: Grammar Learning and Action Learning

Now that we have established the general paradigm, we are ready to introduce different algorithmic procedures to define semantically meaningful temporally-extended actions using context-free as well as probabilistic context-free grammars. Furthermore, we explore the importance of grammar and learning hyperparameters and their relationship. We start with macro-actions for deterministic environments and continue with option discovery for stochastic environments.

4.2 Macro-Action Discovery

As introduced in section 2.3.1 macro-actions provide a powerful solution to long-term credit assignment problems and exploration in such. Since they prescribe the deterministic execution of fixed sequences of actions, they are not well suited for stochastic environments (see figure 2.5). In this subsection we will introduce two procedures which allow us to identify macro-actions suitable for efficient HRL:

1. **CFG Macro-Actions**: Following the *context-free action grammar* approach we define macro-actions by flattening productions rules extracted by any CFG inferring algorithm. The grammar regularization parameter (e.g. k in the k-Sequitur algorithm) is treated as a learning parameter which can be adapted during learning (e.g. similar to temperature annealing or exponential decay).
2. **Hidden Markov Model Macro-Actions**: The *probabilistic action grammar* framework, on the other hand, allows us to sample macro-actions from a generative model. The length of the action sequence is determined by a surprisal threshold which again will be treated as an adaptive learning parameter.

The following sub-sections give details. Empirical test for the macro-action discovery problem can be found in section 5.1.

4.2.1 CFAG: CFG Macro-Actions

Let us start with an thought experiment. Assume that the space of primitive action and therefore the terminal vocabulary is given by

$$\mathcal{A} = \Sigma = \{a, b, c, d, e, f\}$$

where each $a \in \Sigma$ corresponds to some control command in the deterministic environment a reinforcement learning agent lives in. Furthermore, let us assume that the current greedy and deterministic policy of the agent (ϑ) is given by the 63 letters in the first row of table 4.1. As motivated in the previous section we treat this sequence as a sentence generated by a straight-line grammar. We are then able to encode this string using any context-free grammar induction approach. Table 4.1 displays three possible CFG extractions, their corresponding production rules (PR) and a few compression statistics.
We denote the encoded sequence by ϑ^{enc}, the compression ratio by $\frac{\vartheta^{enc}}{\vartheta}$ and the ratio of empirical entropies by $\frac{\mathcal{H}(\vartheta^{enc})}{\mathcal{H}(\vartheta)}$. 2-Sequitur, 3-Sequitur and G-Lexis derive very different grammatical structures. 2-Sequitur and G-Lexis compress ϑ very strongly, while 3-Sequitur provides a more conservative encoding which increases the entropy. The symbols used in the encoding are more uniformly distributed than in the decoded policy sequence. Applying the entropy reduction intuition from section 3.4, we can conclude that the syntactic workload has decreased. Furthermore, strong compression is associated with a deeper grammatical hierarchy (only in the 2-Sequitur case) and longer recursively flattened production rules.
During the grammar learning phase the agent defines a set of macro-actions based on the recursively flattened production rules, where $|\mathbb{N}| = |\mathcal{M}|$. More specifically for the 3-Sequitur case, we define the new action space of an agent as

Policy (ϑ): abdaefabdcedabdaefaedcefabdaefabdcedabdcefaedcedabdaefabdcedabd						
Algorithm	2-Sequitur		3-Sequitur		G-Lexis	
ϑ^{enc}	BCDfBEfDdBF		BCDBEdFfGdaefCDabdcefEdDBCEdC		BaCfBcCdB	
$\frac{\vartheta^{enc}}{\vartheta}$	0.1746		0.4603		0.1429	
$\frac{\mathcal{H}(\vartheta^{enc})}{\mathcal{H}(\vartheta)}$	1.0702		1.3613		0.9721	
$\mathbb{N}$	PR	Flat ($\mathcal{M}^{2-seq}$)	PR	Flat ($\mathcal{M}^{3-seq}$)	PR	Flat ($\mathcal{M}^{lexis}$)
B	CEd	abdaefabdced	CEf	abdaef	DaefDcedD	abdaefabdcedabd
C	FGf	abdaef	Gd	abd	-	efaedce
D	GdH	aedce	Fd	ced	-	abd
E	FH	abdce	-	ae		
F	-	abd	-	ce		
G	-	ae	-	ab		
H	-	ce				

Table 4.1: Macro-Action Construction with CFG Encoding of a Deterministic Policy.

$$\mathcal{A}^{\hat{G}} = \mathcal{A} \cup \mathcal{M}^{3-seq} = \{a, b, c, d, e, f\} \cup \{abdaef, abd, ced, ae, ce, ab\}$$

Afterwards in the action learning phase, she utilizes the inferred action grammar and learns with the help of SMDP-Q-Learning. An overall algorithmic approach would be as follows: We start by initializing the agent as a Q-learner and learn a set of action values from N_{init} episodes. Afterwards, we obtain a set of rollouts and their corresponding action sequences. We choose the best ϑ (minimal number of steps to goal, $N_g = 1$), extract flattened production rules and run a CFG inference algorithm with θ hyperparameters. Afterwards, we alter the macro-action space and run Macro-Q-learning for a set N_a episodes. We update the hyperparameter k and repeat the updating of the action space by estimating a new set of macros.

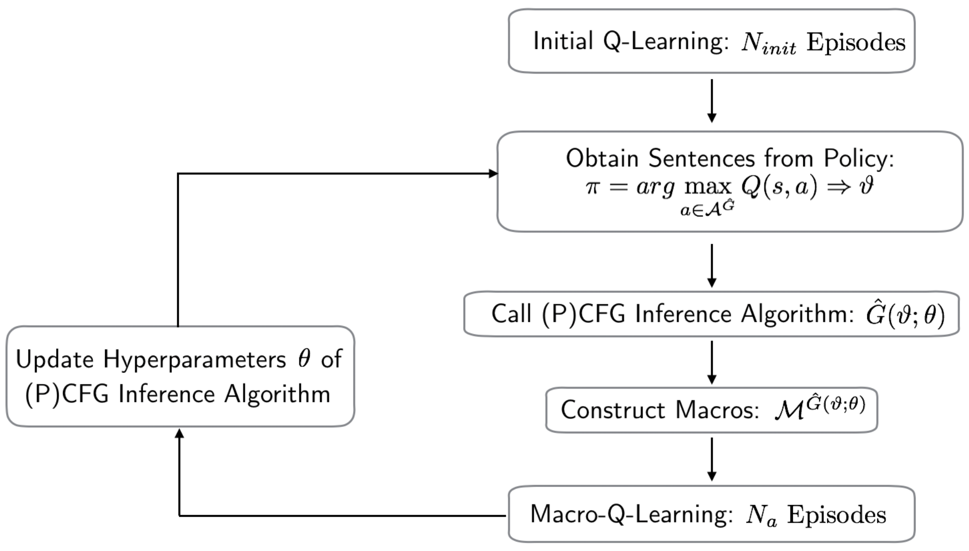

Initial Q-Learning: N_{init} Episodes
Obtain Sentences from Policy: $\pi = arg \max_{a \in \mathcal{A}^{\hat{G}}} Q(s,a) \Rightarrow \vartheta$
Call (P)CFG Inference Algorithm: $\hat{G}(\vartheta;\theta)$
Construct Macros: $\mathcal{M}^{\hat{G}(\vartheta;\theta)}$
Macro-Q-Learning: N_a Episodes
Update Hyperparameters θ of (P)CFG Inference Algorithm

Figure 4.5: Action Grammars for Online Macro-Action Discovery

The chosen hyperparameter updating-schedule has to sensibly balance the trade-off between a parsimonious amount of productions and not being to strict in the regularization. In the beginning of the learning process the agent will need many steps to reach the final goal state. Therefore, the trace on which we run the first Sequitur encoding will generate many productions (and therefore macros) if we do not regularize accordingly. On the other hand, as learning progresses the amount of steps needed decreases. Hence, we have to reduce the regularization parameter at a speed that is proportional to the overall learning progress of the agent. Decreasing or increasing the amount of regularization too fast might even slow down the learning process and introduce additional instability.
As a first rule of thumb it is plausible to treat the regularization parameter θ similar to a learning rule or an on-policy exploration parameter. E.g. for k-Sequitur we might start with a large k_{init} and decrease it over the course of the learning. Intuitively, the amount of noise in the sentence or policy reduces with more experience. Hence, overfitting becomes less of a problem. The overall online algorithm is described below:

Algorithm 2 Macro-Q-Learning with Online Updated CFG Production Rules

```
Input: Initial Q-Learning episodes, $N_{init}$. Number of episodes until update of the macro-
    action set, $N_a$. Initial CFG hyperparameters $\theta_{init}$. Updating schedule for $\theta$.
Output: Optimal value function and the corresponding policy
 1: for 1:$N_{init}$ Episodes do
 2:     Q-Learning Updating with $\mathcal{A}$ as the action space
 3: end for
 4: Generate a few traces from the Q-Learning policy. Denote the best trace by $\vartheta_{init}$.
 5: Call a CFG inference algorithm for $\vartheta_{init}$ and $\theta_{init}$ as inputs. Obtain the grammar
    $\hat{G}(\vartheta_{init}, \theta_{init})$. Extract production rules $\mathcal{M}^{\hat{G}(\vartheta_{init},\theta_{init})}$.
 6: Construct the Macro-action space $\mathcal{A}^{\hat{G}(\vartheta;\theta)} = \mathcal{A} \cup \mathcal{M}^{\hat{G}(\vartheta_{init},\theta_{init})}$
 7: repeat
 8:     for 1:$N_a$ Episodes do
 9:         Macro-Q-Learning Updating with $\mathcal{A}^{\hat{G}(\vartheta;\theta)}$ action space
10:     end for
11:     Update $\theta$ according to the schedule.
12:     Generate a new traces from current Macro-Q-Learning policy, $\vartheta$.
13:     Call a CFG inference algorithm for $\vartheta$ and $\theta$ as inputs. Obtain the grammar $\hat{G}(\vartheta, \theta)$.
14:     Extract production rules $\mathcal{M}^{\hat{G}(\vartheta,\theta)}$.
15: until Convergence
16: return $Q^\star(s,a)$ and $\pi^\star(s) = arg\max_a Q^\star(s,a)$.
```

The algorithm above describes the online version in which the agent refines his grammar estimate on the go. In the imitation learning scenario the agent observes an expert and obtains trajectories ϑ^{expert}. In this case she does not have to update the grammar estimate since the sentences are already sampled from $L(\pi^\star|E)$. Therefore, she does not need to switch back and forth but can simply use the originally extracted flat production rules throughout the entire learning process.

4.2.2 PAG: HMM Macro-Actions

Instead of treating subsequences extracted from a straight-line grammar as macro-actions, we can also sample them from a probabilistic model trained on the sequences ϑ^i.

During the grammar learning phase we train a Hidden Markov Model on the trajectories $\vartheta^1, \ldots, \vartheta^{N_g}$. The number of hidden states are treated as the hyperparameter θ. The HMM then specifies three discrete probability distributions: $p(z_1) \in \mathbb{R}^{\theta}$, $p(z_t|z_{t-1}) \in \mathbb{R}^{\theta \times \theta}$, and $p(a_t|z_t) \in \mathbb{R}^{|\Sigma| \times \theta}$.
In order to obtain a macro-action we first sample a hidden state z_1 according to $p(z_1)$ and a first action from $a_1 \sim p(a|z_1)$. Afterwards, we transition in the latent space from hidden state to the next according to a sample from $p(z_2|z_1)$. The next visible action can then again be sampled from $p(a_2|z_2)$ and so on. But how can we come up with an effective way to stop sampling additional actions? In the end we are interested in extracting useful macros which are not too long or too short. One way of defining an adaptive measure that interrupts the sampling process is surprisal. After the first action has been sampled we are easily able to obtain $p(a_2|a_1)$ by matrix multiplication and marginalization (see section 3.4). We can then compute the surprisal measure $-\log p(a_2|a_1)$ whether or not the next sample was unexpected. If the surprisal exceeds a threshold ξ (that may adapt with the length of the sampled action sequence and the number of grammar updates) we stop the sampling. If it does not we append a_2 to a_1 to form the first two-step macro a_1, a_2 and continue the sampling process.
But now this leaves open the question of how to define the threshold ξ. One possible approach is to define ξ as a fraction $\epsilon \in (0, 1)$ of the maximal entropy distribution. The intuition is that if the posterior is close to simply randomly selecting an action as the successor, we should stop the sampling. Furthermore, we can have a decreasing ϵ-schedule throughout learning. Similar to the k-updating schedule for Sequitur, this can be thought of as the grammar learning rate. The overall procedure is pictorially illustrated in figure 4.6

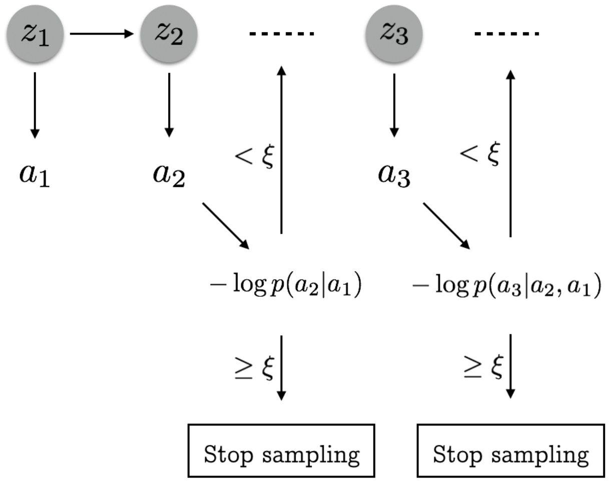


Figure 4.6: PAG: Macro-Action Sampling from a HMM

This framework is not just limited to HMMs but can be generalized to any probabilistic model which allows us to compute a posterior distribution of the form $p(a_t|a_{t-1}, \ldots, a_1)$. The specific steps for sampling individual actions might change but the overall procedure remains in tact.

4.2.3 Transferring Value Estimates between Grammar Updates

As we update the action space of the agent, the Q-table changes its shape. By introducing new temporally-extended actions, we extend and reduce the number of columns. So how can we maintain the knowledge that we have assembled between the updates. Our solution revolves around transferring the value estimates for the actions which persist and keeping a constant rate of exploration so that new macros are incorporated into the learning process.

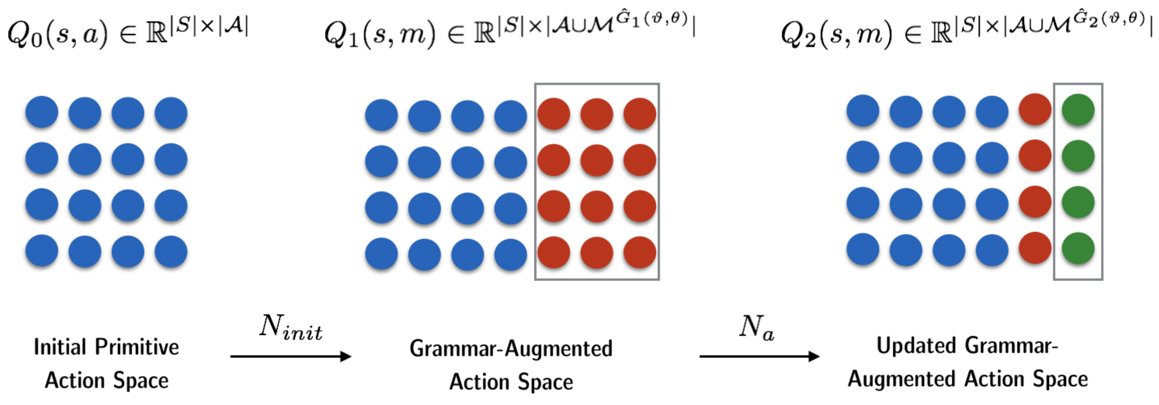

$Q_0(s,a) \in \mathbb{R}^{|S|\times|\mathcal{A}|}$
$Q_1(s,m) \in \mathbb{R}^{|S|\times|\mathcal{A}\cup\mathcal{M}^{\hat{G}_1(\vartheta,\theta)}|}$
$Q_2(s,m) \in \mathbb{R}^{|S|\times|\mathcal{A}\cup\mathcal{M}^{\hat{G}_2(\vartheta,\theta)}|}$
Initial Primitive Action Space
N_{init}
Grammar-Augmented Action Space
N_a
Updated Grammar-Augmented Action Space

Figure 4.7: Transferring Value Estimates between Grammar Updates

In figure 4.7 we show a simple example. The blue dots represent the primitive actions. As we infer the first grammar $\hat{G}_1(\vartheta,\theta)$, we obtain three new macros which are represented by the red columns. In our formulation we keep all value estimates for the primitive actions and initialize all new estimates to be zero. Hence, we have

$$Q_1(s,m) = \begin{cases} Q_0(s,a)\ \forall s \in S \text{ and } m \in \mathcal{A} \\ 0 \text{ otherwise.} \end{cases}$$

Afterwards, we simply continue with the action learning phase and update our value estimates for N_a episodes until we infer a new grammar $\hat{G}_2(\vartheta,\theta)$. If both grammars, $\hat{G}_1(\vartheta,\theta)$ and $\hat{G}_2(\vartheta,\theta)$ share a flattened production m, then again we are going to keep the value estimates and only initialize the macro-action column for the newly discovered macros (green column). Thereby, we have that

$$Q_2(s,m) = \begin{cases} Q_1(s,m)\ \forall s \in S \text{ and } m \in \mathcal{A} \\ Q_1(s,m)\ \forall s \in S \text{ and } m \in \mathcal{M}^{\hat{G}_1(\vartheta,\theta)} \cap \mathcal{M}^{\hat{G}_2(\vartheta,\theta)} \\ 0 \text{ otherwise.} \end{cases}$$

Since all new macro-action values are initialized to zero, the agent is likely to only use the macros in an exploration step. This can be a problem since the agent does not fully exploit the inferred grammar-based action space. Therefore, it is very important to either use a constant exploration rate or a decaying rate which is reset after a grammar update. In sections 5.1 and 5.3 we show results for the transfer and are able to show that this significantly reduces the learning variance.

Finally, we want to note that completely reinitializing the value estimates can also prove to be advantageous. If the agent is stuck in a local optimum, restarting the learning process with an altered action space can help to escape this local optimum due to the increased initial exploration.

4.3 Option Discovery

We hypothesize that a HRL agent is able to improve his learning performance by previously structuring sequences of actions similar to how humans comprehend language. In the previous sub-section we have seen how to do so in one specific HRL context, macro-actions. In this section we extend the Action Grammars framework to options in a similar vein.
The overall paradigm can be graphically expressed in the closed loop below (figure 4.8).

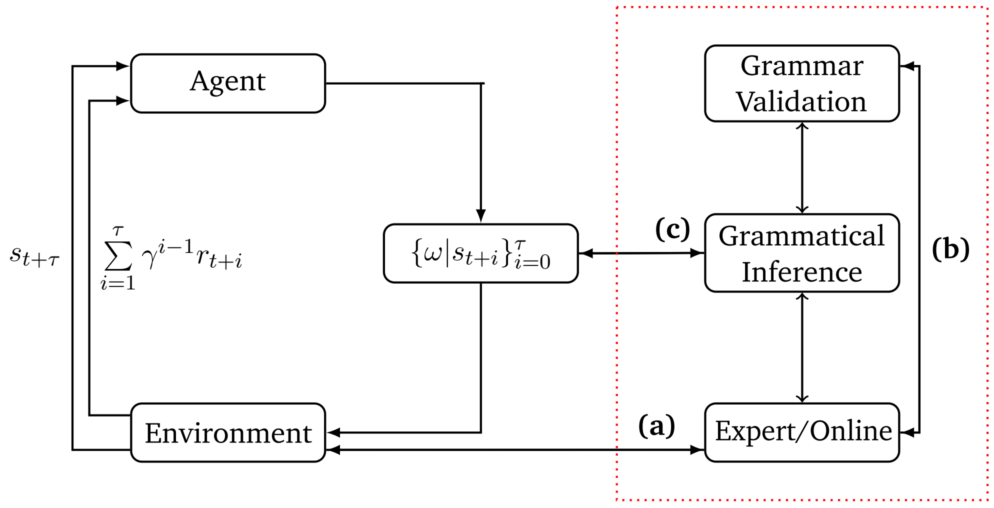


Figure 4.8: PCFAG: Action Grammars Closed-Loop. **(a)** Observation of expert/agent - $\{\vartheta\}_{i=1}^{N_g}$. **(b)** Fit grmmar $\hat{G}(\vartheta, \theta)$, via the grammatical inference algorithms of our choice. Choose best grammar structure by minimizing the cumulated syntactic surprisal value. **(c)** Construct option set, $\mathcal{O}^{\hat{G}}$ and learn action grammar values via SMDP-Q-Learning.

The agent first obtains state and action traces of an agent who follows a policy π. This agent can either be obtained in an on-policy fashion or is given by an expert. The traces can easily be simulated by following a the current or converged Q-Learning policy in a greedy fashion. We then have access to N_a sentences $\vartheta_1, \ldots, \vartheta_{N_a}$. Afterwards, the traces may be split in a set of trajectories used to fit the grammars of our choice and another set on which we compute a validation metric (see 4.4). We choose the "best" performing grammar and construct options in one of three ways:

1. **CFAG Options**: This option construction is based on concatenating trajectories into a single long string and encoding the action sequence using Lexis or k-Sequitur. Afterwards, one matches the termination of the flattened production rules with the state sequence. Thereby, one is able to construct an option set which reflects the frequency of state termination.

2. **Hidden Markov Model Options**: Based on observed trajectories we train a HMM. This HMM samples next actions based on previous transitions and thereby provides action recommendations. The option execution is stopped based on surprisal.

3. **Recurrent Neural Network Options**: Similar to a HMM option one is able to sample actions from the final Softmax layer of a RNN. This architecture is able to capture long-term dependencies whereas HMM rely on the Markov assumption.

The following sub-sections give details. Empirical test for the macro-action discovery problem can be found in section 5.2.

4.3.1 CFAG: Lexis and k-Sequitur Options

The first option construction that we present matches a CFG-encoded action sequence with the corresponding state sequence. Afterwards, one obtains a set of options based on the histogram of state-production rule terminations (for an overview see figure 4.9).
More specifically, we concatenate the actions in the trajectories $\{\vartheta^1, \dots, \vartheta^{N_g}\}$ into one long sequence denoted by $a|_1^{N_g}$ (**step 1**). Afterwards, we call a CFG inference algorithm with $a|_1^{N_g}$ as the input sequence and θ as the hyperparameters. We infer a grammar $\hat{G}(a|_1^{N_g}, \theta)$ which consists of the encoded sequence $\tilde{a}|_1^{N_g}$, the non-flattened production rules $\mathcal{P}$ as well as the recursively flattened production rules $\mathcal{M}$ (**step 2**).

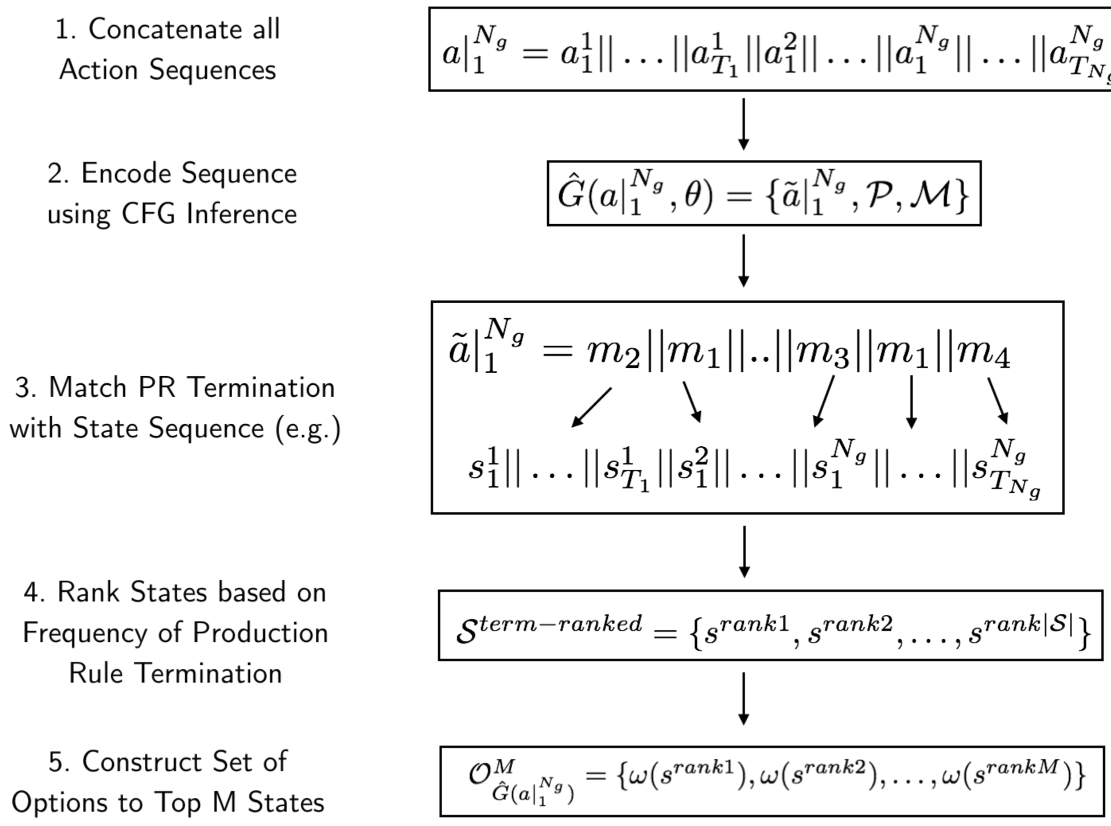

1. Concatenate all Action Sequences
$a|_1^{N_g} = a_1^1 || \dots || a_{T_1}^1 || a_1^2 || \dots || a_1^{N_g} || \dots || a_{T_{N_g}}^{N_g}$
2. Encode Sequence using CFG Inference
$\hat{G}(a|_1^{N_g}, \theta) = \{\tilde{a}|_1^{N_g}, \mathcal{P}, \mathcal{M}\}$
3. Match PR Termination with State Sequence (e.g.)
$\tilde{a}|_1^{N_g} = m_2 || m_1 || .. || m_3 || m_1 || m_4$
$s_1^1 || \dots || s_{T_1}^1 || s_1^2 || \dots || s_1^{N_g} || \dots || s_{T_{N_g}}^{N_g}$
4. Rank States based on Frequency of Production Rule Termination
$\mathcal{S}^{term-ranked} = \{s^{rank1}, s^{rank2}, \dots, s^{rank|\mathcal{S}|}\}$
5. Construct Set of Options to Top M States
$\mathcal{O}^M_{\hat{G}(a|_1^{N_g})} = \{\omega(s^{rank1}), \omega(s^{rank2}), \dots, \omega(s^{rankM})\}$

Figure 4.9: CFAG: Context-Free Grammar Option Construction

We can then obtain the states in which each $m \in \mathcal{M}$ starts and terminates by matching the encoded action sequence $\tilde{a}|_1^{N_g}$ with the state sequence $s|_1^{N_g}$ (**step 3**). A production rules will terminate in a set of specific states, while another production rule will terminate in another set. Hence, we can inspect the distribution of how frequently production rules terminate in specific states given the encoding (**step 4**).[2] Similarly to baseline approaches such as the option-critic framework (Bacon et al., 2017), the number of desired options M has to be chosen as a hyperparameter. We then obtain intra-option policies by simply backtracking macro executions which terminated in one of the states above. Thereby, we obtain an option set $\mathcal{O}^M_{\hat{G}(a|_1^{N_g})}$ (**step 5**).

[2]While doing so, we have to account for the initial concatenation and disregard all productions which are used at the crossing of two trajectories. These productions do not entail any semantic meaning.

A simplified and illustrative example of a CFG option construction based on the sequence "$drruurruuuu$" is given in figure 4.10:

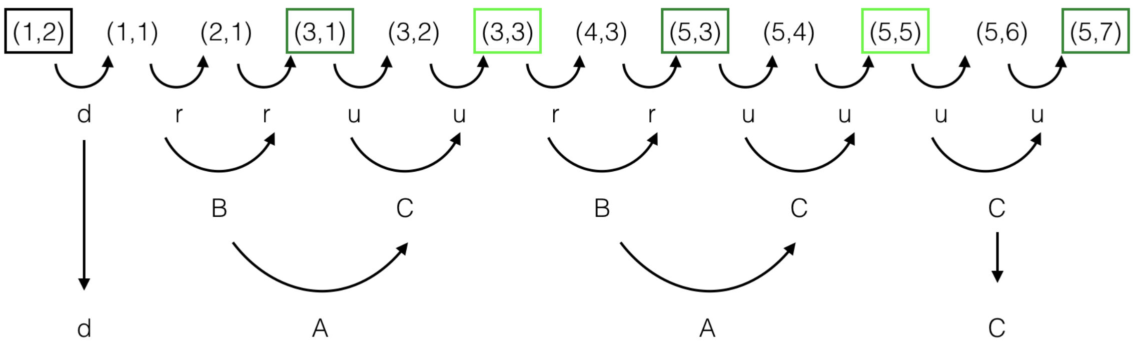


Figure 4.10: CFAG: Matching Context-Free Encoded Action Sequence and Concatenated State Sequence

The first row depicts an example of state transitions of an episode, the second row depicts the corresponding actions encoded as strings. Rows three and four display the production rules which are used to encode the original sequence into $\tilde{a}|_1^{N_g}$ = "$dAAC$". The table below illustrates the production rules and collects the states in which such start and terminate.

$\mathbb{N}$	$\mathcal{P}$	$\mathcal{M}$	Starting States	Termination States	Termination Rank
S	$dAAc$	$drruurruuuu$	$\{(1,2)\}$	$\{(5,7)\}$	1. $(3,3),(5,5),(5,7)$
A	BC	$rruu$	$\{(1,1),(3,3)\}$	$\{(3,3),(5,5)\}$	2. $(3,1),(5,3)$
B	rr	rr	$\{(1,1),(3,3)\}$	$\{(3,1),(5,3)\}$	
C	uu	uu	$\{(3,1),(5,3),(5,5)\}$	$\{(3,3),(5,5),(5,7)\}$	

Table 4.2: CFAG Option Construction Example.

The ranked ordering of the states is therefore given by:

$$\mathcal{S}^{term-ranked} = \{(3,3),(5,5),(5,7),(3,1),(5,3)\}$$

An option which terminates in state $(3,3)$ is then constructed in the following way:

$$\begin{aligned}
&\omega((3,3)) =< \mathcal{I}_{\omega((3,3))}, \pi_{\omega((3,3))}, \beta_{\omega((3,3))} > \\
&\mathcal{I}_{\omega((3,3))} = \{(1,1),(2,1),(3,1),(3,3)\} \\
&\pi_{\omega((3,3))}(r|(1,1)) = 1, \pi_{\omega((3,3))}(r|(2,1)) = 1 \\
&\pi_{\omega((3,3))}(u|(3,1)) = 1, \pi_{\omega((3,3))}(u|(3,2)) = 1 \\
&\beta_{\omega((3,3))}((3,3)) = 1
\end{aligned}$$

Similarly, we are able to construct a set of options for the top M most frequently terminated in states. For example for $M = 3$ we denote the set by:

$$\mathcal{O}^3_{\hat{G}(a|_1^{N_g})} = \{\omega_1((3,3)), \omega_2((5,5)), \omega_3((5,7))\}$$

As the to be encoded sequence $a|_1^{N_g}$ increases in length, the policies constructed by back-tracking the production rules become richer and richer. Furthermore, the states in which the productions terminate can be interpreted as subgoals and their frequency reflects a spatial-sequential importance. Algorithm 3 cleanly summarizes our procedure.

Algorithm 3 Option Construction with Context-Free Grammatical Inference

```
Input: Trajectories $\vartheta^1, \dots \vartheta^{N_g}$, a grammar induction algorithm with hyperparameter $\theta$, the
    desired number of options $M$.
Output: A set of $M$ CFG-inferred options.
 1: Concatenate all action sequences $a|_1^{N_g} = a_1^1 || \dots || a_{T_{N_g}}^{N_g}$.
 2: Call grammar induction algorithm with inputs $a|_1^{N_g}$ and $\theta$. Obtain production rules $\mathcal{P}$,
    flattened macros $\mathcal{M}$ and the encoded sequence $\tilde{a}|_1^{N_g}$.
 3: for each $m \in \mathcal{M}$ do
 4:     Look up the positions in which $m$ was produced in $a|_1^{N_g}$.
 5:     Look up corresponding start and termination states in $s|_1^{N_g}$.
 6:     Keep track of which production rules ended in which states.
 7: end for
 8: for $j = 1, \dots, M$ do
 9:     Select the $j$-th most terminated in state $\tilde{s}(j) \in \mathcal{S}^{term-ranked}$.
10:     Construct an option $\beta_{\omega_j(\tilde{s}(j))}(\tilde{s}(j)) = 1$ and for all other states to $0$.
11:     Rewind production rules from $\tilde{s}(j)$ that have led to it, add all states visited to $\mathcal{I}_{\omega_j(\tilde{s}(j))}$.
12:     Add the rewinded production rule to $\pi_{\omega_j(\tilde{s}(j))}$.
13:     Add the option $\omega_j(\tilde{s}(j)) =< \mathcal{I}_{\omega_j(\tilde{s}(j))}, \pi_{\omega_j(\tilde{s}(j))}, \beta_{\omega_j(\tilde{s}(j))} >$ to the option set.
14: end for
15: return $\mathcal{O}^M_{\hat{G}(a|_1^{N_g})} = \{\omega_j(\tilde{s}(j))\}_{j=1}^M$.
```

Algorithm 4 Option Action Grammars and SMDP Q-Learning

```
Input: Initial Q-Learning episodes, $N_{init}$. Number of episodes until update of the macro-
    action set, $N_a$. Initial CFG hyperparameters $\theta_{init}$. Updating schedule for $\theta$.
Output: Optimal state-option value function and a corresponding policy
 1: for 1:$N_{init}$ Episodes do
 2:     Q-Learning Updating with $\mathcal{A}$ as the action space
 3: end for
 4: Generate a few traces from the Q-Learning policy: $\vartheta^1, \vartheta^2, \dots, \vartheta^{N_g}$.
 5: Call Algorithm 3 and obtain $\mathcal{O}^M_{\hat{G}(a|_1^{N_g})} = \{\omega_j(\tilde{s}(j))\}_{j=1}^M$.
 6: repeat
 7:     for 1:$N_a$ Episodes do
 8:         SMDP Q-learning Update with $\mathcal{O}^M_{\hat{G}(a|_1^{N_g})}$ as the action space
 9:     end for
10:     Update $\theta$ according to the schedule.
11:     Generate new traces from current option-to-option policy, $\vartheta^1, \vartheta^2, \dots, \vartheta^{N_g}$.
12:     Call Algorithm 3 and obtain a new option set $\mathcal{O}^M_{\hat{G}(a|_1^{N_g})} = \{\omega_j(\tilde{s}(j))\}_{j=1}^M$.
13: until Convergence
14: return $Q^\star_{\mathcal{O}_{\hat{G}(\vartheta,\theta)}}(s,\omega)$ and $\pi^\star_{\mathcal{O}_{\hat{G}(\vartheta,\theta)}}(s,\omega) = arg\max_{\omega'} Q^\star_{\mathcal{O}_{\hat{G}(\vartheta,\theta)}}(s,\omega')$.
```

Again, we are able to construct and exploit such options for two learning cases: the imitation learning and the online reinforcement learning case. Algorithm 4 provides the procedure for the online case in which the agent updates his set of options after a specific number of iterations.
Finally, we need to make sure that the union of all initiation sets actually captures the whole state space ($A_s \subseteq \cup_{o \in \mathcal{O}} \mathcal{I}_o$). Otherwise the agent will not be able learn in states which are not captured by the option set. By adding a one-step primitive action option this can be easily achieved.
The approach described above provides only one possible way of constructing options with the help of context-free grammar inference. In practice we will see (see section 5.2) that it requires a lot of successful trajectories in order to infer meaningful options. Furthermore, it performs a lot better in the imitation learning setup than in the online learning case.
Therefore, we will now turn to alternative option construction mechanisms based on probabilistic grammars.

4.3.2 PAG: HMM and RNN Options

Similarly to the way how we sampled macro-actions in section 4.2 we are also able to construct options from stochastic grammars. In this section we derive two ways to construct semi-Markov options. Both rely on surprisal-based option termination and the intra-option policy execution is history dependent.
Compared to the previous section in which we learned a grammar on the action sequence, we now learn a grammar on the state sequences. Furthermore, since we no longer infer a straight-line grammar, we do not need to concatenate the sequences.
Given that we have learned a probabilistic grammar on the state sequences $s^1, \dots s^{N_g}$, a PAG option execution works as follows: Given state s_1 the agent makes a decision between a primitive action (one-step option) and a PAG option based on the value estimates $Q(s, \omega)$. If he chooses to execute the option, an action is sampled from the inferred transition distribution $p(s'|s_1)$.[3] Afterwards, the agent executes this action and transitions to the next state s_2.
We note that the actual state transition not necessarily has to be the planned one. This is due to potential stochasticity in the environment. After the actual transition has occurred, we evaluate the surprisal based on the history of the option execution, $-\log p(s_2|s_1)$. Again, if the surprisal measure exceeds a threshold ξ we stop the option execution and return control to the agent who then has to make a decision about his next action execution. Otherwise, we extend the option-execution-specific history h and continue to follow the state transitions sampled or recommended by the PAG option. After the next transition we repeat the surprisal-based termination check. Next we derive a formal definition of the semi-Markov option construction and give to examples.

Hidden Markov Model Option

The general idea of our PAG approach to option discovery is to endow the HRL agent with the option to follow "action recommendations" of a pre-trained PAG. More specifically, we train the HMM with hidden states z on state transition traces. At time step t in one episode of the RL learning the agent has then access to the history-dependent state transition distribution of the HMM. It can easily be computed by marginalizing over the latent variable at time $t+1$:

[3]Notice that one is also able to take the $arg \max_{s'} p(s'|s_1)$. This can be interpreted as an analog to greedy or Softmax on-policy execution. Furthermore, sampling incentivizes additional exploration.

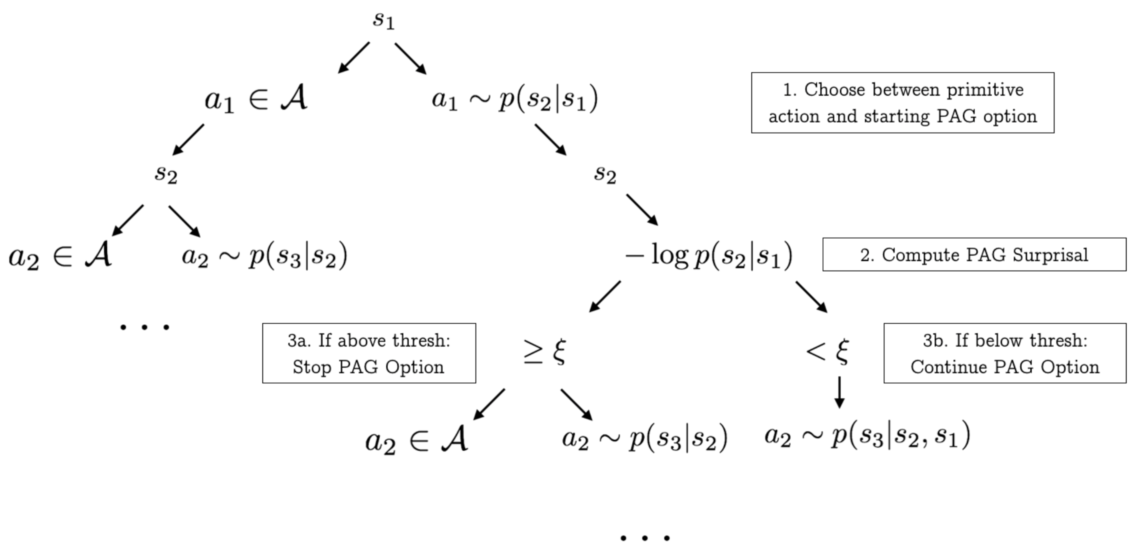


Figure 4.11: PAG: Probabilistic Grammar Option Execution

$$p(s_{t+1}|s_1, \dots s_t) = \sum_{z_{t+1}} \underbrace{p(s_{t+1}|z_{t+1})}_{\text{Emissions}} \underbrace{p(z_{t+1}|z_t)}_{\text{Transitions}} \underbrace{p(z_t|s_1, \dots, s_t)}_{\text{Filtering}}$$

The action and state transition which the agent tries to achieve is then given by one of two operations:

$$s_{t+1} = arg \max_s p(s|s_t, \dots, s_{t-\tau}) \textbf{ (Greedy)}$$
$$s_{t+1} \sim p(s|s_t, \dots, s_{t-\tau}) \textbf{ (Sampling)}$$

The agent can either choose to maximize the posterior probability in a greedy fashion or he can sample the next state. The executed primitive action is then given by the one that would achieve this transition. More specifically, we define a PAG-based option as follows:

- $\mathcal{I}_\omega = \mathcal{S}, \ \forall \omega \in \mathcal{O}$: We assume that we are able to initiate the PCFAG option in every state of the state space. This is a common assumption in the option discovery literature. We note that this is a pragmatic assumption that simplifies the option construction but limits the degree of specialization and interpretability.
- $\pi_\omega(a_t|h_t) \Leftrightarrow p(s_{t+1}|h_t)$ where $h_t = \{s_t, \dots, s_{t-\tau}\}$: The intra-option policy is given by the action selection which is consistent with the state transition distribution inferred by the HMM.
- $\beta(s_{t+1}|h_{t+1}) \approx \mathbf{1}_{\{-\log p(s_{t+1}|h_{t+1}) > \xi\}}$: The option termination is based on the surprisal induced by the state transition. If $-\log p(s_{t+1}|h_{t+1})$ exceeds the threshold ξ the option execution stops. Otherwise, it continues.

In practice (see section 5.2), we usually set ξ to be the 90th percentile of an empirical surprisal distribution computed on a hold-out set of trajectories. Furthermore, we also found it to be useful to set a maximal number of state transitions within an option execution (e.g. a maximum of 10 primitive action executions).

Recurrent Neural Network Option

Recurrent Neural Networks have successfully been applied to natural language processing problems for a long time. A RNN compared to a simple Multilayer Perceptron (MLP) has recurrent connections which allow it to capture long-term dependencies. We experiment with such an alternative construction since HMMs are limited by the Markov assumption. The hidden state transition at time t only depends on the hidden state at time $t-1$. RNNs, on the other hand, allow for variable time dependencies which are learned in an end-to-end fashion via backpropagation through time. Hence, HMMs have the advantage of being more parsimonious while only being able to capture a limited amount of dependency.

So what does this mean in the context of Hierarchical Reinforcement Learning? An agent who either observes an expert or has already developed substantial skills herself will have meaningful trajectories at his disposal. These trajectories exhibit long-term dependencies so that RNNs are well suited to model the observed dynamics and to generate action proposals in an option setting. An agent, on the other hand, who is in the early stages of the learning process has probably only mastered small sub-skills. In this case it is more effective to use a more parsimonious model such as a HMM. This is related to the observation by Smith et al. (2018) that number of necessary options might vary throughout the learning process. Similarly, the degree of grammatical complexity is sensitive to the learning stage.

Similar to the previous HMM option we can have a RNN option which recommends state transitions given the intra-option history h_t. E.g. given that a RNN option execution started in $t-\tau$, the history consists of the state sequence $h_t = \{s_{t-\tau}, \dots, s_{t-1}, s_t\}$. At time t we are then able to encode these states and to feed them into a RNN which was previously trained on state sequences given from $\vartheta^1, \dots, \vartheta^{N_g}$. For example we might encode the states using one-hot encoding or by simply giving them a symbolic representation. The final layer of the RNN then consists of a Softmax layer which returns a vector $y \in \mathbb{R}^{|S|}$. We treat this vector as the posterior $p(s_{t+1}|s_t, \dots, s_{t-\tau})$, hence y_1 characterizes the posterior probability of transitioning to the first state (in terms of our encoding) after having traversed through the state history $h_{t-\tau}$. As before, we can again either take a greedy argmax operation or sample from this distribution to obtain the next state.[4] Furthermore, we can easily compute the syntactic surprisal metric as $-\log p(s_{t+1}|s_t, \dots, s_{t-\tau})$ and terminate the RNN option execution accordingly.

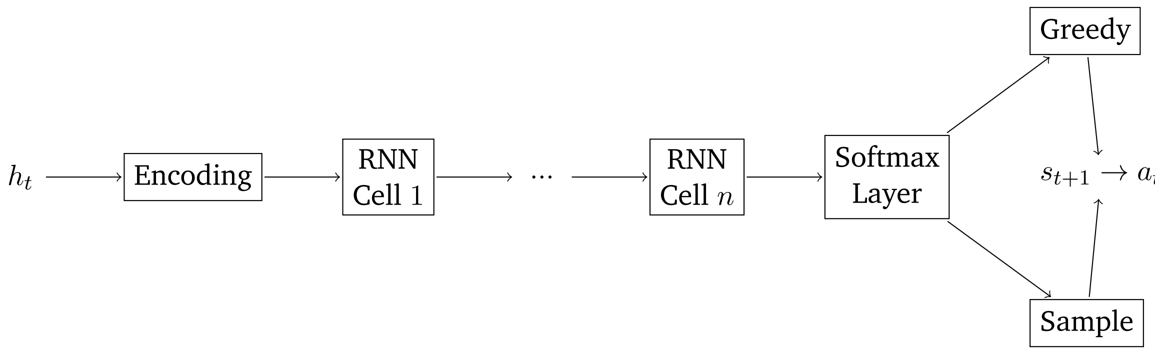


Figure 4.12: PAG: Recurrent Neural Network Architecture

[4]In practice not every state transition is possible due to the primitive actions. Therefore, we drop all entries of the vector which would lead to an invalid transition and choose the state according to this smaller normalized vector.

In section 5.2 we will experiment with different neural network architectures: Long Short-Term Memory networks (Hochreiter and Schmidhuber, 1997) and Gated Recurrent Unit networks (Chung et al., 2015). They are characterized by specific cells.
LSTM cells consist of parametrized gates which allow information to smoothly be "forgotten" and "remembered". During learning the parameters of such are optimized using mini-batch gradient descent to optimize some performance metric (e.g. accuracy/cross-entropy). Gated Recurrent Units (GRUs), on the other hand, are well known to be better suited for cases where data is scarce. This is mainly due to the fact that they do not have an output gate and therefore less parameters need to be trained. GRUs therefore might provide a good intermediate architecture between the complex LSTM and the parsimonious HMM.

Notes on Surprisal-Based Option Termination

Our approach terminates an option given that the syntactic surprisal exceeds a threshold ξ. But what does this mean in the context of an agent who moves through an environment. We distinguish between two types of surprisal. First, an agent can be surprised by an unsuccessful state transition. We refer to this surprisal as being qualitatively "negative". The option does not execute actions as planned. It seems natural to return to the agent so that she can make the next decision. Hence, negative surprisal signals that the agent needs to contemplate about the current situation. In some cases an unsuccessful state transition can actually lead to an improvement. This is especially the case for parts of the state space which have not been part of the trajectories used to train the PAG. Unexpected transitions can therefore also lead to "positive" surprisal which is related to exploration. Empirically, interruption in such situation does not pose a problem (especially in sparse environments). In both cases returning control to the agent can be interpreted as a form of conservative safety insurance. Instead of blindly following the actions recommended by the option, the agent has to make a "conscious" (slow) decision.
We are going to use the notion of surprisal not only in the context of option termination but also to compare different grammatical structures and to choose the grammar that is actually used within the action learning phase.

4.4 The Grammar Comparison Problem

Ultimately, we are interested in inferring the grammar that consists of the best hierarchical structures which can afterwards be used to construct a set of options. So how can we make use of a clever parallelism to supervised learning? The answer is simple: An agent who has learned a good grammatical structure of the hierarchical behavior of an agent will not be surprised by parsing further action traces of this expert. Hence, we propose to treat the grammar selection problem as a special version of the model comparison problem in supervised learning.

We are able to compare different grammars by splitting the the overall set of trajectories into a training set and a test set:

$$\{\vartheta^1, \dots \vartheta^{N_g}\} = \vartheta_{test} \cup \vartheta_{train} = \{\vartheta^1_{test}, \dots, \vartheta^{N_{test}}_{test}\} \cup \{\vartheta^1_{train}, \dots, \vartheta^{N_{train}}_{train}\}$$

Afterwards, we learn a grammar using ϑ_{train} and evaluate the goodness-of-fit measure of our choice on the test set. For example, we can optimize the accumulated surprisal on the hold-out set of trajectories. We call this measure *replay surprisal*:

$$RS^{\hat{G}}(\vartheta_{test}) = \frac{1}{N_{test}} \sum_{i=1}^{N_{test}} \frac{1}{T_i} \sum_{t=1}^{T_i - 1} - \log P_t^{\hat{G}(\vartheta_{train}, \theta)}(s_{t+1}^{i;test}) \geq 0$$

While parsing through each sequence in the test set, we accumulate surprisal according to how well the inferred grammar captures the underlying hierarchical structure.

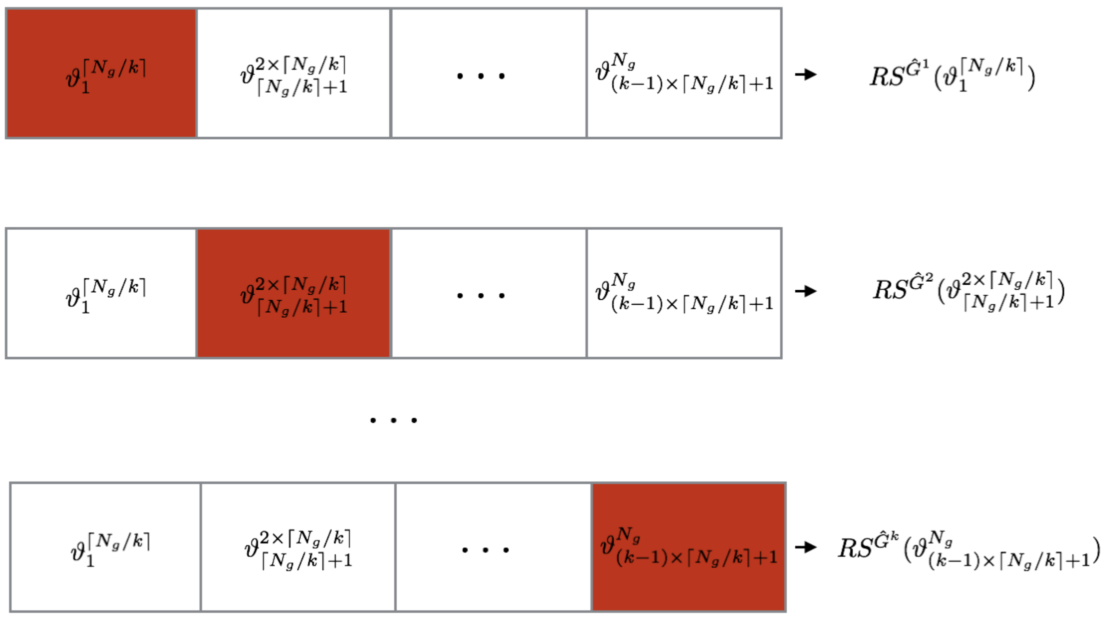

$\vartheta_1^{\lceil N_g/k \rceil}$
$\vartheta_{\lceil N_g/k \rceil+1}^{2\times\lceil N_g/k \rceil}$
$\cdots$
$\vartheta_{(k-1)\times\lceil N_g/k \rceil+1}^{N_g}$
$RS^{\hat{G}^1}(\vartheta_1^{\lceil N_g/k \rceil})$
$RS^{\hat{G}^2}(\vartheta_{\lceil N_g/k \rceil+1}^{2\times\lceil N_g/k \rceil})$
$RS^{\hat{G}^k}(\vartheta_{(k-1)\times\lceil N_g/k \rceil+1}^{N_g})$

Figure 4.13: PAG: Cross-Validation Replay Surprisal

Similar to supervised learning, the notion of overfitting also applies to the domain of learning hierarchies of sequential actions. By overfitting the noise observed in the training trajectories, we might be unable to capture the true hierarchical process generating the transitions at hand. We face a similar bias-variance trade-off and have to find the fine balance between fitting the data-generating process (the underlying hierarchical structure) and not overfitting noise in form of non-hierarchical structure and possibly unsuccessful transitions.

We can further improve our estimate of the generalization error by generalizing the train-test split with k-fold cross validation (see e.g. Friedman et al. (2001, p. 241f.)). Therefore, we split the traces into k folds, fit a grammar on $k-1$ blocks and compute the replay surprisal for the k-th block. We do so for every split and finally average the computed metrics.

$$\overline{RS}^{\hat{G}} = \frac{1}{k} \sum_{j=1}^{k} RS^{\hat{G}^j}(\vartheta_{(j-1)\times\lceil N_g/k \rceil+1}^{j\times\lceil N_g/k \rceil}) = \frac{1}{k} \sum_{j=1}^{k} RS^{\hat{G}^j}(\vartheta_{(j-1)\times\lceil N_g/k \rceil+1}, \dots, \vartheta_{j\times\lceil N_g/k \rceil})$$

The 'best' grammar is then obtained by minimizing this estimate of the grammar generalization error. Algorithm 5 summarizes the cross-validation estimate of the surprisal generalization error.

Algorithm 5 Grammar Comparison via Replay Surprisal

```
Input: An action sentence $\mathcal{A}$, a set of grammar induction algorithms $\mathcal{G}$, and a set of $K$ trace splits.
Output: Hierarchical grammar structure minimizing the cross-validation error.
1: for each grammatical inference algorithm do
2:     for $j = 1, \dots, k$ do
3:         Use training set of split $j$ to construct the grammar $\hat{G}^j$
4:         Calculate $RS^{\hat{G}}(\vartheta_{test}) = \frac{1}{N_{test}} \sum_{i=1}^{N_{test}} \frac{1}{T_i} \sum_{t=1}^{T_i - 1} - \log P_t^{\hat{G}(\vartheta_{train}, \theta)}(s_{t+1}^{i;test})$
5:     end for
6:     Average the replay surprisal: $\overline{RS}^{\hat{G}} = \frac{1}{k} \sum_{j=1}^{k} RS^{\hat{G}^j}(\vartheta_{(j-1)\times\lceil N_g/k \rceil + 1}^{j \times \lceil N_g/k \rceil})$
7: end for
8: Obtain the best grammar as $G_K^\star = arg \min_{\hat{G} \in \mathcal{G}} \overline{RS}^G$.
9: return $G^\star$ the best grammar after relearning on all $N$ expert traces.
```

Note that any other model comparison/hyperparameter optimization algorithm which uses replay surprisal as the loss metric can be used to compare grammars.
This chapter has introduced a first formal framework which connects Hierarchical Reinforcement Learning with Computational Linguistics. First, we have shown how one is able to efficiently obtain semantically meaningful macro-actions by either making use of CFG or PCFG inference. On the one hand, we showed how to use a CFG encoding such as Sequitur and the inferred production rules to construct macros both online and in an imitation learning setting. On the other hand, we derived a procedure which samples macro-actions based on a trajectory-trained HMM.
Second, we illustrated two algorithms which construct options based on hierarchical structures inferred by grammatical inference. By matching production rules obtained from concatenated action sequences with the corresponding state sequence, one can obtain options which are based on the frequency of rule termination. Furthermore, we showed how to construct options which sample action recommendations based on state transitions in an online fashion.

The next chapter will illustrate how such approaches can be used to solve HRL problems in environments with severe sparse rewards and long-term credit assignment difficulties. We will show that grammatical inferred macro-actions as well as options capture valuable information to solve such difficult problems at an efficient pace.

Chapter 5

Experiments and Results

After having introduced the action grammars framework which attempts to solve the problem of hierarchical sub-structure discovery in the context of HRL, we are now ready to display empirical results of our approaches on several environments. We distinguish between two different tasks:

1. **Imitation Action Grammar Learning Task**: Before the learning starts an agent observes a set of expert traces and has to discover temporally-extended actions. These actions are then used to learn and generalize to new trajectories.

2. **Online Action Grammar Learning Task**: The agent is not endowed with any prior knowledge but alternates between grammar learning on his own traces and action learning. Solving this task is the ultimate aim of our endeavors.

Thereby, we discover the following dynamics:

- Towers of Hanoi (**deterministic**): Macro-actions extracted with k-Sequitur or Lexis (see section 4.2) are able to effectively solve the curse of dimensionality. Furthermore, grammar macros allow us to propagate value information efficiently and to thereby solve the long-term credit assignment problem. As learning progresses the extracted macros converge to the macros extracted from the optimal policy. In special situations one has to deal with local optima which can only be escaped with additional exploration incentives. The updating schedule of the grammar inference hyperparameter θ plays an important role in stabilizing learning. The described approaches provide strong learning results in both imitation learning and online reinforcement learning setups.

- Four Room Problem (**stochastic**): As state transitions become stochastic, macro-actions become inefficient. We saw that the hallway options provide the agent with strong structural information but have to be manually supplied. Automatically inferred grammar-based options provide an alternative (see section 4.3). We show that both CFG-based as well as PCFG-based options are able to automatically construct options which perform as well as the hallway options. Furthermore, we show that they are also useful in the online learning case. While improving the learning speed of the HRL agent, they introduce additional unwanted variance. GRU options seem to perform best in small sample size domains.

- OpenAI Environments: We test grammar induction-based macro-action and option discovery in several OpenAI gym (Brockman et al., 2016) environments. We find that the success of our approach heavily depends on the hierarchical nature of the environment. If the solution requires the hierarchical achievement of subgoals and is deterministic, grammar-based macro-actions can improve learning. If the environment is not hierarchically structured our approach fails.

5.1 Towers of Hanoi

Towers of Hanoi is a very simple game (see figure 5.1 for an example of the optimal three-disk policy). The general game setting for N disks is as follows: In every episode of the game the agent is initialized in the tuple $(1)_{i=1}^{N}$. The elements of the tuple represent the individual disks. The position of the smallest disk corresponds to the leftmost entry of the tuple while the largest disk is given by the rightmost. Hence, from left to right (in the tuple) the size of the disks increases. This is important since the environment does not allow a bigger disk to be placed on a smaller one. Knowing this, the N-dimensional tuple uniquely identifies the state of the environment. A few examples for $N = 3$ to make this clear:

- $(1, 1, 1)$: Initial state where all 3 three disks are stacked on pole 1
- $(3, 3, 3)$: Final state where all 3 three disks are stacked on pole 3
- $(1, 2, 3)$: Smallest disk on pole 1 middle disk on pole 2, largest on pole 3

At each point in (discrete) time the agent transitions between states with the help of the following moves:

- $(1, 2)$: Move top disk of pole 1 to top of pole 2 abbreviated by string "a"
- $(1, 3)$: Move top disk of pole 1 to top of pole 3 abbreviated by string "b"
- $(2, 1)$: Move top disk of pole 2 to top of pole 1 abbreviated by string "c"
- $(2, 3)$: Move top disk of pole 2 to top of pole 3 abbreviated by string "d"
- $(3, 1)$: Move top disk of pole 3 to top of pole 1 abbreviated by string "e"
- $(3, 2)$: Move top disk of pole 3 to top of pole 2 abbreviated by string "f"

The three disk environment and the corresponding optimal policy is given in figure 5.1.[1] The agent maximizes his expected cumulative discounted reward by reaching the final state $(3)_{i=1}^{N}$ as quickly as possible. For every transition that does not lead to the goal state, the agent receives 0 reward. If he reaches the final state he obtains a reward of 100. He discounts his reward by $\gamma \in (0, 1]$. Hence, we are dealing with an environment with a severe long-term credit assignment problematic. The size of the action is independent of the number of disks, $|A| = |\{a, b, c, d, e, f\}| = 6$. Furthermore, the action space conditioned on the state s has size three, $|A_s| = 3,\ \forall s \in S \setminus (1)_{i=1}^{N}$. Disregarding the reverse move of the previously executed one, the effective action space in every state has size 2. The size of the state space, on the other hand, grows exponentially, $|S| = 3^N$ (all possible allowed orderings), and the optimal number of moves to solve this game is given by $2^N - 1$.

[1]Accessed and altered on 20-08-2018 from `https://www.includehelp.com/data-structure-tutorial/tower-of-hanoi-using-recursion.aspx`.

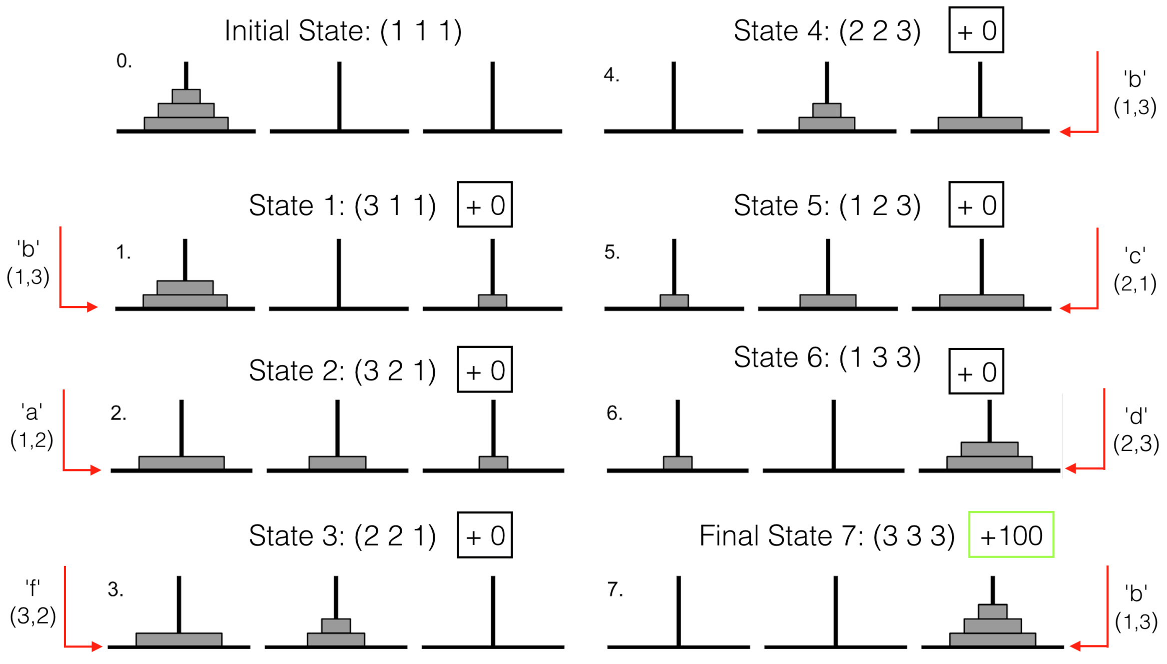


Figure 5.1: Towers of Hanoi: Three Disk Optimal Policy

A simple recursive procedure to solve the problem for all states in which the top $N - n$ disks are already correctly ordered on the third pole is given by (see e.g. Petkovi (2009):

- Move $n - 1$ disks from source pole to auxiliary pole
- Move the n-th disk from source pole to target pole
- Move the $n - 1$ disks that we left on auxiliary pole onto target pole

Furthermore, it is well known that this problem can be represented as a undirected graph which constitutes a Sierpinski triangle. A Sierpinski triangle is a fractal figure which can be constructed by drawing a triangle and the connecting all mid-points of the edges with each other. The number of triangles K stands in the following relation to the number of iterations/disks: $K = 3^{N-1}$. The optimal policy of the RL agent is then given by a traversal along an outer edge of overall the triangle.
As the number of disks grows, the problem quickly becomes a sparse long-term credit assignment problem which is hard to solve with vanilla-Q-learning updates of the action values. For $N = 6$ the state space has a size of 729 and the optimal deterministic policy has 63 moves. Due to the fact that the agent only observes a non-zero reward once he reaches the final state, learning is almost impossible. She essentially gets lost in the search tree and desperately searches for some form of reward signal that would structure her understanding of the environment. Instead of having to make a decision over single action steps, the agent wants to combat the initial sequential decision making uncertainty by only retaining control at key points in time. We will now see how such points can be obtained using grammatical inference as described in section 4.2.
In all experiments outlined below and if not otherwise stated, we choose the following hyperparameters:

Towers of Hanoi Hyperparameters.

1. Exploration parameter: $\epsilon = 0.1$
2. Discount factor: $\gamma = 0.8$
3. Learning rate: $\alpha = 1$
4. Eligibility parameter: $\lambda \in \{0, 0.1\}$
5. Number of PCFAG Macros used in learning: $M \in 5, 10$
6. Number of traces on which we infer a HMM-based grammar: $N_g \in \{100, 500\}$
7. Number of initial Q-Learning iterations: $N_{init} = 200$
8. Number of episodes between grammar updates: $N_a = 312$
9. Number of grammar updates: 9 (at episodes $\{200, 624, 936, 1248, 1560, 1872, 2184, 2496, 2808\}$)
10. k-Sequitur hyperparameter: $\theta = k_{init} \in \{4, 6, 8\}$
11. HMM hyperparameter: $\theta = 4$ (Hidden States)
12. Updating schedule for k-Sequitur hyperparameter: $k_{next} = k_{prev} - 1$ if $k_{prev} > 2$
13. Maximal surprisal (macro-sampling termination criterion): $\xi = 0.9 \times -\mathbb{E}[\log_2 p(a)]$ where $p(a) = \frac{1}{|\mathcal{A}|} = \frac{1}{6}$

5.1.1 Macro-Action Discovery with (Probabilistic) Context-Free Action Grammars

Besides the vanilla-Q-learning agent, we provide 2 different approaches which differ significantly in terms of the amount of prior knowledge with which we endow the HRL agent:

1. **Imitation Learning with Macros extracted from optimal N or $(N-1)$-disk policy**: We extract production rules from the string which represents the optimal policy. Afterwards, the action space of the agent is augmented with macros that correspond to the flattened production rules. The agent then learns to compose these temporally-extended actions.

2. **Online Reinforcement Learning with iteratively updated (P)CFAG Macros**: We update the set of macro-actions iteratively and in an online fashion. First, the agent runs simple Q-learning updates for a warm-up period. Afterwards, we rollout a few game play traces and construct a first set of macro-actions. Then we start Macro-Q-learning with this fixed set of macro-actions. After a certain number of updates, we again run a few rollouts and update the set of macro-actions.

The optimal deterministic policies for all three considered environments (4, 5, 6 and 7 disks) are shown in the first row of figure 5.1. Furthermore, rows two and three show the macro-actions or flattened productions that one can derive by obtaining a 2-Sequitur or Lexis encoding of the optimal policy. Ultimately, we are not only interested in obtaining the optimal policy but also in inferring the optimal set of productions which constitute the action space of the HRL agent. Hence, another form of intelligence is captured by reflecting on which sub-sequences of actions lead to successful experiences. Exactly this form of syntactic intelligence is identified by grammar learning.

Disks	4	5	6	7
$\pi^\star$	$abdaefabdcedabd$	$bafbcdbafecfbafb$ $cdbcfecdbafbcdb$	$abdaefabdcedabda$ $efaedcefabdaefab$ $dcedabdcefaedc$ $edabdaefabdcedabd$	$bafbcdbafecfbafbcd$ $bcfecdbafbcdbafecf$ $bafecdbcfecfbafbc$ $dbafecfbafbcdbcfe$ $cdbafbcdbcfecfbafec$ $dbcfecdbafbcdbafecf$ $bafbcdbcfecdbafbcdb$
2-Seq.	$\{abd\}$	$\{bafbcd, baf, ec, bc\}$	$\{abdaefabdced\}$ $\{abdaef, aedce\}$ $\{abdce, abd, ae, ce\}$	$\{bafbcdbafecfb$ $afbcdbcfecd\}$ $\{bafbcdbafecf\}$ $\{bafecdbcfec\}$ $\{bafbcdbcfec\}$ $\{bafbcd, bafec\}$ $\{bcfec, baf, bc, ec\}$
G-Lexis	$\{abd\}$	$\{bafbcdb\}$	$\{abd, efaedce\}$ $\{abdaefabdcedabd\}$	$\{bafbcdbafecfbaf$ $bcdbcfecdbafbcdb\}$ $\{fec, bafbcdb\}$ $\{fecfbafecdbcfec\}$

Table 5.1: Towers of Hanoi: Optimal Policies and Extracted Macro-Actions

Figure 5.2 shows how different CFG inference algorithms encode the optimal policies for four to eight disks. We can observe that the compression rate of the 4-Sequitur encoding remains constant, while Lexis and 2-Sequitur infer more and more productions as the disk number and optimal policy length increases. This observation is very alarming since the data-generating process (e.g. the recursive solution reviewed above) remains exactly the same for all N disk environments. Hence, only 4-Sequitur seems to be robust to the length of the policy. Consequently, 4-Sequitur also does the worst job in structuring the original sequence. Comparing both the entropy of the unencoded optimal policy and post-CFG inference encoded string, 4-Sequitur only induces more structure (less post-compression entropy/point below the diagonal line) for the 4 and 5 disk environment. 2-Sequitur and Lexis on the other hand, seem to overfit noise and thereby do a better job at compression as well as decreasing entropy.

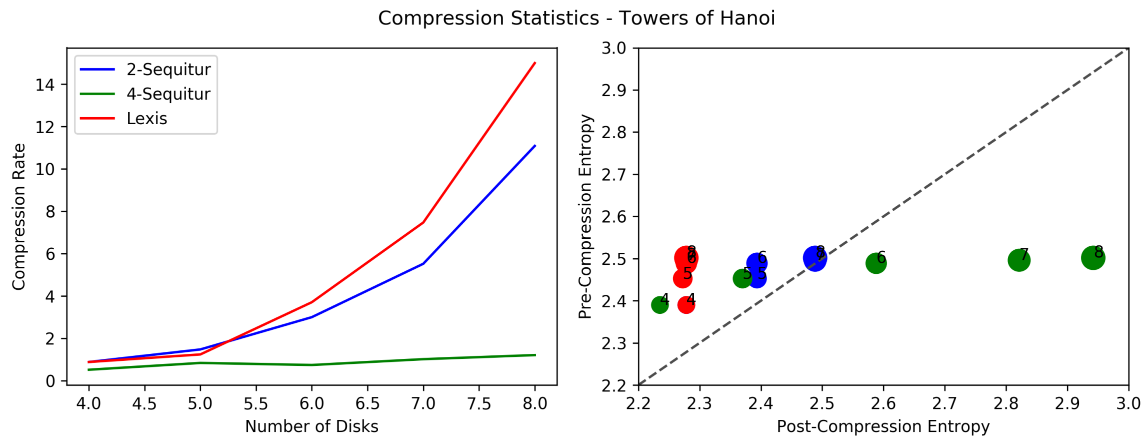


Figure 5.2: Towers of Hanoi: Compression Statistics of CFG Encoding

Figure 5.3, on the other hand, shows how severe the long-term credit assignment problem is. The simple Q-learner with either ϵ-greedy or Softmax exploration learns very slowly. For 4, 5, 6 and 7 disks convergence occurs after around 3000, 20000, 125000 and 8000000 episodes, respectively.

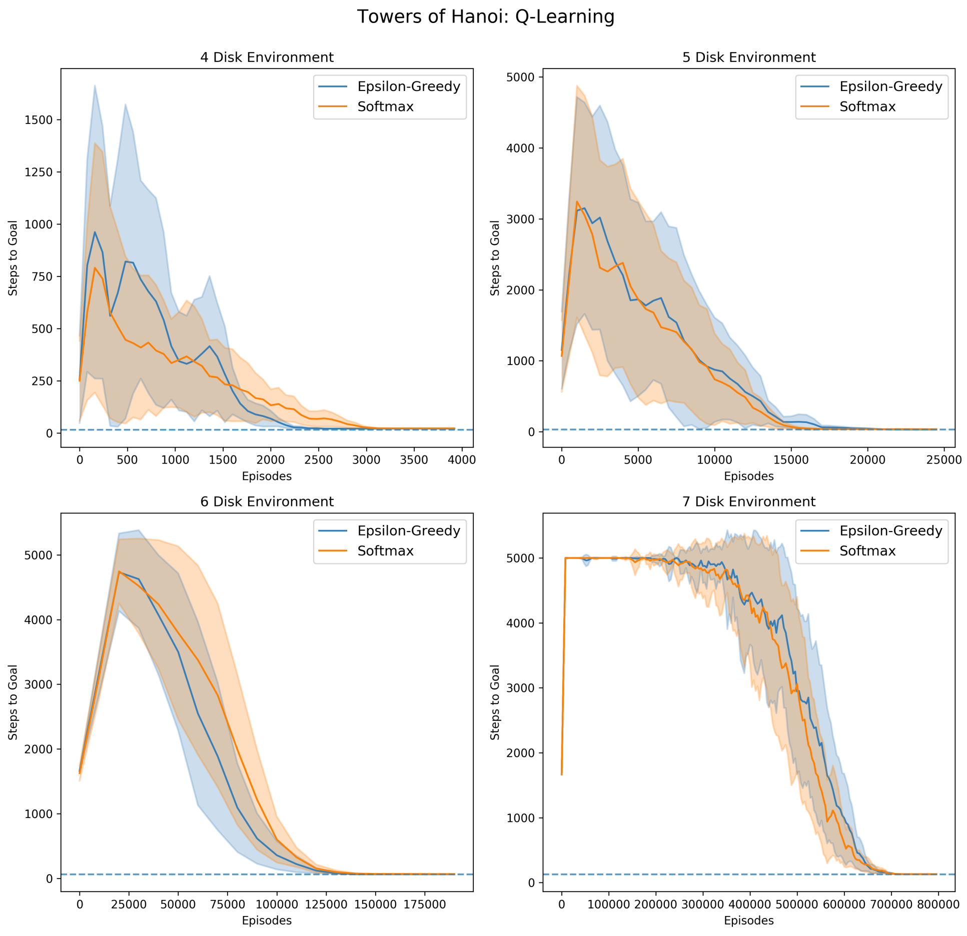


Figure 5.3: Towers of Hanoi: Q-Learning Performance

Imitation Action Grammar Learning Task

To solve the curse of dimensionality during imitation learning the designer endows the agent with the macro-actions listed in table 5.1. Afterwards, algorithm 6 is run. Figures 5.4 and 5.5 let us observe that our approach greatly outperforms the simple Q-learning baseline.[2]

The Macro-Q-learner endowed with the optimal N-disk policy grammar (see figure 5.4) converges to the optimal policy after very few experiences. For the 4, 5 and 6 disk environments the agent has identified the optimal policy consisting of the sequential selection of macros and primitives after around 25 episodes. Increasing λ from 0 to 0.1 does not seem to

[2]Note that we only show results for the first 100 episodes of learning. This is due to the fact that we are mainly interested in speeding up the early phases of learning. A possible future extension might allow the agent to automatically stop the grammar learning phase and to switch back to his primitive action space in order to refine her policy on a finer timescale.

provide a significant effect across both Sequitur or Lexis-based macro-actions and different environments. We also notice that learning in the 7 disk environment appears unstable for Lexis-based macro-actions and that in this case eligibility traces happen to act stabilizing.

Algorithm 6 Imitation Learning with CFAG Macro-Actions Extracted from Optimal Policy

```
Input: Trace from optimal policy, $\vartheta^\star_N = \pi^\star$ (or $\vartheta^\star_{N-1}$) for the $N$ (or $N-1$) disk problem.
Output: Optimal state-option value function and the corresponding policy for the $N$-disk problem
1: Call a CFG inference algorithm with $\vartheta^\star_N$ as an input. Obtain grammar $G(\vartheta^\star_N, \theta)$ and extract flat production rules $\mathcal{M}^{G(\vartheta^\star_N,\theta)}$.
2: Construct the Macro-action space $\mathcal{A}^{G(\vartheta^\star_N,\theta)} = \mathcal{A} \cup \mathcal{M}^{G(\vartheta^\star_N,\theta)}$.
3: repeat
4:     Macro-Q-learning Update with $\mathcal{A}^{G(\vartheta^\star_N,\theta)}$ as the augmented action space.
5: until Convergence
6: return $Q^\star_{\mathcal{A}^{G(\vartheta^\star_N,\theta)}}(s,m)$ and $\pi^\star(s) = arg\max_m Q^\star_{\mathcal{A}^{G(\vartheta^\star_N,\theta)}}(s,m)$.
```

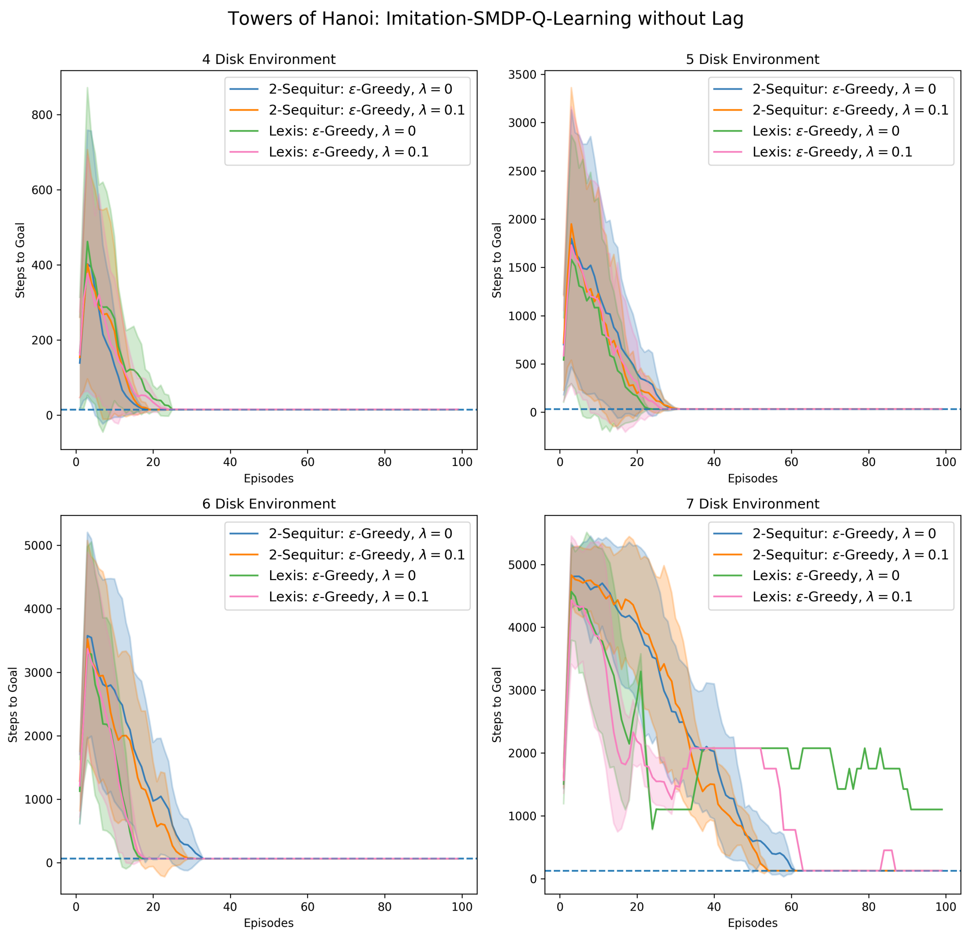


Figure 5.4: Towers of Hanoi: Imitation Learning Macro-Q-Learning Performance

In the case where the Macro-Q-learner has to generalize from an $N-1$ optimal grammar to

the N-disk environment we can observe that she can overcome initial exploration difficulties and then converges to a local optimum. After around 100 episodes the 5 disk agent has converged to a policy of length 46, while the optimal policy has length 31. For the 6 disk agent the same occurs after around 200 episodes with a policy of length 105, while 63 is optimal. Furthermore, eligibility traces enhance learning both with Lexis and 2-Sequitur-based macros (see e.g. 7 disk environment).
A potential reason for the sub-optimality of the solution is that the macro-actions influence the exploration behavior of the agent (see section 2.3.1) in a substantial way. Macro-actions lead the agent into specific parts of the sub-space and can thereby "abstract away" substantial amounts of experiences. Hence, we conclude that exploration becomes even more important when using statically inferred sub-optimal macros. A possible way how this might be overcome, is to have an increasing exploration schedule. Furthermore, similar to inter-option learning it might be helpful to use inter-macro learning to also update the value of the primitive actions. As soon as the agent is stuck in a local optimum, one might then turn of the macro-actions and continue using only primitive actions. Thereby, more exploration is induced since the agent has to make decisions at every time step.

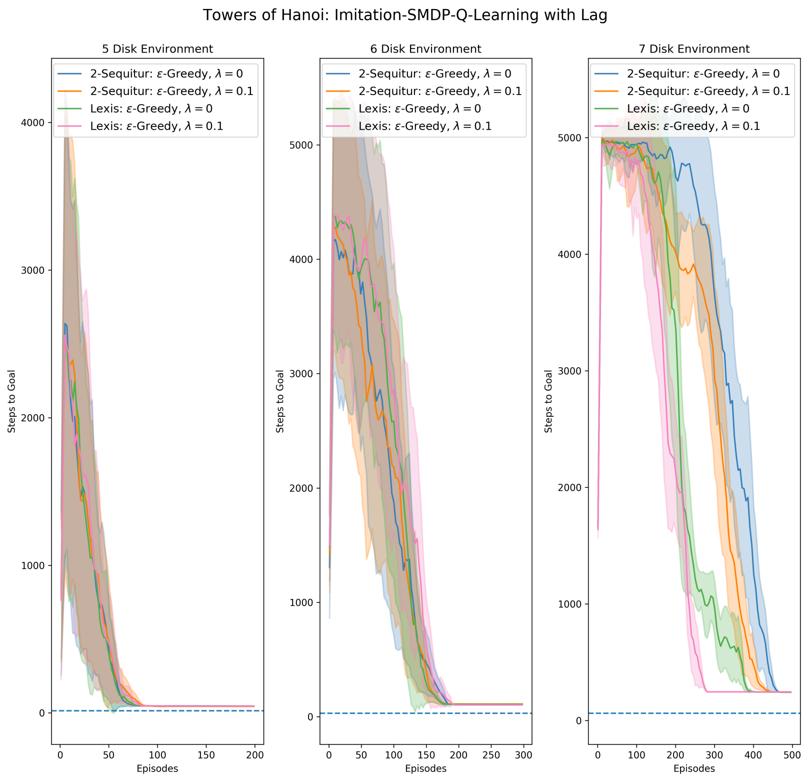


Figure 5.5: Towers of Hanoi: Imitation Learning Macro-Q-Learning Performance with Lag

In conclusion, we found that in the N-disk environment with N-disk macros the fast convergence is due to the fact that the augmented action space is obtained by decomposing the optimal deterministic policy into a straight-line grammar. The agent simply has to order the temporally-extended actions which drastically reduces the complexity of the reinforcement learning problem. The agent in the N-disk environment with $N-1$-disk macros, on the other hand, has to perform a more difficult task. Given the decomposition of the easier environment, the agent is required to learn how to generalize to the more complicated setting. In order to do so, she has to essentially identify the recursive algorithm that solves the Towers of Hanoi problem for all N disk settings. While doing so she converges to a local optimum. We now turn to the hard case of online learning of both actions and grammatical structures.

Online Action Grammar Learning Task

In the online experiments we chose the following hyperparameters: $N_{init} = 200$, $N_a = 312$, $k_{init} = \{4, 6, 8\}$ for $N = \{4, 5, 6\}$ respectively. Furthermore, our updating schedule for k-Sequitur decreases k by 1 after each grammar update (see table 5.2). The agent starts off by running 200 Q-Learning iterations. Afterwards, we rollout 10 trajectories and encode the best performing one using either Sequitur or Lexis. We extract macro-actions from the flat production rules and the agent then continues learning using the augmented action space. After 312 SMDP-Q-Learning iterations, we again obtain 10 trajectories and extract a new set of macros. This way the agent alternates between action learning during the SMDP-Q-Learning phase and grammar learning. Figure 5.6 plots the learning performance of the online Action Grammar agent. The first row depicts an agent whose action value table is reset after each grammar update. In the second row, on the other hand, the agent carries over the values (as described in section 4.2) which were previously learned for actions that remain in the macro-action space after the update.

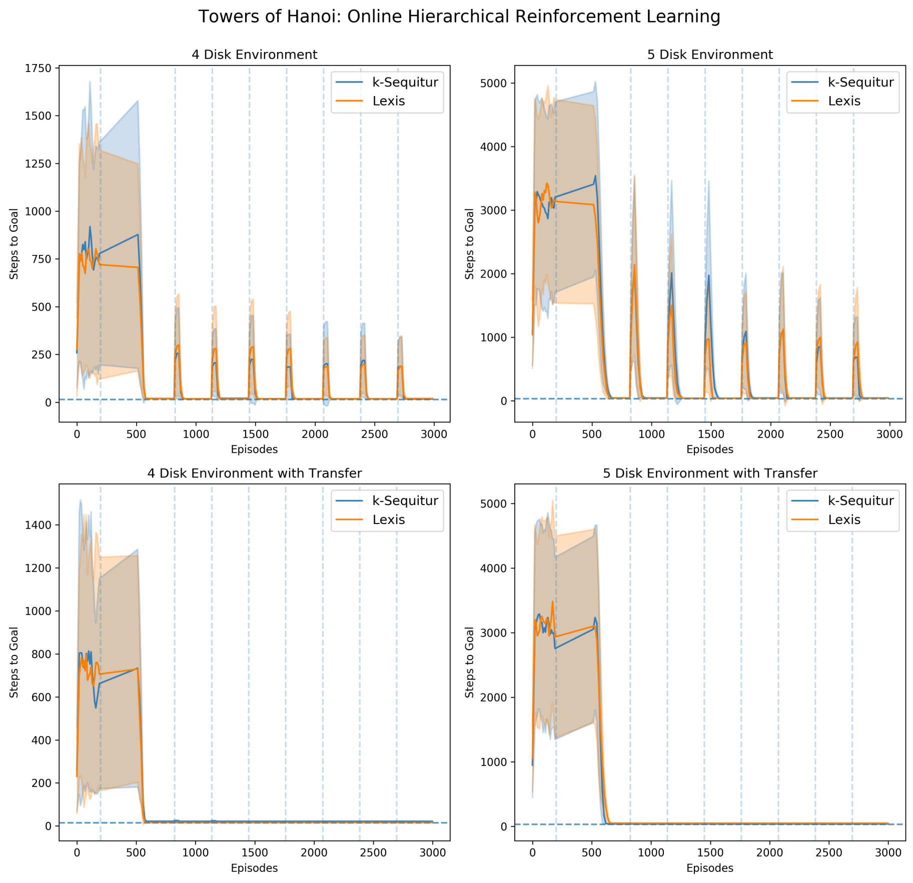


Figure 5.6: Towers of Hanoi: Online CFAG Macro-Q-Learning Performance

k-Sequitur Schedule for Different Disk Environments					
Disks (N)	Initial	After 200 Eps	After 624 Eps	...	After 2808 Eps
4	4	3	2	...	2
5	6	5	4	...	2
6	8	7	6	...	2

Table 5.2: Towers of Hanoi: k-Updating schedule for Online CFG-Macro-Q-Learning

One can observe that the grammar updates effect the agent differently across the learning process. The first updates greatly help the agent in structuring his exploration process. The later updates essentially "shock" the agent and reset his learning progress. Still, this shocking effect tends to decrease which indicates a convergence in the refinement of the grammar. Allowing for value transfer between grammar updates reveals the strength of the grammar in capturing the hierarchical structure underlying the environment. The variance induced by the grammar update can be circumvented. This is due to the CFG algorithm inferring the same macros and therefore all values can be transferred between updates. Hence, a convergence in the grammar space allows for robust learning. For all three environments the action grammar agent is able to extract the simple recursive structure after two grammar updates and 625 Macro-Q-Learning episodes.

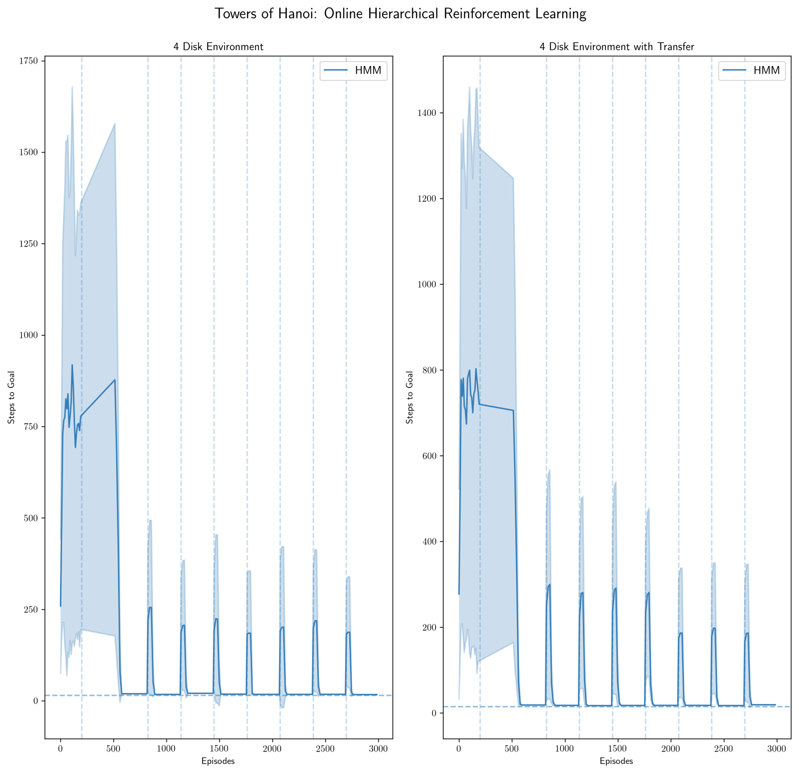


Figure 5.7: Towers of Hanoi: Online PAG Macro-Q-Learning Performance

Figure 5.7, on the other hand, shows learning with macro-actions being sampled from a 4 hidden state HMM that is trained based on 100 rollout trajectories. We can directly observe that the value transfer does not decrease the variance induced by the grammar updates. This is due to the fact that the sampling variance induced by the HMM leads to a set of macros that

	Sequitur			Lexis			HMM		
Update	1	5	8	1	5	9	1	5	8
t	200	1761	2697	200	1761	2697	200	1761	2697
$\lvert\mathcal{M}\rvert$	3.4	1.8	1.4	0.2	0.4	0.8	3.8	4.8	3.8
Std	2.06	1.17	0.8	0.4	0.49	0.4	0.98	0.4	1.17
$\lvert\pi\rvert$	151.2	18.2	17	104.8	17.6	19.2	42.6	18.6	19.4
Std	74.7	3.54	2.28	56.55	1.74	3.12	13.31	4.22	4.45

Table 5.3: Towers of Hanoi: Grammar Learning Performance for the 4 Disk Environment

changes at every single grammar update. Hence, less values can be transferred and learning remains instable.
We can explore the behavior of the extracted grammars throughout the learning process. Table 5.3 depicts statistics which capture the grammar convergence behavior.
The number of extracted macros changes throughout the updates. Sequitur in the beginning extracts several macros while only one is optimal for the 4 disk environment ($''abd''$). After a few updates this number is correctly reduced. The agent converges to both the optimal policy as well as the optimal set of macros. For G-Lexis, on the other hand, the same holds. Here though, the number of macros increases throughout learning. For the HMM-based macros our intuition is confirmed. It constantly extracts too many macro-actions and does not converge in the grammar space. Hence, also transferring values between updates does not help. All learning results are summarized and compared in figure 5.8.
In conclusion, we have shown that macro-actions extracted with k-Sequitur and Lexis provide an efficient solution to the curse of dimensionality in the Towers of Hanoi problem. More specifically, we have shown that both imitation learning as well as online learning performance can be improved sample efficiently. Therefore, our CFAG approach was validated. Macro-actions based on HMM samples, on the other hand, did not perform well. The additional sampling variance was larger then the gain in action learning. The next section now implements the option discovery approaches introduced in chapter 4.3.

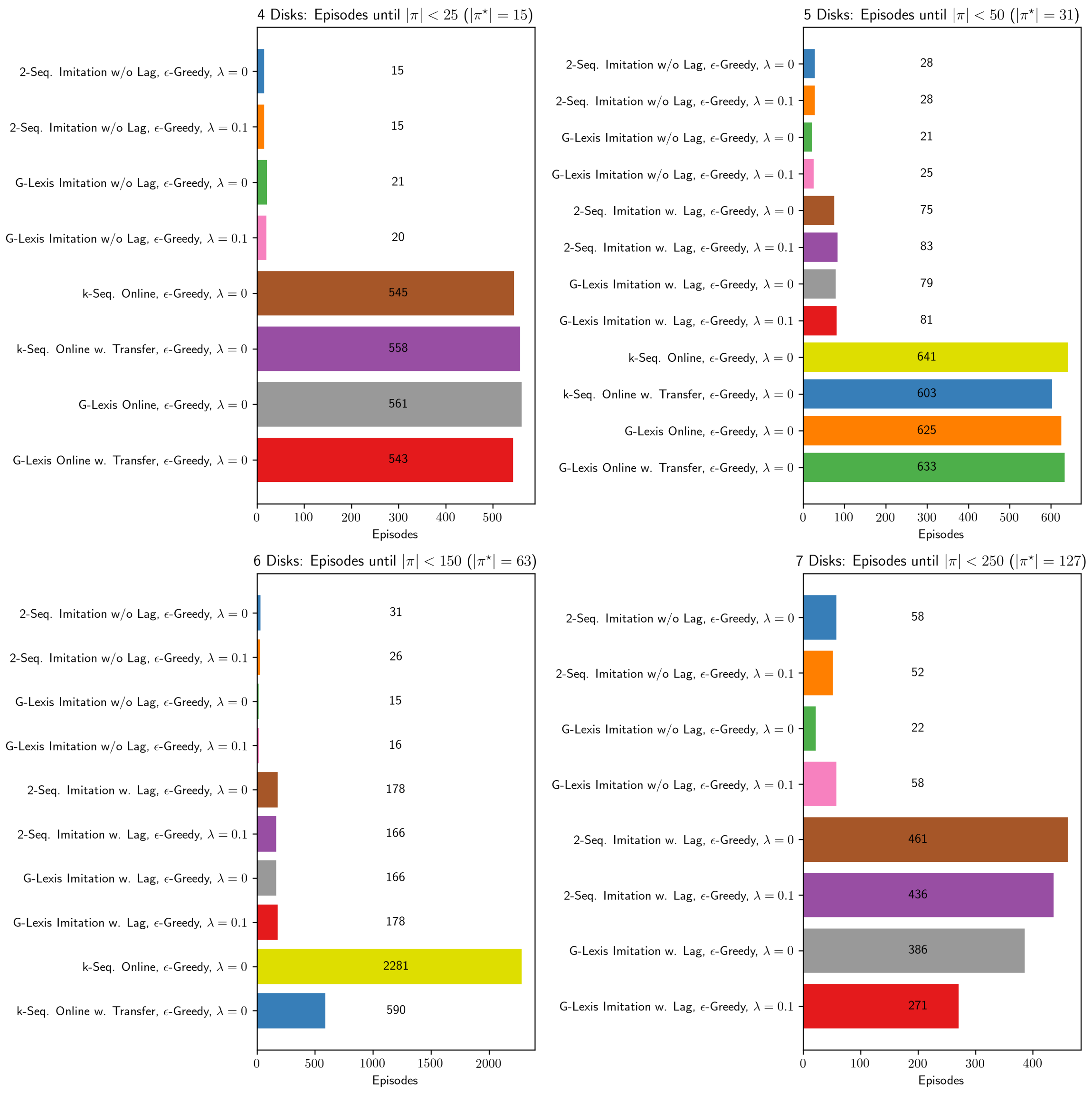


Figure 5.8: Towers of Hanoi: Learning Comparison. Not shown Q-Learning (ϵ-greedy/Softmax) baselines for 4, 5, 6, 7 disk environments with goal achievement after 2240/2960, 18599/16000, 110000/11000, 648000/652000 episodes, respectively.

5.2 The Four Room Problem

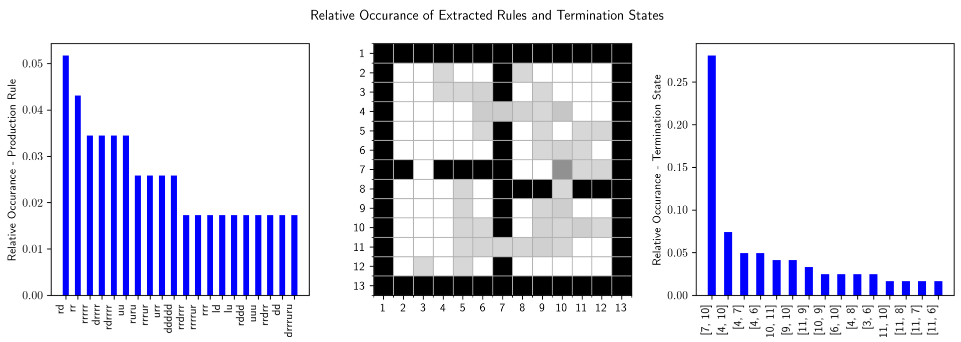
The four room problem as introduced by Sutton et al. (1999) (see figure 2.3) provides a baseline environment to test our option discovery approach outlined in section 4.3. The agent is randomly placed within the environment at the beginning of each episode. She can move in all directions as long as she does not walk into a wall. If she reaches the goal position (indicated in grey) the agent receives a reward of 1 and the episode terminates. For every other transition she yields a null reward. Hence, we are again dealing with sparse rewards which complicate the reward-based learning. She discounts her rewards with a factor of 0.9. State transitions are successful in 90 percent of the cases. The possible moves are denoted by d ("down"), u ("up"), l ("left"), r ("right"). Hence, the terminal vocabulary is defined as $\Sigma = \mathcal{A} = \{d, u, l, r\}$. In all experiments outlined below and if not otherwise stated, we choose the following hyperparameters:

Four Room Hyperparameters:
1. Goal state: $(7, 10)$
2. Exploration parameter: $\epsilon = 0.1$
3. Discount factor: $\gamma = 0.9$
4. Learning rate: $\alpha = 0.05$
5. Number of CFAG options used in learning: $M = 4$
6. Number of traces on which we infer a grammar: $N_g = 104$
7. Number of initial Q-Learning iterations: $N_{init} = 20$
8. Number of episodes between grammar updates: $N_a = 36$
9. Number of grammar updates: 5 (at episodes $\{20, 56, 92, 128, 164\}$)
10. k-Sequitur hyperparameter: $\theta = k \in \{5, 4, 3, 2\}$
11. Updating schedule for k-Sequitur hyperparameter: $k_{next} = k_{prev} - 1$ if $k_{prev} > 2$
12. Maximal surprisal (option termination criterion): $\xi = 7.5$
13. Maximum steps of PCFAG option execution: 10

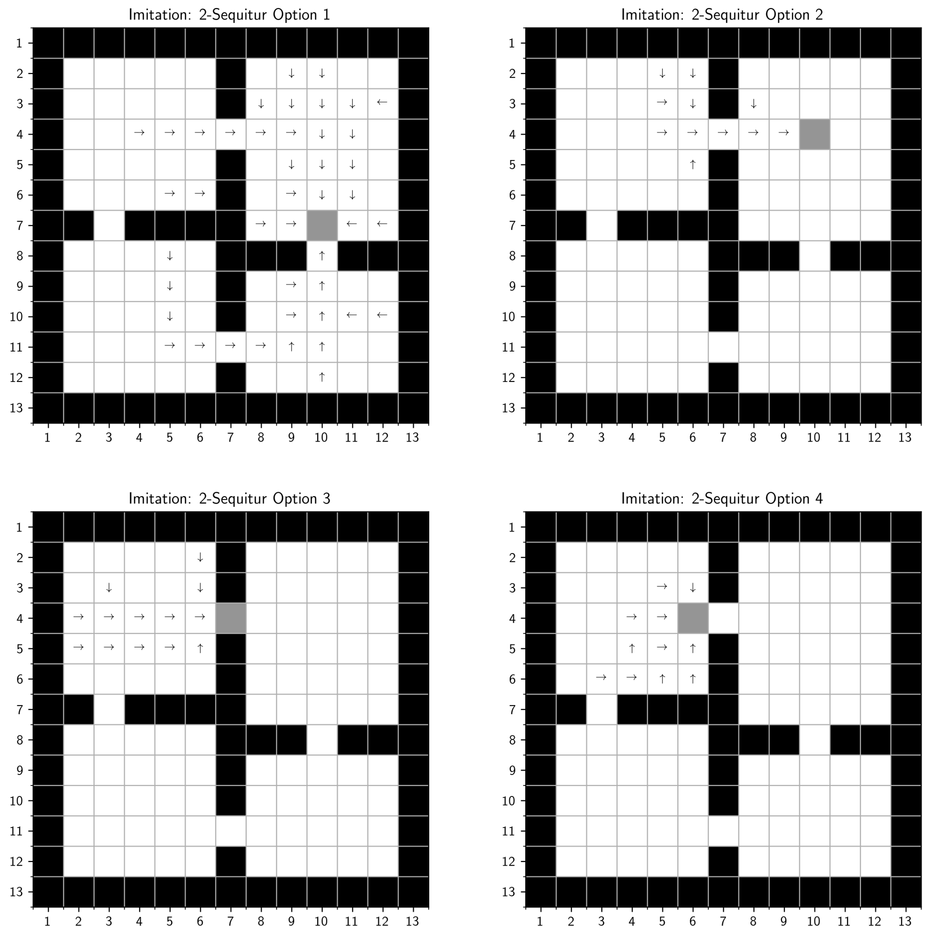
5.2.1 Option Discovery with Context-Free Action Grammars

Imitation Action Grammar Learning Task

During the imitation learning task we only learn a grammar once before the start of the action learning phase. More specifically, we infer production rules using CFG grammar induction algorithms on 100 expert traces from a converged Q-learning policy after 10,000 training episodes. Following our CFG-based option construction approach (see section 4.3), figure 5.9a depicts how often the individual production rules inferred by 2-Sequitur occur in the encoded concatenated action sequence. Unsurprisingly, most of them seem to be directly driven by the goal location and the bottleneck of the hallway passages. Since the goal location is located on the right of the environment most production rules incorporate some number of right moves. The middle part of the figure shows the environment in which we have colored the states by how often a production rule terminates in it. A corresponding bar chart can be found in the right part of the figure. We can identify that most extracted production rules directly lead the agent to goal state. Furthermore, since the goal is allocated in the right half of the environment, also most termination states are located in the right half.

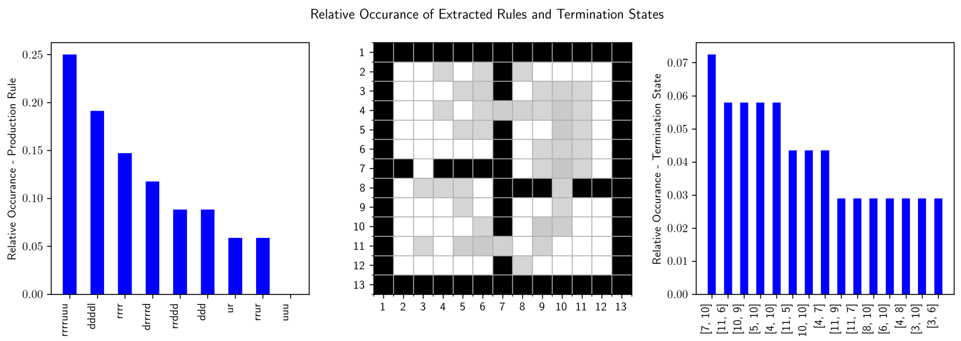


(a) 2-Sequitur Imitation Learning: **Left**. Relative occurrance of the extracted production rules in the encoded concatenated action sequence. **Middle**. Four room environment with cells colored according to how often production rules terminate in specific state. **Right**. The corresponding bar chart.

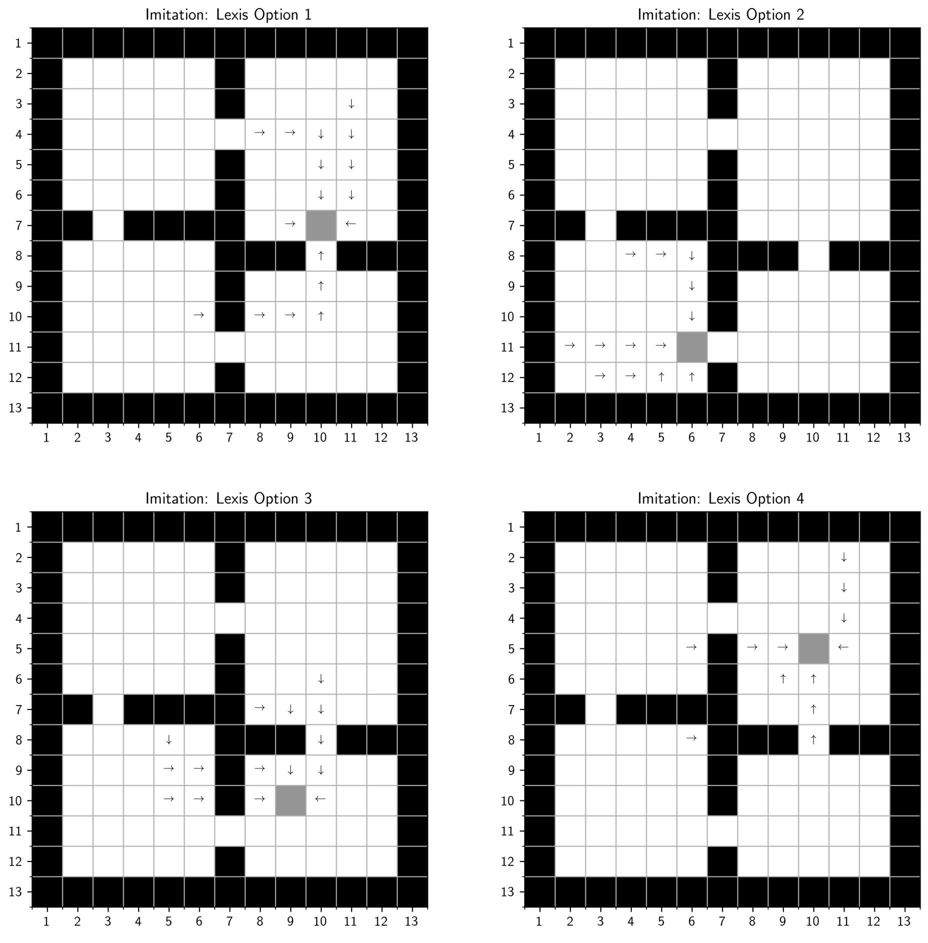

Imitation: 2-Sequitur Option 1
Imitation: 2-Sequitur Option 2
Imitation: 2-Sequitur Option 3
Imitation: 2-Sequitur Option 4

(b) 2-Sequitur Imitation Learning: Option Set Discovered using 100 traces of expert behavior.

Figure 5.9: Four Rooms: Imitation Learning - 2-Sequitur Option Construction.



(a) Lexis Imitation Learning: **Left**. Relative occurrance of the extracted production rules in the encoded concatenated action sequence. **Middle**. Four room environment with cells colored according to how often production rules terminate in specific state. **Right**. The corresponding bar chart.


Imitation: Lexis Option 1
Imitation: Lexis Option 2
Imitation: Lexis Option 3
Imitation: Lexis Option 4

(b) Lexis Imitation Learning: Option Set Discovered using 100 traces of expert behavior.

Figure 5.10: Four Rooms: Imitation Learning - Lexis Option Construction.

We construct the option set for the top 4 most terminated in states (for $M = 4$: $\{(7, 10), (4, 10), (4, 7), (4, 6)\}$). Figure 5.9b displays the set of automatically inferred options using 2-Sequitur as the CFG induction algorithm. The options lead through the hallways and therefore also capture the notion encoded in the manual specification of the hallway options (see figure 2.4). Furthermore, the inferred intra-option policies become sparser as the number of options increases. This is only natural since the termination states are chosen, so that the number of productions that terminate in them decreases.

Figure 5.10a displays the corresponding encoding statistics for the inferred G-Lexis grammar. Compared to the 2-Sequitur context-free grammar, Lexis clearly estimates fewer production rules (part a). The frequently used flattened production rules are also on average longer than for the corresponding Sequitur encoding. This verifies the weakness of G-Lexis to overfit noise and to estimate unreasonably long productions. Again, most of the states in which the productions terminate in are located in the right half of the environment which resembles the proximity to the goal state (state $(7, 10)$). Furthermore, the distribution of termination states has a higher entropy and is less structured. This is a first signal that G-Lexis might not be well suited for the grammar learning part of the Action Grammar loop since it is unable to differentiate between important states in the state space. Again, figure 5.10b shows the corresponding top four option set.

But how does the HRL agent perform when being endowed with this set of options? We now display results for the action learning part of the imitation learning problem. Here the agent chooses the first four options ($M = 4$, either Lexis or Sequitur-based) to augment his action space before starting to learn for 200 episodes ($N_a = 200$). The performance of the HRL agents is averaged across 5 learning runs and 100 on-policy rollouts. Figure 5.11 shows the learning results for three different CFG inference algorithms.

- The first row depicts the learning results using the top four options inferred using 2-Sequitur on 100 expert traces and depicted in figure 5.9b.
- The second row depicts the learning results using the top four options inferred using 4-Sequitur on 100 expert traces.
- The third row depicts the learning results using the top four options inferred using G-Lexis on 100 expert traces and depicted in figure 5.10b.

Both Sequitur-based options are able to capture the hierarchical structure of the environment needed to solve the reinforcement learning problem at an efficient pace. After approximately 25 episodes the problem can be regarded as solved. Furthermore, options extracted using 2-Sequitur as the CFG inference algorithm perform the best. This might be due to the fact that the average length of the 100 expert trajectories is approximately 10. Hence, $k = 2$ sensibly balances the need to infer enough production rules while not overfitting sub-optimal noise. Increasing k only increases the variance of the learning process while not increasing the rate of convergence. G-Lexis performs the worst. When inspecting the options in figure 5.10b we observe that the production rules tend to be longer in terms of actions than for Sequitur. Furthermore, we only infer "sparse" options which can only be initiated in a few states. Furthermore, G-Lexis option-based learning does not efficiently propagate value information compared to Sequitur. After 20 iterations both Sequitur-based agents have learned a substantial amount about the proximity to the goal state. The G-Lexis-based agent, on the other hand, did not estimate the values of the neighboring states well.

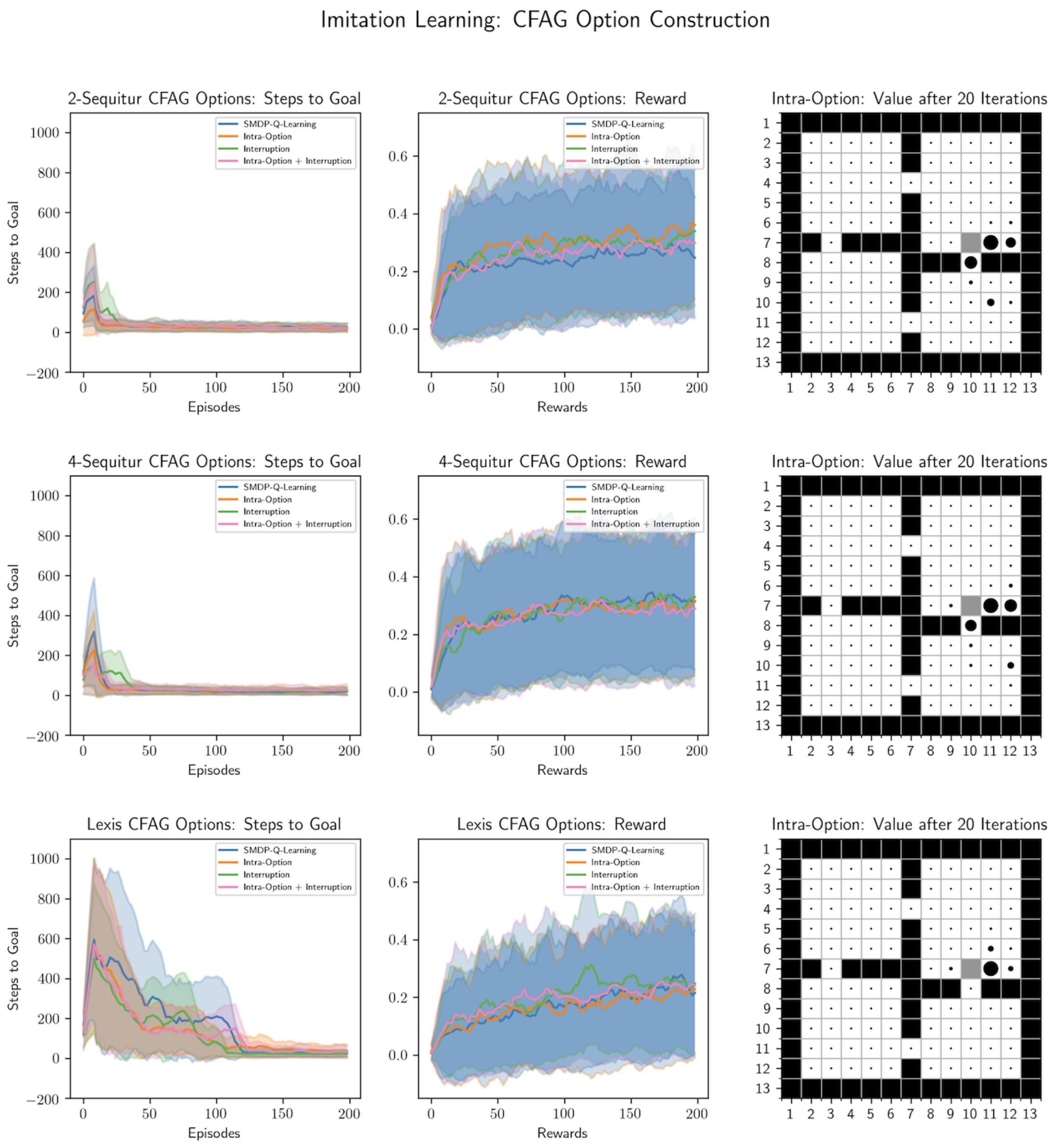


Figure 5.11: Four Rooms: Imitation Learning - CFAG Options Learning Performance

Hence, we conclude that only Sequitur-based option constructions are suitable to infer the hierarchical structure innate to the four rooms problem. They reliably produce learning results comparable to the manually constructed set of hallway options. G-Lexis, on the other hand, does not achieve competitive results. In the following we focus on the online version and show results for the fully-automated approach.

Online Action Grammar Learning

Next, we display results for the online RL framework (see section 4.3). Before the agent starts to alternate between grammar and action learning, she first runs a few simple Q-learning episodes ($N_{init} = 20$). Experiments have shown the necessity for a minimal hierarchical signal encoded in the sequences on which we run the first grammar induction.

After this initial set of Q-Learning iterations we rollout a set of 104 traces and infer the first grammar. We then construct the first set of corresponding options and run SMDP-Q-Learning with the inferred options. After every 36 episodes the option set is updated. We can visualize the development of the different options throughout the alternating grammar and action learning phases. Figures 5.12, 5.13 and 5.14 show how the set of inferred options (only top three shown) changes throughout the updating (updates 1, 3 and 5).
We can observe that unsuccessful transitions induce noise into the intra-option policies. Still most of the options capture the hierarchical nature of the environment and lead the agent through the hallways. As learning progresses the option set becomes more and more sophisticated and the amount of noise in the intra-option policies appears to decrease.

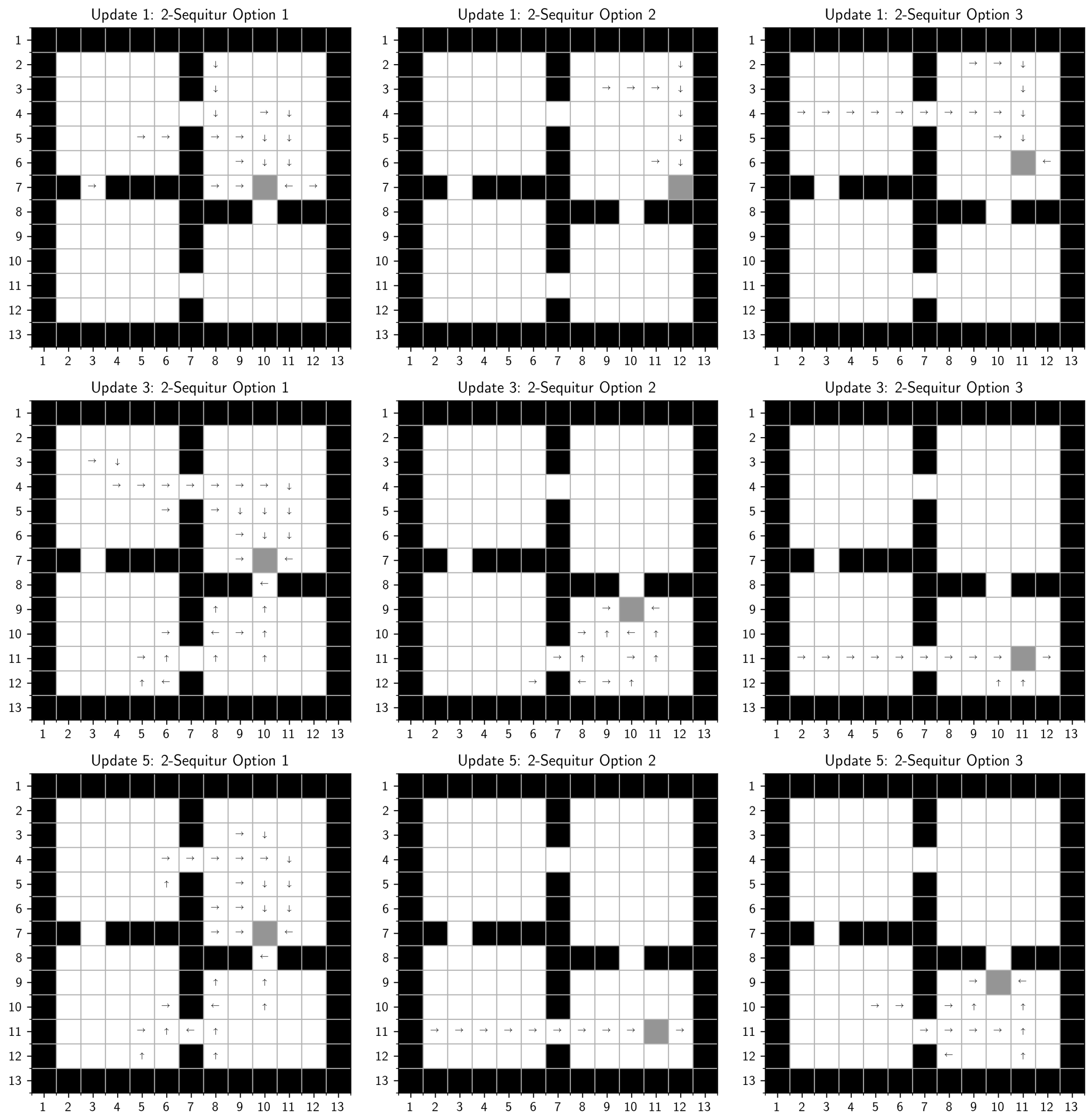

Update 1: 2-Sequitur Option 1
Update 1: 2-Sequitur Option 2
Update 1: 2-Sequitur Option 3
Update 3: 2-Sequitur Option 1
Update 3: 2-Sequitur Option 2
Update 3: 2-Sequitur Option 3
Update 5: 2-Sequitur Option 1
Update 5: 2-Sequitur Option 2
Update 5: 2-Sequitur Option 3

Figure 5.12: Four Rooms: CFAG Options Development for 2-Sequitur

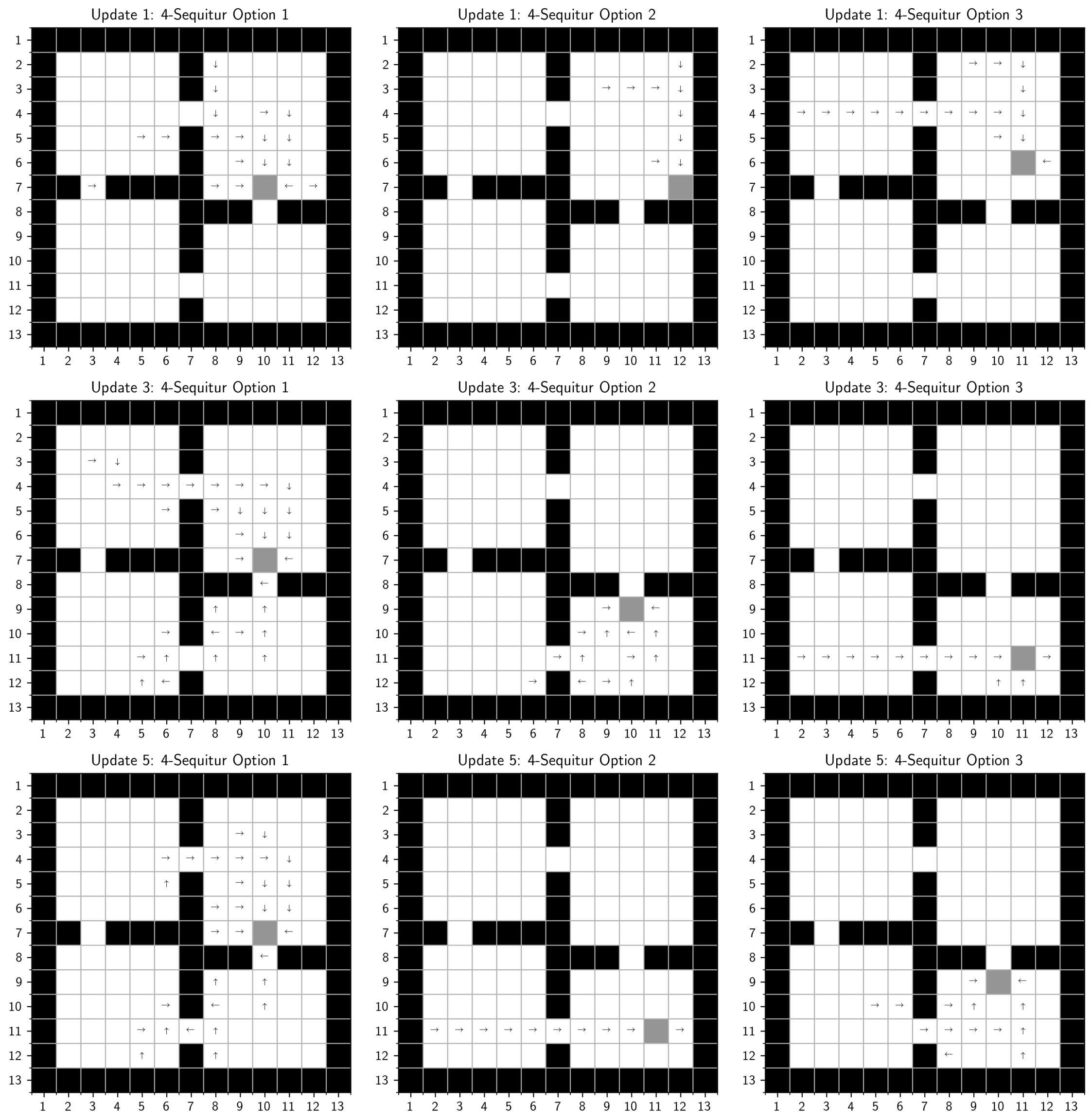

Update 1: 4-Sequitur Option 1
Update 1: 4-Sequitur Option 2
Update 1: 4-Sequitur Option 3
Update 3: 4-Sequitur Option 1
Update 3: 4-Sequitur Option 2
Update 3: 4-Sequitur Option 3
Update 5: 4-Sequitur Option 1
Update 5: 4-Sequitur Option 2
Update 5: 4-Sequitur Option 3

Figure 5.13: Four Rooms: CFAG Options Development for 4-Sequitur

When comparing 2-Sequitur options with 4-Sequitur and adapting k-schedule options, we do not observe drastic differences. In terms of functionality there appears to be an option which leads from the lower left into the right room, another option leads to the top part of the lower right room and the final (and most dense) option directly leads to the goal state.

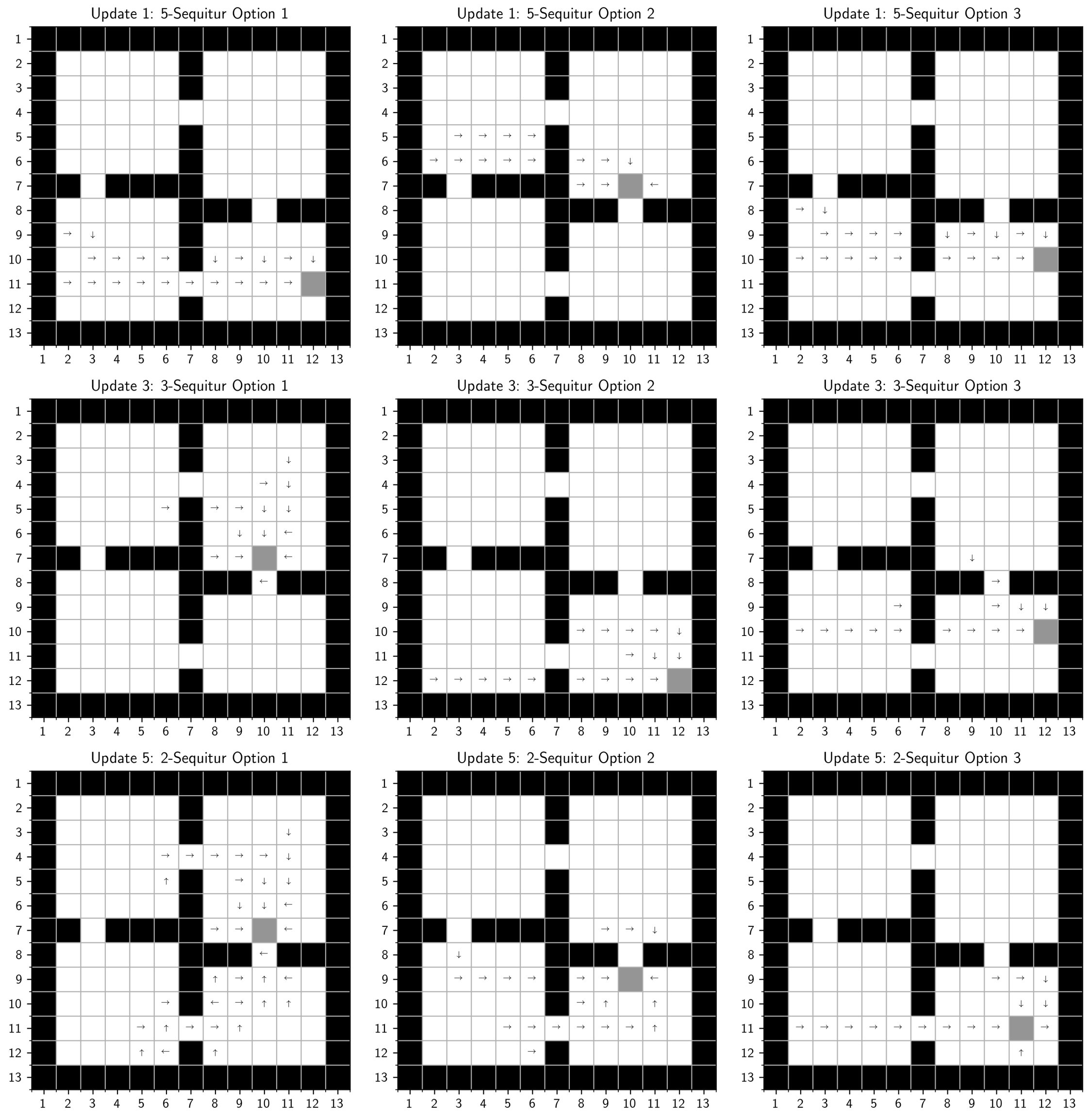

Update 1: 5-Sequitur Option 1
Update 1: 5-Sequitur Option 2
Update 1: 5-Sequitur Option 3
Update 3: 3-Sequitur Option 1
Update 3: 3-Sequitur Option 2
Update 3: 3-Sequitur Option 3
Update 5: 2-Sequitur Option 1
Update 5: 2-Sequitur Option 2
Update 5: 2-Sequitur Option 3

Figure 5.14: Four Rooms: CFAG Options Development for an Adaptive Sequitur Schedule

Online learning results can be found in figure 5.15:

1. Row one depicts the learning results using the top four options inferred using 2-Sequitur.
2. Row two depicts the learning results using the top four options inferred using 4-Sequitur.
3. Row three depicts the learning results using the top four options inferred using G-Lexis.

The online algorithm with CFG-inferred options converges slower than the manual hallway options. Still, all HRL agents achieve convergence after 100 episodes, while Q(λ) converges after around 150 episodes. Furthermore, 4-Sequitur options with interruption (best performance) achieve convergence after 50 episodes.

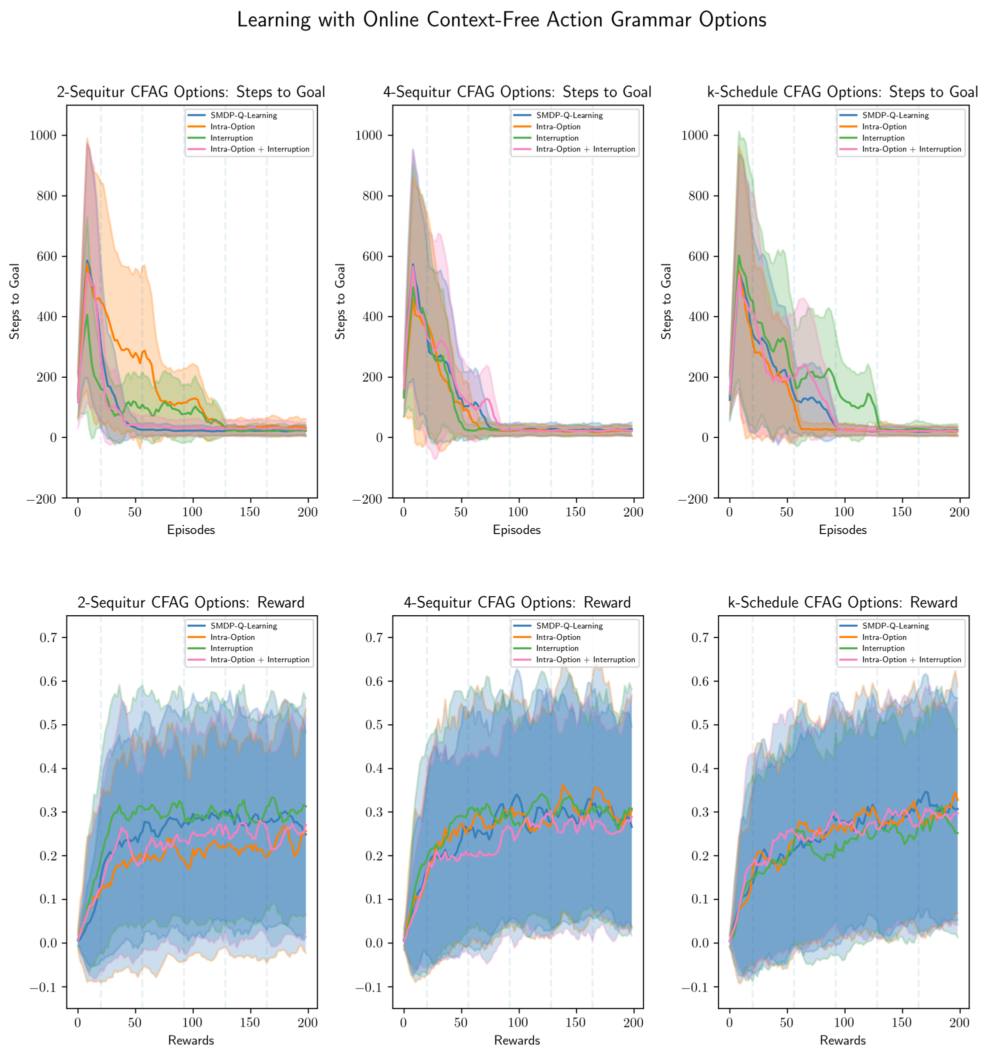


Figure 5.15: Four Rooms: Online Reinforcement Learning with CFAG Options

In conclusion, we have shown that our approach helps in the early stages of learning and reduces the variance of the learning results (compare with figure 2.5). Furthermore, we want to emphasize the effect of the first grammar update in helping the agent to structure his exploration process in the four rooms environment.

5.2.2 Option Discovery with Probabilistic Context-Free Action Grammars

In this section we implement the framework introduced in section 4.3 using stochastic grammar inference to construct an option. As previously discussed we obtain such a semi-

Markov option by training a sequence model such as a HMM or RNN. If the agent chooses to execute the option, actions are sampled based on a history dependent posterior distribution. The option terminates either if the surprisal of a specific transition surpasses a threshold or if a maximum number of steps is reached.

Imitation Action Grammar Learning Task

In order to select a first set of possible architectures in the imitation learning case, we train HMMs with different numbers of hidden states and compute both the Akaike information criterion (AIC) and the Bayesian information criterion (BIC). The left part of figure 5.16 shows the results of this exercise. In the following we will use the best three performing models (12, 16 and 21 hidden states). For the RNN model selection, on the other hand, we compute accuracies (see right part of figure 5.16). We observe that using more RNN cells does not significantly improve the predictive accuracy.

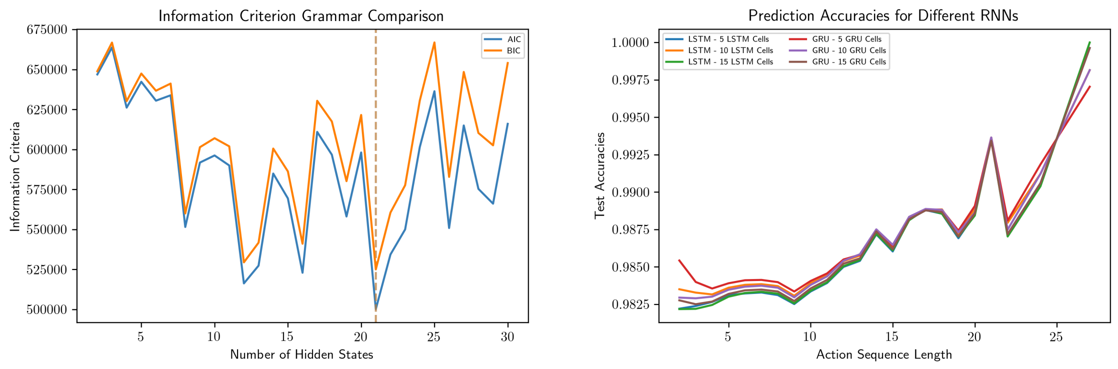


Figure 5.16: Four Rooms: Information Criteria for Different HMMs and RNN Test Accuracies

Based on these observations we choose and train the following architectures (either online at grammar update time or once in the imitation learning setting):

Type	HMM Architecture	LSTM Architectures	GRU Architectures
1	13 Hiddens	5 LSTM Cells	5 GRU Cells
2	26 Hiddens	10 LSTM Cells	10 GRU Cells
3	29 Hiddens	15 LSTM Cells	15 GRU Cells

Table 5.4: Four Rooms: Recurrent Neural Network Architectures.

All RNN architectures start with an embedding layer which transforms the state of the agent into a one-hot encoded vector. Following the RNN cells there is a dense layer which feeds an output Softmax layer which we treat as a posterior distribution.
Following our replay surprisal approach outlined in section 4.4 we compute the 10-fold cross-validated replay surprisal on 10000 expert traces for all of the above models and across different action sequence/history lengths (see figure 5.17). This yields a very interesting and intuitive revelation: HMMs are well equipped to make prognosis for short-term behavior. For longer action sequences they become less suited. This is most likely due to the Markov assumption. LSTMs, on the other hand, are badly suited for sequences with short lengths but develop their potential as the sequence length increases. At a sequence length of around

12 they start to outperform the HMM in terms of replay surprisal. GRU-based RNNs, on the other hand, achieve this comparable performance already after a sequence length of 6. Hence, we can conclude the following: HMMs are adequate for modeling short dependencies, GRUs are suited for modeling intermediate dependencies and LSTMs for long-term dependencies.

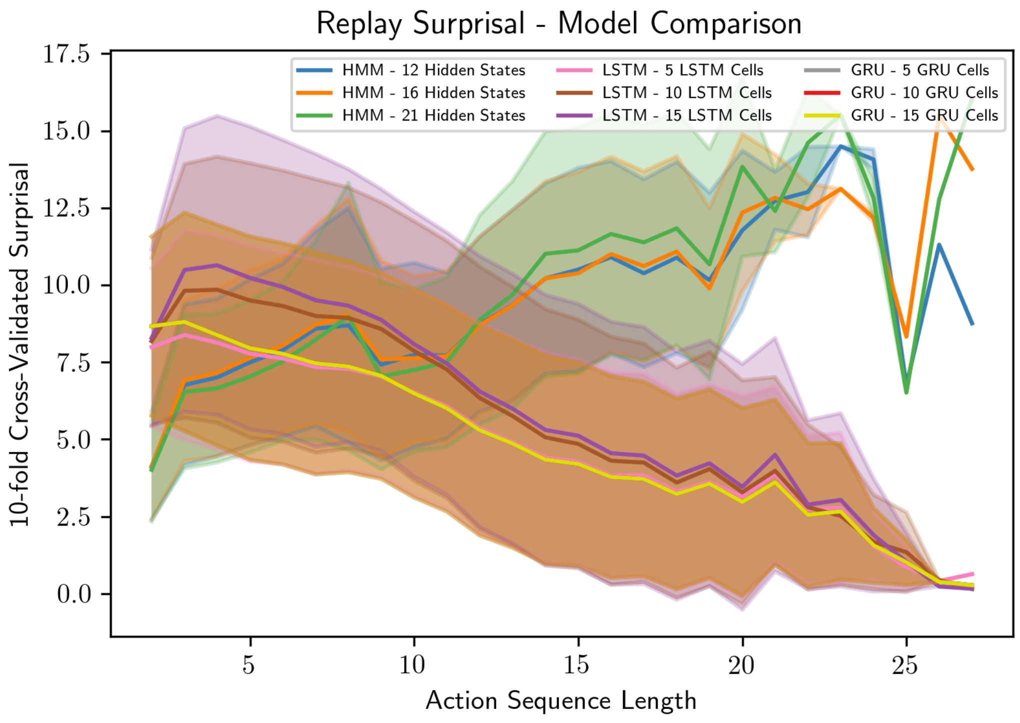


Figure 5.17: Four Rooms: 10-fold Cross-Validated Replay Surprisal

Again, we perform imitation learning by training the option on 10,000 expert episodes. The RNNs are trained using mini-batch gradient descent (Adam optimizer) with batch size of 64 for 5 epochs. Afterwards, the agent starts learning from environment interactions and with the help of the stochastic option. Figure 5.18 shows that we are again able to learn at an extremely efficient pace. Learning performance is comparable to the manually constructed hallway options. Still, we have to note that the amount of required training episodes is quite high and this approach might not be regarded as sample efficient. Further investigation is needed. Finally, we note that the HMM and GRU constructions perform the best and with the least variance.

Online Action Grammar Learning Task

As for the Towers of Hanoi environment we also perform online Reinforcement Learning. We first run 20 episodes of Q-Learning before obtaining a set of 104 rollouts. These are then used to train a first PAG option as described above. Afterwards, we run 36 episodes of SMDP-Q-Learning with the augmented action space before we update the option estimate again. We repeat the two alternating steps. Figure 5.19 shows the learning results (averaged over 5 agents and 10 rollouts every 2 episodes) for this experiment. The 4 hidden state HMM option achieves convergence after 25 episodes, 5 LSTM cells RNN option after 35 and the

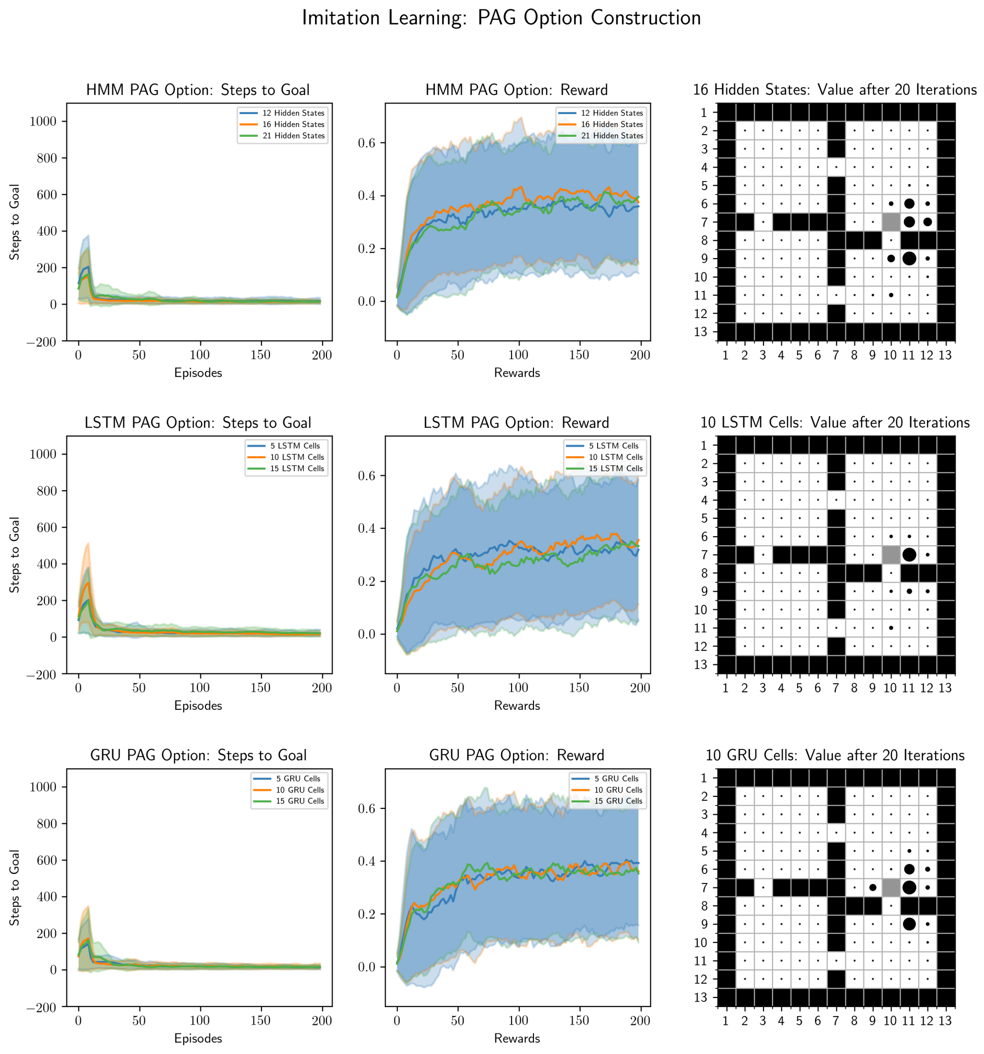


Figure 5.18: Four Rooms: Imitation Learning with PAG Options

15 GRU cells RNN option after 35 as well. Furthermore, the learning variance is reduced compared to baseline results (see figure 2.5). Successive grammar updates do not strongly improve learning and again we highlight the importance of the first inferred grammar. We highlight that the best performing HMM option has as many hidden states as the number of rooms or hallway options. Again, all learning results are summarized in the bar charts in figure 5.20.
In summary we have shown that PAG options provide a good way to improve learning in the imitation and online Reinforcement Learning task. They were able to decrease variance and improve learning speed. CFAG options provide an alternative that is more sample efficient. We hypothesize that this is due to the fact that there are less parameters to be inferred.

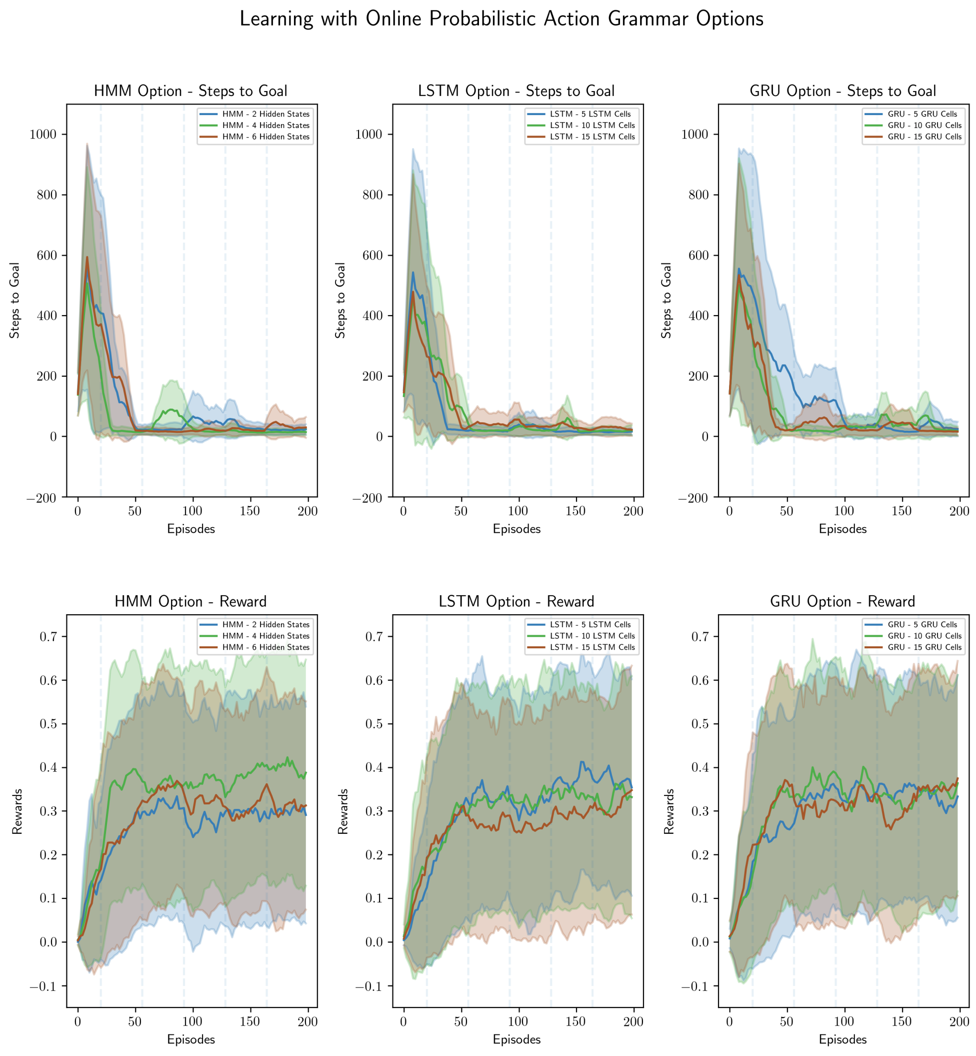
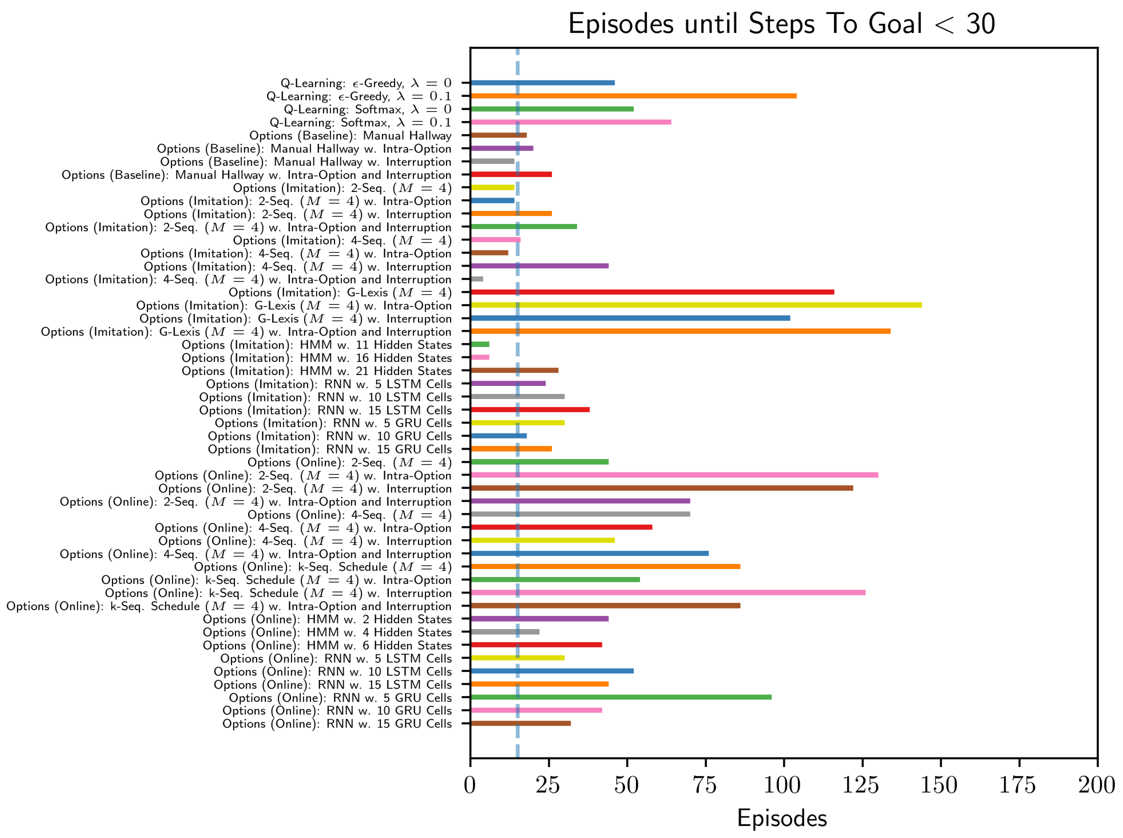
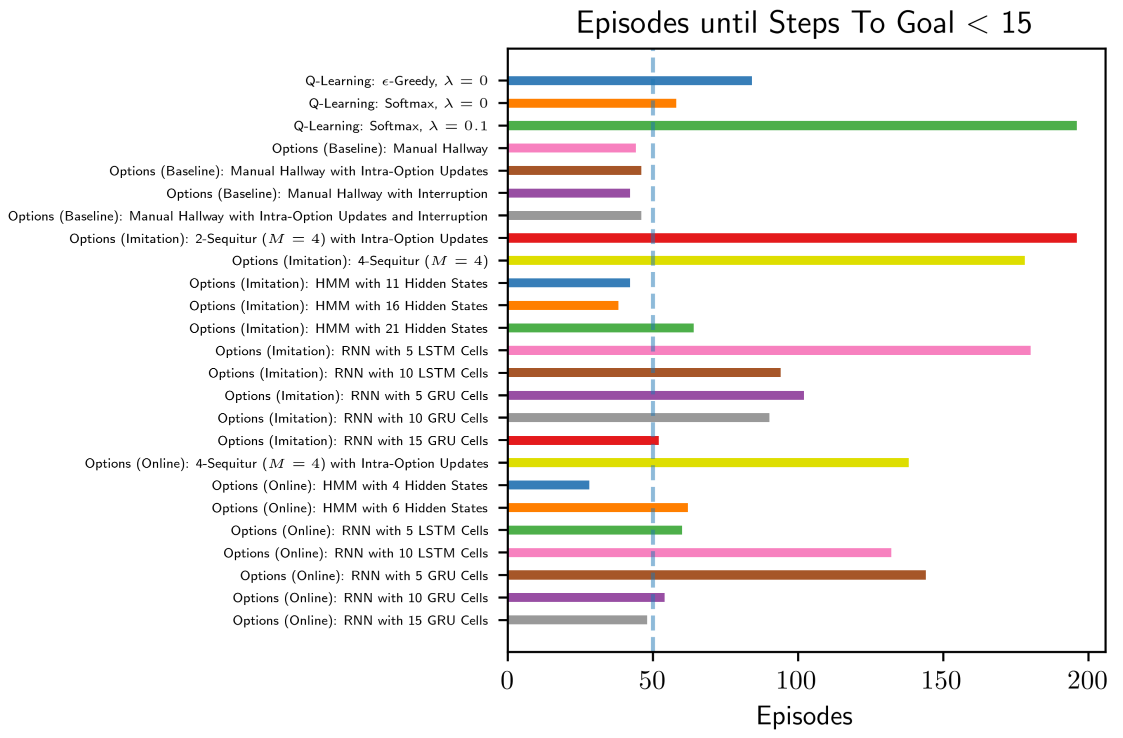


Figure 5.19: Four Rooms: Online Reinforcement Learning with PAG Options



(a) Average Number of Episodes to Reach 30 Steps to Goal Position



(b) Average Number of Episodes to Reach 15 Steps to Goal Position

Figure 5.20: Four Rooms: Learning Comparison

5.3 OpenAI Environments

As of now we have concentrated on two environments with distinct hierarchical nature. The Towers of Hanoi problem could be solved by a recursive algorithm. The Four Rooms problem, on the other hand, had a clear subgoal structure which imposed to leave a specific room if the goal location was not in it. In order to further validate and generalize our proposed action grammar framework, we have provided two tests in OpenAI's gym environment (Brockman et al., 2016). The considered environments are the Taxi problem (Dietterich, 2000, p. 236) and the "large" (64 state) Frozen Lake environment.

The Taxi problem is an episodic RL problem placed within a 5 times 5 grid state space. At the beginning of an episode the state of the agent is randomly initiated. A target object (passenger) is randomly placed in one of four landmark positions. The agent then has to pick up the target and deliver it to a randomly sampled goal location. There are six primitive actions: Movement in all four directions and a "pickup" and "putdown" action. A successful completion of the episode is rewarded with 20 units while each other transition is negatively conditioned with a reward of -1. Finally, an illegal pickup or putdown attempt is punished by a -10 reward. Dietterich (2000, p. 243) did provide a manual hierarchical solution to this problem with the help of the MAXQ task graph. In the following we will investigate how our approach is able to solve this problem.

The Frozen Lake environment, on the other hand, does not have such a clear hierarchical structure. Instead, there are four different types of states: Start, frozen, hole and a goal state. The agent is initiated in a deterministic start state and has to reach the goal state to end the episode. He maneuvers in a 8 by 8 grid and if he reaches a hole state his state is reset to the starting state. A frozen state, on the other hand, leads to a completely random transition irregardless of the selected action. Again, he aims to reach the goal as fast as possible.

In all experiments outlined below and if not otherwise stated, we choose the following hyperparameters:

OpenAI Hyperparameters:
1. Exploration parameter: $\epsilon = 0.1$
2. Discount factor: $\gamma = 0.95$
3. Learning rate: $\alpha = 0.8$
4. Number of traces on which we infer a CFG grammar: $N_g = 10$
5. Number of traces on which we infer a PAG grammar: $N_g = 1000$ (imitation) and $N_g = 100$ (online)
6. Number of initial Q-Learning iterations: $N_{init} = 50$
7. Number of episodes between grammar updates: $N_a = 195$
8. Number of grammar updates: 10 (at episodes $\{50, 245, 440, 635, 830, 1025, 1220, 1415, 1610, 1805\}$)
9. k-Sequitur hyperparameter: $\theta = k = 2$
10. Maximal surprisal (option termination criterion): $\xi = 2$ (after distribution has been properly normalized)
11. Maximum steps of PAG option execution: 10

For both environments we provide learning results for imitation and online learning. Figures 5.21 and 5.22 display the learning results for the Taxi environment while figure 5.23 and 5.24 does the same for the Frozen Lake environment. The plots are structured as follows: The upper row shows the average amount of steps taken until the goal state is reached (lower is better) over the course of the learning episodes. The second row displays the average reward accumulated (higher is better). The first column, on the other hand, shows learning performance for online inferred macros. After an initial set of 50 traces we rollout 10 traces and infer a macro set. After 195 episodes we do the same again. The overlapping bar charts in the top row show how many macros were on average identified. We implement both SMDP-Q(λ)-Learning as well as value transfer (see section 4.2). The bar charts in the bottom row, on the other hand, display the mean length of such extracted macros.

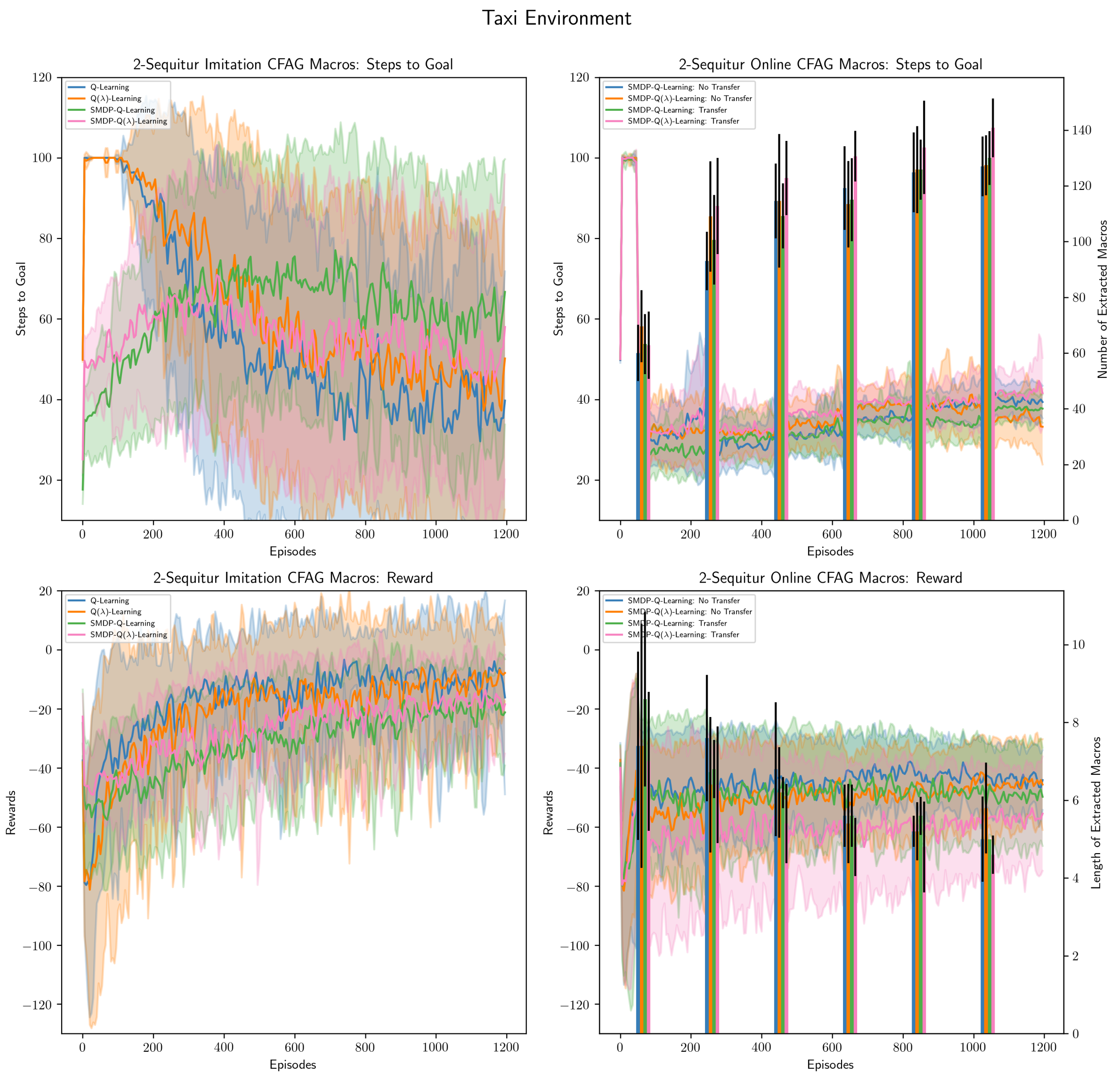


Figure 5.21: OpenAI Taxi Environment: Online Reinforcement Learning with CFAG Macros

For the Taxi environment, we observe that online inferred macros perform even better than macros obtained from 10 traces of expert behavior. This is astonishing and might be related to the fact that the agent might first have to obtain a rough value estimate of the primitive actions before he is able to leverage the macro-actions. Furthermore, we observe that during the course of learning the amount of macros that are inferred increases while the length of the macros decreases. This implies a form of low-level specialization and generalization.

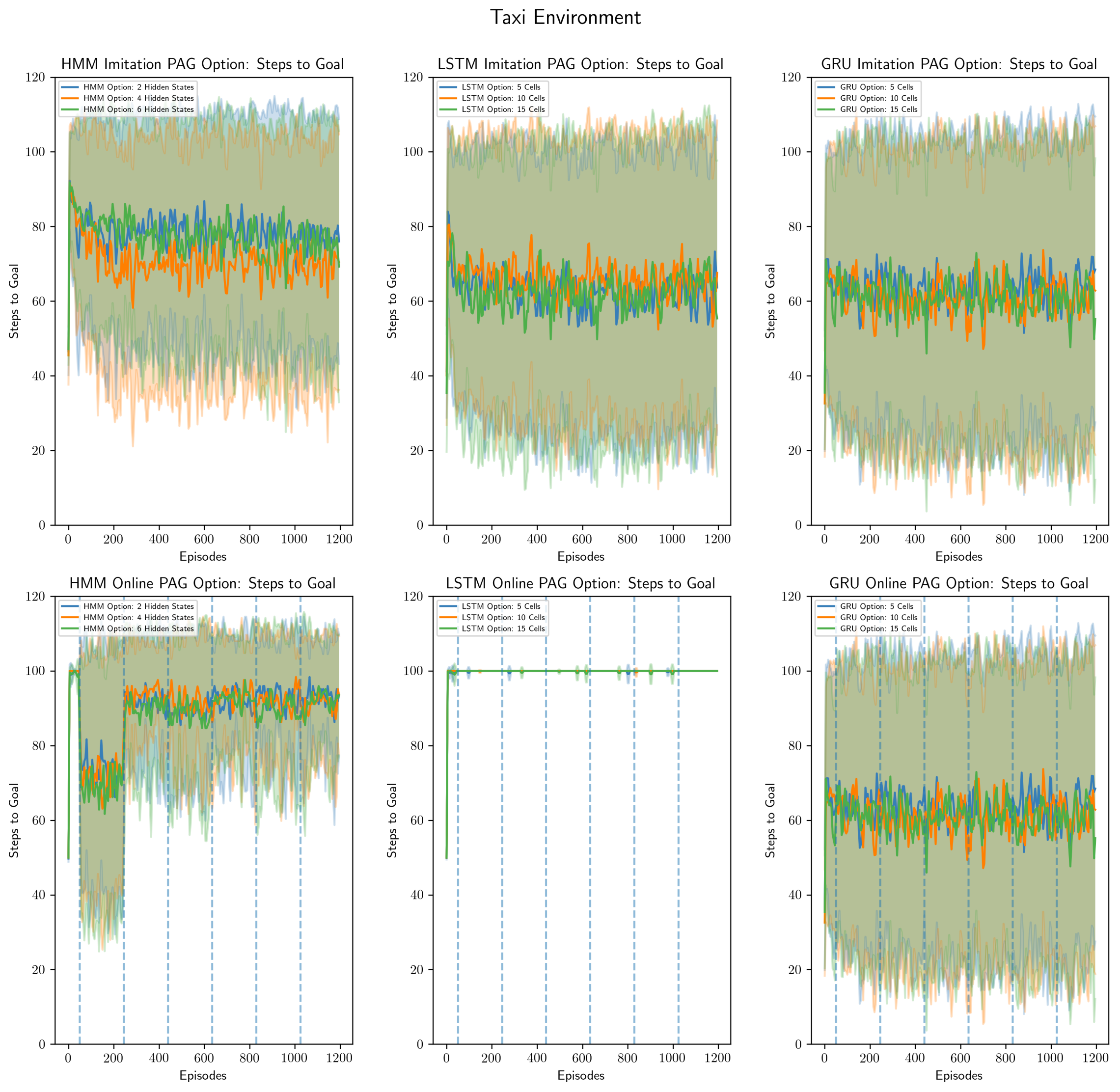


Figure 5.22: OpenAI Taxi Environment: Online Reinforcement Learning with PAG Options

The PAG options (figure 5.22) do not work well in either the imitation learning (trained on 1000 expert traces) or the online reinforcement learning (trained on 100 on-policy rollouts). We are not sure why this is the case and can only assume that it is due to the surprisal-based termination constraint. Another possible explanation might be that the grammatical structure provided by the HMMs or RNNs is just not suited for environments in which goal, starting and subgoal positions are randomly initiated. Macro-actions, on the other hand, might be superior due to their state invariance.

Again, in the Frozen Lake environment macro-actions clearly outperform simple Q-Learning baselines in both online and imitation learning (see figure 5.23). We want to highlight the observation that the first grammar update leads to solving the problem. We still extract many macros due to the fact that we concatenate 10 traces and run CFG inference on the concatenated traces (discarding semantically meaningless productions). The length of the macros and the number of extracted macros stays constant throughout learning which indicates an early 'grammar convergence'.

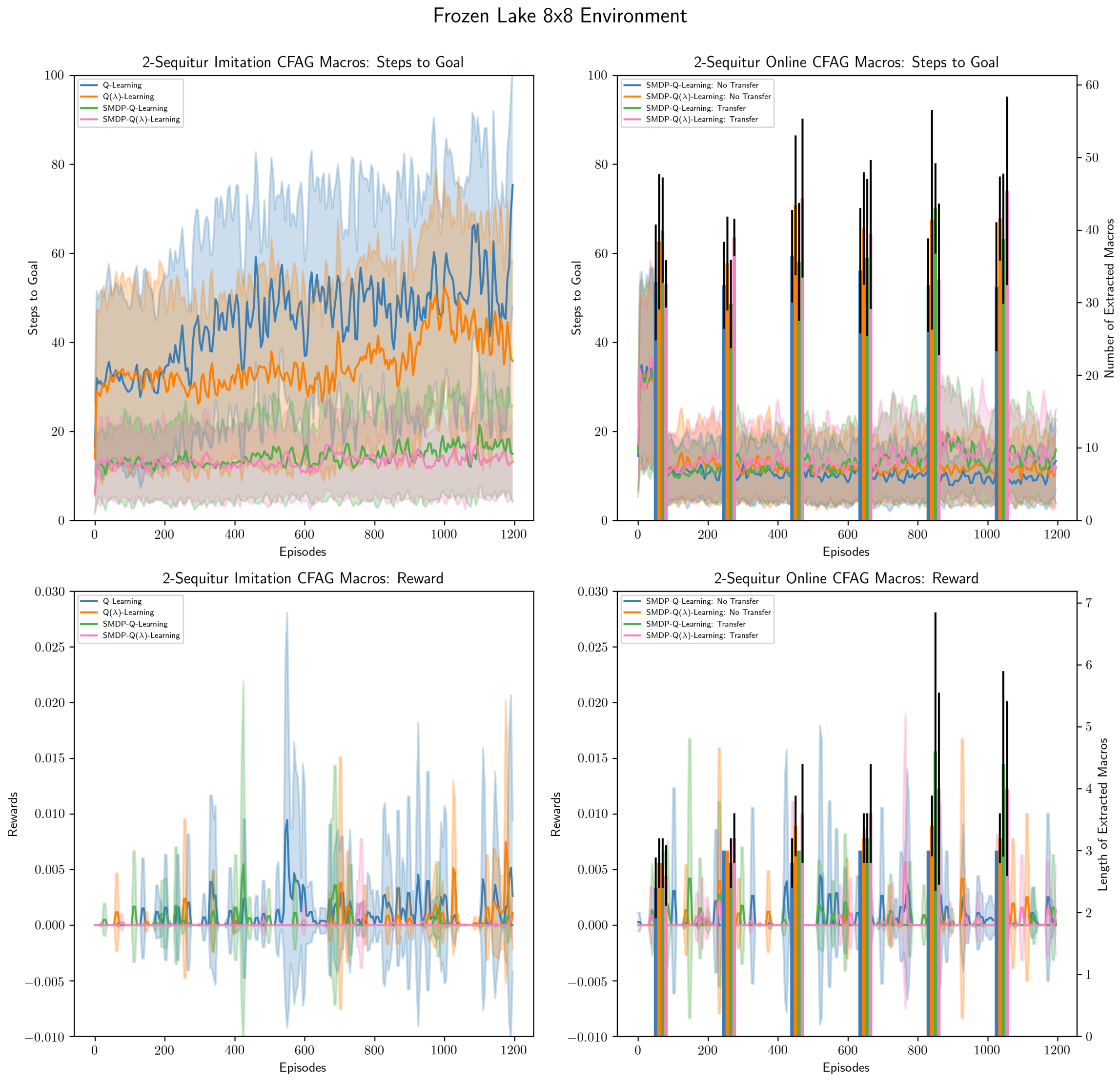


Figure 5.23: OpenAI FrozenLake 8x8 Environment: Online Reinforcement Learning with CFAG Macros

Online reinforcement learning with options only seems to improve learning for the 2 hidden state HMM as well as the 5 and 10 CRU cells-based option. Again, learning a stochastic grammar on top of expert traces does not provide better results than directly learning a grammar from on-policy rollouts.

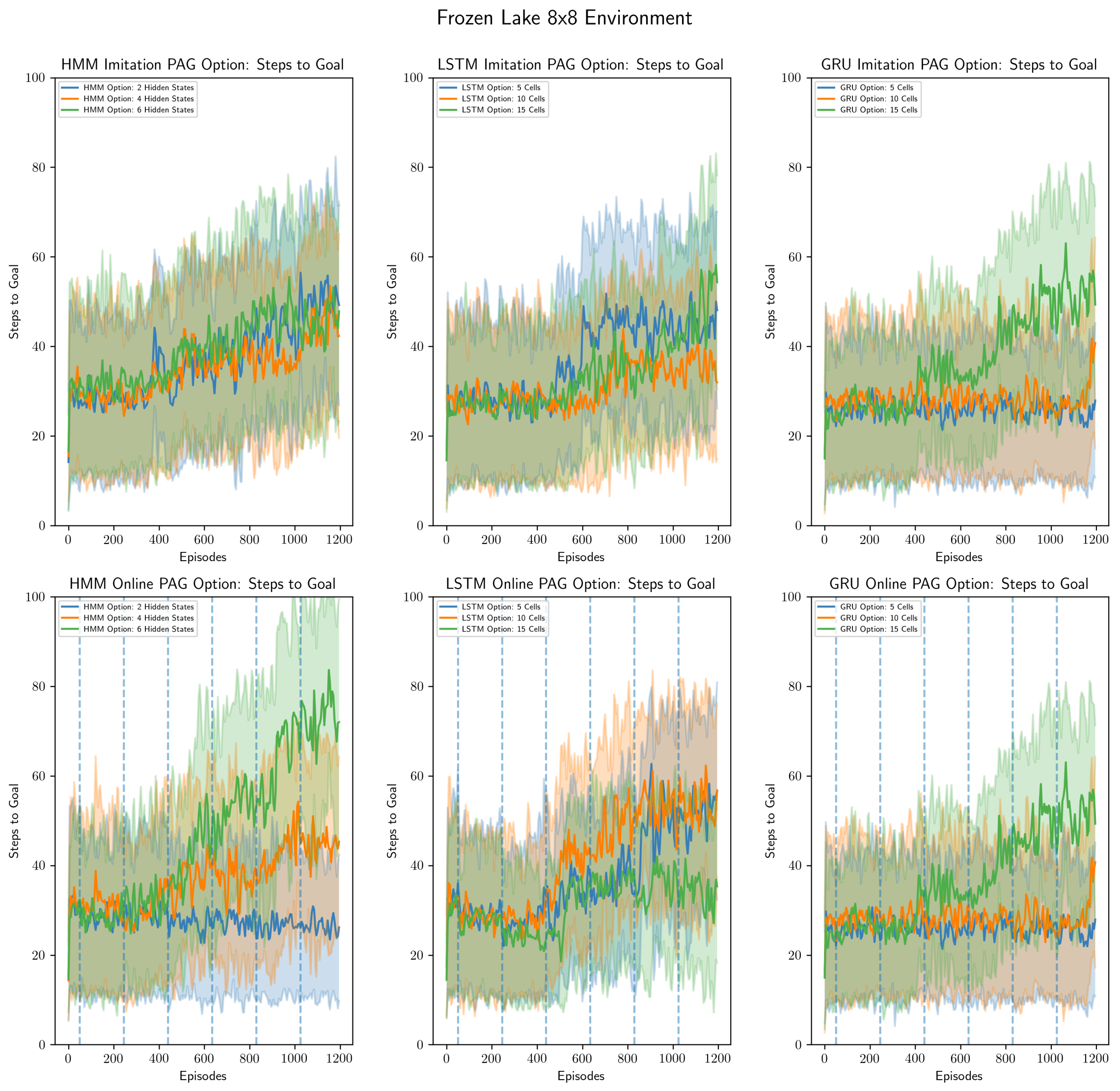


Figure 5.24: OpenAI FrozenLake 8x8 Environment: Online Reinforcement Learning with PAG Options

In conclusion, we find that macro-action discovery based on Sequitur CFG inference provides a powerful solution that is robust to the choice of environment. Furthermore, the online approach and the corresponding grammar development can yield interesting insights into the degree of required skill specialization. The performance of PAG options, on the other hand, seem to be very dependent on the chosen environment. We remain skeptical of this approach. One further has to analyze the role of the learning parameter ξ.

Chapter 6

Conclusion and Future Work

Motivated by a parallelism between the hierarchical generating processes of language and motion, we have derived multiple algorithmic approaches which exploit powerful grammatical inference frameworks to identify temporally-extended actions.
At the center of this analysis was the formal notion of Semi-Markov Decision Processes and their capability to model stochastic waiting times between decisions. Macro-actions and the options framework induced probability distributions over such transition periods. By sensibly defining temporally-extended actions and abstracting away unnecessary decision points, one is able to overcome the curse of dimensionality. Learning volatility is decreased by efficient exploration that incorporates domain knowledge. In order to overcome the necessity to manually articulate this knowledge, we proposed to turn to grammatical inference. By inferring the hierarchical structure of agent-environment interactions, we were able to fully automate the Hierarchical Reinforcement Learning pipeline.

Under the assumption that language and motor control can be formalized in one overarching computational setup, we outlined two specific approaches to action grammars: Context-free action grammars and probabilistic action grammars. Macro-actions as well as options were shown to be derivable from context-free grammars and their production rules. By training a grammar on sequences of experiences, the Hierarchical Reinforcement Learning agent was able to extract hierarchical information from state-action transitions. The grammar naturally defined a temporal hierarchy over actions which can be transformed into macro-actions by flattening the production rules. A CFG-based option set, on the other hand, was constructed by matching encoded action sequences with the state sequence. Furthermore, we also demonstrated how one is able to sample macro-actions from PCFGs and define options based on recurrent models of language parsing. Both the sampling process and option termination where driven by a syntactic surprisal measure. In the options framework we argued that surprisal expresses the need for control. The agent chooses to follow an intra-option policy until a surprising state transition occurs. At that point control is given back to the agent, so that she can deliberately take the next action. Sampling macro-actions, on the other hand, proved difficult due to the additional sampling variance. We remain skeptical of this approach. In order to validate our proposed framework, we tested both approaches for an imitation learning as well as an online RL task in multiple environments. Our observations can be summarized as follows:

- The CFAG approach to macro-actions extraction from flat production rules performs very well in both imitation and online learning. The agent can easily generalize from the inferred hierarchical structure and is able to increase the action learning speed drastically. Especially, the first grammars updates prove to be powerful in reducing the dimensionality of the problem.

- Alternating between grammar updates and learning action values is an effective way of both online learning of an optimal grammar as well as an optimal policy. The first grammar extraction and action space augmentation has the largest significant effect in the further learning procedure of the agent.
- PAG options when trained on expert traces are able to capture recurrent behavioral strategies which are comparable with manually crafted options. We found evidence for a connection between the complexity hyperparameter of the grammar and the hierarchical structure of the environment.

In summary and chronological order our contributions are the following:

1. We formalized an eligibility traces extension to SMDP-Q-Learning.
2. We propose a production rule-based construction of macro-actions which performs well in online and imitation learning tasks.
3. We outline a CFG-based construction of options as well as HMM and RNN-based versions which are able to automatically infer options given expert traces.
4. We show how syntactic surprisal is useful in both comparing different PAG grammars for HRL and in providing an automated termination condition for options.

In future work we are interested in testing and extending our approach to both visual (pixel-based state representations, e.g. ATARI games) and physical (joint and velocity-bases state representations, e.g. MoJuCo) domains. Formal grammars are especially useful for languages with large terminal vocabulary. So far we have only experimented with small action spaces and single agents. Pastra and Aloimonos (2012, p. 113) note that social interactions of more than one agent can also be formulated within the notion of tool use. Hence, we are interested in possible applications to multi-agent RL and testing the scalability of our approach to real-life domains.
Another important question that we have not been able to fully address, is how to learn initiation sets for options. Most common approaches allow options to be executed starting from every state in the state space. Since not all sub-routines can be efficiently executed from every state, such a crude assumption can slow down learning. In section 4.3 we saw that learning a grammar on sequences of state transitions might provide a good first step. More experiments are needed.
Furthermore, our approach has only attempted to merge grammatical inference with two HRL algorithms. There remain many other promising frameworks such as the Hierarchies of Abstract Machines (HAMs, Parr (1998); Parr and Russell (1998)). HAMs define a hierarchy over finite state machines. This could naturally lead itself to automated identification via Hierarchical Hidden Markov Models (Fine et al., 1998).
Future work also has to further analyze the development of the inferred grammar throughout the learning process. Edit distances such as the Levenshtein and Jaro-Winkler distance provide two measures of string similarity which might be used to efficiently monitor the development of the inferred flat productions compared with the optimal grammar.
Ultimately, we envision a form of dictionary of action which provides an expandable library of skills for Hierarchical Reinforcement Learning agents which act in diverse naturalistic

environments. This could provide a mayor contribution to a key endeavor in general artificial intelligence: life-long learning.

Bibliography

ALOIMONOS, Y. (2008): "HAL: human activity language," *Journal of Vision*, 8, 1050–1050. pages 18

ALOIMONOS, Y., G. GUERRA-FILHO, AND A. OGALE (2010): "The language of action: a new tool for human-centric interfaces," in *Human-Centric Interfaces for Ambient Intelligence*, Elsevier, 95–131. pages 18

BACON, P.-L., J. HARB, AND D. PRECUP (2017): "The Option-Critic Architecture," in *AAAI*, 1726–1734. pages 1, 14, 15, 36, 90

BAKKER, B. AND J. SCHMIDHUBER (2004): "Hierarchical reinforcement learning based on subgoal discovery and subpolicy specialization," in *Proc. of the 8-th Conf. on Intelligent Autonomous Systems*, 438–445. pages 14, 16

BARTO, A. G. AND S. MAHADEVAN (2003): "Recent advances in hierarchical reinforcement learning," *Discrete Event Dynamic Systems*, 13, 341–379. pages 1, 5, 9, 10

BAUM, L. E., T. PETRIE, G. SOULES, AND N. WEISS (1970): "A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains," *The annals of mathematical statistics*, 41, 164–171. pages 15

BERTSEKAS, D. P. AND J. N. TSITSIKLIS (1995): "Neuro-dynamic programming: an overview," in *Decision and Control, 1995., Proceedings of the 34th IEEE Conference on*, IEEE, vol. 1, 560–564. pages 1

BRADTKE, S. J. AND M. O. DUFF (1995): "Reinforcement learning methods for continuous-time Markov decision problems," in *Advances in neural information processing systems*, 393–400. pages 3, 4, 6, 7

BRENNAN, J. R., E. P. STABLER, S. E. VAN WAGENEN, W.-M. LUH, AND J. T. HALE (2016): "Abstract linguistic structure correlates with temporal activity during naturalistic comprehension," *Brain and language*, 157, 81–94. pages 2, 18

BROCKMAN, G., V. CHEUNG, L. PETTERSSON, J. SCHNEIDER, J. SCHULMAN, J. TANG, AND W. ZAREMBA (2016): "Openai gym," *arXiv preprint arXiv:1606.01540*. pages 45, 71

CHARIKAR, M., E. LEHMAN, D. LIU, R. PANIGRAHY, M. PRABHAKARAN, A. SAHAI, AND A. SHELAT (2005): "The smallest grammar problem," *IEEE Transactions on Information Theory*, 51, 2554–2576. pages 21

CHOMSKY, N. (1959a): "A note on phrase structure grammars," *Information and control*, 2, 393–395. pages 20

——— (1959b): "On certain formal properties of grammars," *Information and control*, 2, 137–167. pages 20

——— (1995): *The minimalist program*, MIT press. pages 18

CHUNG, J., C. GULCEHRE, K. CHO, AND Y. BENGIO (2015): "Gated feedback recurrent neural networks," in *International Conference on Machine Learning*, 2067–2075. pages 42

CO-REYES, J., Y. LIU, A. GUPTA, B. EYSENBACH, P. ABBEEL, AND S. LEVINE (2018): "Self-Consistent Trajectory Autoencoder: Hierarchical Reinforcement Learning with Trajectory Embeddings," in *Proceedings of the 35th International Conference on Machine Learning*, ed. by J. Dy and A. Krause, Stockholmsmässan, Stockholm Sweden: PMLR, vol. 80 of *Proceedings of Machine Learning Research*, 1009–1018. pages 16

DANIEL, C., H. VAN HOOF, J. PETERS, AND G. NEUMANN (2016): "Probabilistic inference for determining options in reinforcement learning," *Machine Learning*, 104, 337–357. pages 14, 15

DIETTERICH, T. G. (2000): "Hierarchical reinforcement learning with the MAXQ value function decomposition," *J. Artif. Intell. Res.(JAIR)*, 13, 227–303. pages 1, 10, 71, 89

DING, N., L. MELLONI, X. TIAN, AND D. POEPPEL (2017): "Rule-based and word-level statistics-based processing of language: insights from neuroscience," *Language, Cognition and Neuroscience*, 32, 570–575. pages 2, 18

FADIGA, L., L. CRAIGHERO, AND A. D'AUSILIO (2009): "Broca's area in language, action, and music," *Annals of the New York Academy of Sciences*, 1169, 448–458. pages 18

FAZIO, P., A. CANTAGALLO, L. CRAIGHERO, A. D'AUSILIO, A. C. ROY, T. POZZO, F. CALZOLARI, E. GRANIERI, AND L. FADIGA (2009): "Encoding of human action in Broca's area," *Brain*, 132, 1980–1988. pages 18

FINE, S., Y. SINGER, AND N. TISHBY (1998): "The hierarchical hidden Markov model: Analysis and applications," *Machine learning*, 32, 41–62. pages 77

FLORENSA, C., Y. DUAN, AND P. ABBEEL (2017): "Stochastic neural networks for hierarchical reinforcement learning," *arXiv preprint arXiv:1704.03012*. pages 14, 16

FRANK, M. C., J. A. SLEMMER, G. F. MARCUS, AND S. P. JOHNSON (2009): "Information from multiple modalities helps 5-month-olds learn abstract rules," *Developmental science*, 12, 504–509. pages 2

FRANK, S. L. AND M. H. CHRISTIANSEN (2018): "Hierarchical and sequential processing of language," *Language, Cognition and Neuroscience*, 0, 1–6. pages 2

FRANS, K., J. HO, X. CHEN, P. ABBEEL, AND J. SCHULMAN (2017): "Meta learning shared hierarchies," *arXiv preprint arXiv:1710.09767*. pages 16

FRIEDMAN, J., T. HASTIE, AND R. TIBSHIRANI (2001): *The elements of statistical learning*, vol. 1, Springer series in statistics New York, NY, USA:. pages 43

HALE, J. (2001): "A probabilistic Earley parser as a psycholinguistic model," in *Proceedings of the second meeting of the North American Chapter of the Association for Computational Linguistics on Language technologies*, Association for Computational Linguistics, 1–8. pages 3, 23, 24

——— (2003): "The information conveyed by words in sentences," *Journal of Psycholinguistic Research*, 32, 101–123. pages 23, 24

——— (2006): "Uncertainty about the rest of the sentence," *Cognitive Science*, 30, 643–672. pages 23

HALE, J. T. (2014): *Automaton theories of human sentence comprehension*. pages 23, 24

HAUSKRECHT, M., N. MEULEAU, L. P. KAELBLING, T. DEAN, AND C. BOUTILIER (1998): "Hierarchical solution of Markov decision processes using macro-actions," in *Proceedings* *of the Fourteenth conference on Uncertainty in artificial intelligence*, Morgan Kaufmann Publishers Inc., 220–229. pages 9

HENGST, B. (2002): "Discovering hierarchy in reinforcement learning with HEXQ," in *ICML*, vol. 2, 243–250. pages 1, 12, 14, 90

HOCHREITER, S. AND J. SCHMIDHUBER (1997): "Long short-term memory," *Neural computation*, 9, 1735–1780. pages 42

KULKARNI, T. D., K. NARASIMHAN, A. SAEEDI, AND J. TENENBAUM (2016a): "Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation," in *Advances in neural information processing systems*, 3675–3683. pages 1, 16

KULKARNI, T. D., A. SAEEDI, S. GAUTAM, AND S. J. GERSHMAN (2016b): "Deep successor reinforcement learning," *arXiv preprint arXiv:1606.02396*. pages 1, 16

LANGE, R. T. (2018a): "Independent Study Option Presentation: Hierarchical Reinforcement Learning and Action Grammars," *Imperial College London*. pages i, 2, 89

——— (2018b): "Independent Study Option Report: Hierarchical Reinforcement Learning and Action Grammars," *Imperial College London*. pages i, 5, 6, 10, 89

LARI, K. AND S. J. YOUNG (1990): "The estimation of stochastic context-free grammars using the inside-outside algorithm," *Computer speech & language*, 4, 35–56. pages 23

LASHLEY, K. S. (1951): *The problem of serial order in behavior*, vol. 21, Bobbs-Merrill. pages 18

LEVELT, W. J. (2008): *An introduction to the theory of formal languages and automata*, John Benjamins Publishing. pages 20, 21

LEVY, R. (2008): "Expectation-based syntactic comprehension," *Cognition*, 106, 1126–1177. pages 3, 23, 24

MANNOR, S., I. MENACHE, A. HOZE, AND U. KLEIN (2004): "Dynamic abstraction in reinforcement learning via clustering," in *Proceedings of the twenty-first international conference on Machine learning*, ACM, 71. pages 12, 14

MARCUS, G. F., K. J. FERNANDES, AND S. P. JOHNSON (2007): "Infant rule learning facilitated by speech," *Psychological science*, 18, 387–391. pages 2

MARCUS, G. F., S. VIJAYAN, S. B. RAO, AND P. M. VISHTON (1999): "Rule learning by seven-month-old infants," *Science*, 283, 77–80. pages 2

MCGOVERN, A. AND A. G. BARTO (2001): "Automatic discovery of subgoals in reinforcement learning using diverse density," in *ICML*, vol. 1, 361–368. pages 1, 12, 14

MCGOVERN, A. AND R. S. SUTTON (1998): "Macro-actions in reinforcement learning: An empirical analysis," Tech. rep., Technical Report 98-70, University of Massachusetts, Department of Computer Science. pages 4, 7, 8

MCGOVERN, A., R. S. SUTTON, AND A. H. FAGG (1997): "Roles of macro-actions in accelerating reinforcement learning," in *Grace Hopper celebration of women in computing*, vol. 1317. pages 1, 4, 7, 8

MENACHE, I., S. MANNOR, AND N. SHIMKIN (2002): "Q-cut—dynamic discovery of sub-goals in reinforcement learning," in *European Conference on Machine Learning*, Springer, 295–306. pages 1, 12, 14

MNIH, V., A. P. BADIA, M. MIRZA, A. GRAVES, T. LILLICRAP, T. HARLEY, D. SILVER, AND

K. KAVUKCUOGLU (2016): "Asynchronous methods for deep reinforcement learning," in *International Conference on Machine Learning*, 1928–1937. pages 16

MNIH, V., K. KAVUKCUOGLU, D. SILVER, A. GRAVES, I. ANTONOGLOU, D. WIERSTRA, AND M. RIEDMILLER (2013): "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*. pages 16

MNIH, V., K. KAVUKCUOGLU, D. SILVER, A. A. RUSU, J. VENESS, M. G. BELLEMARE, A. GRAVES, M. RIEDMILLER, A. K. FIDJELAND, G. OSTROVSKI, ET AL. (2015): "Human-level control through deep reinforcement learning," *Nature*, 518, 529. pages 16

NELSON, M. J., I. EL KAROUI, K. GIBER, X. YANG, L. COHEN, H. KOOPMAN, S. S. CASH, L. NACCACHE, J. T. HALE, C. PALLIER, ET AL. (2017): "Neurophysiological dynamics of phrase-structure building during sentence processing," *Proceedings of the National Academy of Sciences*, 201701590. pages 2, 18

NEVILL-MANNING, C. G. AND I. H. WITTEN (1997): "Identifying Hierarchical Structure in Sequences: A linear-time algorithm," *CoRR*, cs.AI/9709102. pages 3, 21

PARR, R. AND S. J. RUSSELL (1998): "Reinforcement learning with hierarchies of machines," in *Advances in neural information processing systems*, 1043–1049. pages 1, 77, 89

PARR, R. E. (1998): *Hierarchical control and learning for Markov decision processes*, University of California, Berkeley Berkeley, CA. pages 7, 77

PASTRA, K. AND Y. ALOIMONOS (2012): "The minimalist grammar of action," *Philosophical Transactions of the Royal Society of London B: Biological Sciences*, 367, 103–117. pages i, 2, 18, 19, 20, 77, 90

PETKOVI, M. (2009): *Famous puzzles of great mathematicians*, American Mathematical Soc. pages 47

RUMELHART, D. E., G. E. HINTON, AND R. J. WILLIAMS (1986): "Learning representations by back-propagating errors," *nature*, 323, 533. pages 16

SCHOENHENSE, B., A. HAMPSHIRE, AND A. FAISAL (2017): "Data-efficient inference of hierarchical structure in sequential data by information-greedy grammar inference," *Under review for the Conference on Neural Information Processing Systems (NIPS)*. pages 3, 22

SCHULMAN, J., F. WOLSKI, P. DHARIWAL, A. RADFORD, AND O. KLIMOV (2017): "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*. pages 16

SIMSEK, O., A. P. WOLFE, AND A. G. BARTO (2004): "Local graph partitioning as a basis for generating temporally-extended actions in reinforcement learning," in *AAAI Workshop Proceedings*. pages 12, 14

SIYARI, P., B. DILKINA, AND C. DOVROLIS (2016): "Lexis: An optimization framework for discovering the hierarchical structure of sequential data," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM, 1185–1194. pages 3, 21, 22

SIYARI, P. AND M. GALLÉ (2016): "The Generalized Smallest Grammar Problem," in *International Conference on Grammatical Inference*, 79–92. pages 20, 21

SMITH, M., H. VAN HOOF, AND J. PINEAU (2018): "An Inference-Based Policy Gradient Method for Learning Options," in *Proceedings of the 35th International Conference on Machine Learning*, 4710–4719. pages 9, 14, 15, 16, 41

STOLCKE, A. (1994): "Bayesian learning of probabilistic language models," Ph.D. thesis, University of California, Berkeley. pages 23

STOLLE, M. AND D. PRECUP (2002): “Learning options in reinforcement learning,” in *International Symposium on abstraction, reformulation, and approximation*, Springer, 212–223. pages 12, 14, 15

STOUT, D., T. CHAMINADE, A. THOMIK, J. APEL, AND A. A. FAISAL (2018): “Grammars of action in human behavior and evolution,” *bioRxiv*, 281543. pages 2, 18, 22, 90

SUTTON, R. S. AND A. G. BARTO (1998): *Introduction to reinforcement learning*, vol. 135, MIT press Cambridge. pages 1, 5, 8

SUTTON, R. S., D. PRECUP, AND S. SINGH (1999): “Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning,” *Artificial intelligence*, 112, 181–211. pages iv, 1, 4, 6, 7, 8, 9, 10, 11, 12, 57, 90

TAKAHASHI, Y., G. SCHOENBAUM, AND Y. NIV (2008): “Silencing the critics: understanding the effects of cocaine sensitization on dorsolateral and ventral striatum in the context of an actor/critic model,” *Frontiers in neuroscience*, 2, 14. pages 27

VEZHNEVETS, A., V. MNIH, S. OSINDERO, A. GRAVES, O. VINYALS, J. AGAPIOU, ET AL. (2016): “Strategic attentive writer for learning macro-actions,” in *Advances in neural information processing systems*, 3486–3494. pages 1, 16

VEZHNEVETS, A. S., S. OSINDERO, T. SCHAUL, N. HEESS, M. JADERBERG, D. SILVER, AND K. KAVUKCUOGLU (2017): “Feudal networks for hierarchical reinforcement learning,” *arXiv preprint arXiv:1703.01161*. pages 1, 14, 16

WATKINS, C. J. AND P. DAYAN (1992): “Q-learning,” *Machine learning*, 8, 279–292. pages 4, 6, 8

Appendices

Appendix A

Experimental Details and User Guide

We provide 3 directories which contain the code used to derive the results of chapter 5 in the report. Each one of them is self-contained and has to be run independently:

1. **hrl_hanoi**: Reproduces results of chapter 5.1 and macro-action discovery for the Towers of Hanoi environment.
2. **hrl_rooms**: Reproduces results of chapter 5.2 and option discovery for the Four Rooms environment.
3. **hrl_openai**: Reproduces results of chapter 5.3 and macro-action/option discovery for several OpenAI environments.

In general we average the results over 5 agents and obtain performance statistics by rolling out a set of on-policy trajectories after a set of learning episodes. We parallelize the agents (not in the rooms environment) so that learning speed can be increased when having access to multiple CPUs.
In order to get started we recommend to review our OpenAI implementation since it is the easiest to generalize from. As of now the framework is implemented using model-free and tabular reinforcement learning approaches. A few general comments:

- Everything is implemented in Python 2.7 and on UNIX-based operating systems.
- Required packages are listed in requirements.txt files of specific repositories.
- Grammar induction algorithms (Sequitur/Lexis) need to be compiled (*make*) before you can run them within the learning environments. I do not take any credit for such implementations and you can find the original repositories here:
  - Sequitur:`https://github.com/craignm/sequitur`
  - Lexis: `https://github.com/payamsiyari/Lexis`
- The PAG options are implemented using RNNs which we train in Keras with Tensorflow backend. In order to efficiently train them, please activate GPU support (if you have access):

```
export LD_LIBRARY_PATH=LD_LIBRARY_PATH:/vol/cuda/9.0.176/lib64
export CUDA_VISIBLE_DEVICES=0
```

For all three directories please follow the general steps below (* $\in \{hanoi, rooms, openai\}$):
1. Change directory to specific repository

```
cd hrl_*
```

2. Create a virtualenv (optional but recommended) and activate it

```
sudo apt-get install python-virtualenv
virtualenv -p python AG
source AG/bin/activate && pip install --upgrade pip
```

3. Run make to get Lexis running:

```
cd ~/hrl_*/utils/Lexis/repeats1 && make
```

4. Run make to get Sequitur running:

```
cd ~/hrl_*/utils/sequitur && make
```

5. Go back to main folder and install all dependencies:

```
cd ~/hrl_* && pip install -r requirements.txt
```

6. Afterwards, you can now run the following commands to start simulating the different agents.

Towers of Hanoi (Chapter 5.1)

The main environment setup for Q-Learning is partially taken from `https://github.com/khpeek/Q-learning-Hanoi`. We provide results for the following (H)RL agents:

1. Q-Learning
2. Imitation Learning with Lag 0 or 1 for Sequitur and Lexis
3. Online HRL for Sequitur/Lexis/HMM with and without value transfer

To reproduce the agents please run the following commands from the command line:

```
python run_in_parallel --q

python run_in_parallel --imitation_seq
python run_in_parallel --imitation_lexis
python run_in_parallel --imitation_seq_lag
python run_in_parallel --imitation_lexis_lag

python run_in_parallel --online_seq
python run_in_parallel --online_seq_transfer
python run_in_parallel --online_lexis
python run_in_parallel --online_lexis_transfer
python run_in_parallel --online_hmm
python run_in_parallel --online_hmm_transfer
```

Four Rooms Problem (Chapter 5.2)

The main environment setup for Q-Learning is partially taken from `https://github.com/tmomose/sutton_rooms`. We provide results for the following (H)RL agents:

1. Converged Q-Learner (to get expert traces - run first!)
2. Simple Learning Baselines: Q-(λ)-Learning, Macro-Q(λ)-Learning, Hallway Options
3. Imitation Learning Results: CFAG, PCFAG Macros/Options
4. Online Hierarchical Reinforcement Learning Results: CFAG, PCFAG Options

To reproduce the agents please run the following commands from the command line:

```
python run_learning --expert

python run_learning --q
python run_learning --macro
python run_learning --hallway

python run_learning --sequitur
python run_learning --lexis

python run_learning --hmm
python run_learning --lstm
python run_learning --gru

python cfg_online_learning --seq2
python cfg_online_learning --seq4
python cfg_online_learning --seq_schedule

python scfg_online_learning --hmm
python scfg_online_learning --lstm
python scfg_online_learning --gru
```

OpenAI Environments (Chapter 5.3)

We provide results for the following (H)RL agents:

1. Q-Learning, Q(λ)-Learning
2. Imitation Learning Results: CFAG Macros
3. Online Hierarchical Reinforcement Learning Results: PCFAG Options

To reproduce the agents for the specific environments, please run the following commands from the command line:

```
python main.py --env Taxi-v2 -l_type q_learning
python main.py --env Taxi-v2 -l_type macro_q_imitation
python main.py --env Taxi-v2 -l_type macro_q_online

python main.py --env Taxi-v2 -l_type options_imitation_hmm
python main.py --env Taxi-v2 -l_type options_imitation_lstm
python main.py --env Taxi-v2 -l_type options_imitation_gru

python main.py --env Taxi-v2 -l_type options_online_hmm
python main.py --env Taxi-v2 -l_type options_online_lstm
python main.py --env Taxi-v2 -l_type options_online_gru

python main.py --env FrozenLake8x8-v0 -l_type q_learning
python main.py --env FrozenLake8x8-v0 -l_type macro_q_imitation
python main.py --env FrozenLake8x8-v0 -l_type macro_q_online

python main.py --env FrozenLake8x8-v0 -l_type options_imitation_hmm
python main.py --env FrozenLake8x8-v0 -l_type options_imitation_lstm
python main.py --env FrozenLake8x8-v0 -l_type options_imitation_gru

python main.py --env FrozenLake8x8-v0 -l_type options_online_hmm
python main.py --env FrozenLake8x8-v0 -l_type options_online_lstm
python main.py --env FrozenLake8x8-v0 -l_type options_online_gru

python plotting.py --env Taxi-v2 --sm 2 -s
python plotting.py --env FrozenLake8x8 --sm 2 -s
```

The results of the learning procedures will be saved in the corresponding /results directory. In order to visualize the results please refer to the run_plotting.py file.

Appendix B

Summary of ISO Report and Relationship to this Project

This project was inspired by an Independent Study Option (Lange, 2018a,b) conducted under the supervision of Professor Aldo Faisal. The ISO was motivated by a desire to find a connection between Hierarchical Reinforcement Learning and the field of computational linguistics. The main result of the ISO report was the revelation that HRL, while being a very powerful framework to learn sequential task achievements, is lacking a satisfying solution to the automatic discovery of temporally-extended actions. We reviewed the history and current state of Hierarchical Reinforcement Learning which included classical approaches such as the MAXQ value function decomposition (Dietterich, 2000) and HAMs (Parr and Russell, 1998) as well as more recent developments in Deep HRL. We refer to figure 1.1 for a concise summary of our results. Finally, we motivated the idea of applying Formal Grammars to Hierarchical Reinforcement Learning.

This project report continues where the ISO ended. We develop a solution to the sub-structure discovery problem by the means of grammatical inference. When appropriate we have quoted the previous work. At multiple points in time we have cited the original ISO report and its presentation throughout this report. Additionally, in this appendix as prescribed, we give a short summary of the ISO and its relationship to this project.

More specifically, the table below shows the individual sections of the ISO and indicates whether or not they inspired concepts that were further pursued in this project.

Independent Study Option Sections - Parallels	**Yes**	**No**
Section 1: Introduction		
Section 2: Markov Decision Processes and Semi-Markov Decision Processes		
Subsection 1: Markov Decision Processes	**X (ch: 2.1)**	
Subsection 2: Semi-Markov Decision Processes	**X (ch: 2.2)**	
Section 3: Hierarchical Reinforcement Learning		
Subsection 1: Early Approaches to the High Resolution Problem		**X**
Subsection 2: Options	**X (ch: 2.3.2)**	

Subsection 3: Hierarchies of Abstract Machines		**X**
Subsection 4: MAXQ Value Function Decomposition		**X**
Subsection 5: Recent Approaches	**X (ch: 2.4)**	
Section 4: Language Comprehension, Formal Grammars and Subroutine Extraction		
Subsection 1: The Cognitive Processes of Language Processing	**X (ch: 3.1)**	
Subsection 2: Formal Grammars and Grammatical Inference	**X (ch: 3.2 and 3.3)**	
Subsection 3: Action Grammars: Learning Options via Formal Grammars	**X (first idea for ch: 4.3)**	
Section 5: Conclusion		

Section 2 of the ISO report was reviewed and extended in chapter 2.1 and 2.2 of this report. They provide the necessary formal foundations for temporally-extended actions and cannot be excluded. Additionally, we added more intuition about stochastic waiting time distributions and the relationship to MDPs.

From section 3 of the ISO report we stuck with the options framework (Sutton et al., 1999). Options provide a strong HRL algorithm which we used to develop our Action Grammars framework in. In chapter 2.3.2 of this report we review the formalism. We provide a different point of view, implemented the Four Rooms environment and programmed learning with the Hallway options (see figure 2.3 and 2.5). Furthermore, a few papers (most notably Hengst (2002) and (Bacon et al., 2017)) which were reviewed in chapter 2.4 were also reviewed in section 3.5.1 of the ISO report.

In section 4.1 of the ISO we discussed recent neuroscientific findings that inspired our approach to identify hierarchical behavior with the help of formal grammars. Some of these motivating findings have been quoted throughout this text in order to make the content more vivid. Furthermore, section 3.1 is partially inspired by such results. Section 4.2 of the ISO, on the other hand reviewed context-free grammars and the Sequitur algorithms which are both addressed in chapter 3.2 and 3.3 of this report. Still, here we provide rigorous details and further extend the content to new domains. E.g. we cover probabilistic CFGs, inference algorithms such as G-Lexis and Hidden Markov Models and papers such as the ones by Stout et al. (2018) and Pastra and Aloimonos (2012). Finally, in section 4.3 of the ISO report we outlined a first idea of how to craft options from production rules. This approach is different and inferior to the one discussed in this project report (see section 4.3) and can be viewed as a starting point.

The contributions, implementations and reviews conducted in this project are substantial. First, we enriched our analysis of HRL algorithms by reviewing macro-actions. We discovered an algorithm that combines both macro-actions with eligibility traces and implemented it. Second, we implemented SMDP-Q(λ)-Learning and options with interruption or inter-option learning. We further reviewed the literature which is focussed on learning options and temporal abstractions. Chapters 4 and 5 (the main conceptual contributions) of this report remain completely independent of the previous ISO. All of our macro-action and option discovery algorithms are novel and their implementation was done during the time of the project.

Appendix C

Ethical and Legal Considerations

The continuing automation of processes which were originally executed by human workers is becoming more and more prevalent. This will have severe consequences on the way how we deal with our daily lives. In the short term unemployment might increase and will require significant rethinking of deep structures in society. Education for example will have to adapt to the needs of the 21st century. Many citizens who become unemployment have to be able to quickly obtain new skills. We are already starting to see a revolution in digital education and Massive Open Online Courses (MOOCs). In the long run many questions regarding electricity demand, sustainable computing and privacy preservation have to be answered. Machine Learning (ML) is the key to both developments. First, it provides the technical requirements to automated large scale processes in the industry. Second, it also changes the way how content will be taught. Automatic grading and adaption of curricula are only two examples. Developments in privacy preserving machine learning such as the OpenMined project piloted by Andrew Trask are important steps in the right direction. Algorithmic efforts such as homeomorphic encryption and federated learning provide the necessary tools to ensure that everyone is able to share his or her data without damaging themselves. Except for these general considerations regarding Machine Learning and automation we do not see any particular danger regarding our project.

This project has introduced a novel view to constructing sub-routines from expert demonstration or in a feedback/reinforcement signal modulated way. By leveraging computational linguistics, one is able to extracts hierarchical structures from observed behavior. We have shown that these structures contain significant meaning as subgoal achievements and help the agent to learn faster in the early stages. They are easy to interpret and contain semantic meaning. Furthermore, they are specialized to the situation the agent finds herself in. We conclude that Hierarchical Reinforcement Learning provides a valuable contribution to the long-term credit assignment problem as well as the question of interpretability. Furthermore, we highlight the necessity for discussion among all groups of society. At the moment a small subgroup of highly educated people are involved in the discussion. But these people are likely to be the last ones to be affected by the upcoming revolution in artificial intelligence. People who are less educated have to go through the most transformation and have to adapt to the fast changing needs of society. Hence, we must all come together and think about fundamental changes and how to shape our future.

Appendix D

Ethics Checklist

	Yes	No
Section 1: HUMAN EMBRYOS/FOETUSES		
Does your project involve Human Embryonic Stem Cells?		X
Does your project involve the use of human embryos?		X
Does your project involve the use of human foetal tissues / cells?		X
Section 2: HUMANS		
Does your project involve human participants?		X
Section 3: HUMAN CELLS / TISSUES		
Does your project involve human cells or tissues? (Other than from “Human Embryos/Foetuses” i.e. Section 1)?		X
Section 4: PROTECTION OF PERSONAL DATA		
Does your project involve personal data collection and/or processing?		X
Does it involve the collection and/or processing of sensitive personal data (e.g. health, sexual lifestyle, ethnicity, political opinion, religious or philosophical conviction)?		X
Does it involve processing of genetic information?		X
Does it involve tracking or observation of participants? It should be noted that this issue is not limited to surveillance or localization data. It also applies to Wan data such as IP address, MACs, cookies etc.		X
Does your project involve further processing of previously collected personal data (secondary use)? For example Does your project involve merging existing data sets?		X
Section 5: ANIMALS		
Does your project involve animals?		X
Section 6: DEVELOPING COUNTRIES		
Does your project involve developing countries?		X
If your project involves low and/or lower-middle income countries, are any benefit-sharing actions planned?		X

Could the situation in the country put the individuals taking part in the project at risk?		X
Section 7: ENVIRONMENTAL PROTECTION AND SAFETY		
Does your project involve the use of elements that may cause harm to the environment, animals or plants?		X
Does your project deal with endangered fauna and/or flora /protected areas?		X
Does your project involve the use of elements that may cause harm to humans, including project staff?		X
Does your project involve other harmful materials or equipment, e.g. high-powered laser systems?		X
Section 8: DUAL USE		
Does your project have the potential for military applications?		X
Does your project have an exclusive civilian application focus?		X
Will your project use or produce goods or information that will require export licenses in accordance with legislation on dual use items?		X
Does your project affect current standards in military ethics – e.g., global ban on weapons of mass destruction, issues of proportionality, discrimination of combatants and accountability in drone and autonomous robotics developments, incendiary or laser weapons?		X
Section 9: MISUSE		
Does your project have the potential for malevolent/criminal/terrorist abuse?		X
Does your project involve information on/or the use of biological-, chemical-, nuclear/radiological-security sensitive materials and explosives, and means of their delivery?		X
Does your project involve the development of technologies or the creation of information that could have severe negative impacts on human rights standards (e.g. privacy, stigmatization, discrimination), if misapplied?		X
Does your project have the potential for terrorist or criminal abuse e.g. infrastructural vulnerability studies, cybersecurity related project?		X
Section 10: LEGAL ISSUES		
Will your project use or produce software for which there are copyright licensing implications?		X
Will your project use or produce goods or information for which there are data protection, or other legal implications?		X
Section 11: OTHER ETHICS ISSUES		
Are there any other ethics issues that should be taken into consideration?	X	