Towards a Verifiable Toolchain for Robotics

Charlie Street[1], Yazz Warsame[1], Masoumeh Mansouri[1], Michaela Klauck[2], Christian Henkel[2], Marco Lampacrescia[2], Matteo Palmas[2,3], Ralph Lange[2], Enrico Ghiorzi[3,6], Armando Tacchella[3], Razane Azrou[4], Raphaël Lallement[4], Matteo Morelli[4], Ginny I. Chen[5], Danielle Wallis[5], Stefano Bernagozzi[3,6], Stefano Rosa[6], Marco Randazzo[6], Sofia Faraci[6], Lorenzo Natale[6]

[1]School of Computer Science, University of Birmingham, UK
[2]Robert Bosch GmbH, Bosch Research, Germany
[3]Università degli Studi di Genova, Italy
[4]List, CEA, Université Paris-Saclay, France
[5]Inventya Ventures, Ireland
[6]Istituto Italiano di Tecnologia, Italy
c.l.street@bham.ac.uk, lorenzo.natale@iit.it

Abstract

There is a growing need for autonomous robots to complete complex tasks *robustly* in dynamic and unstructured environments. However, current robot performance is limited to simple tasks in controlled environments. To improve robot autonomy in complex environments, the robot's deliberation system must be able to synthesise correct plans for a task and generate contingency plans for handling anomalous scenarios that were not expected at design time. The robustness of such a system can be quantified using techniques for *formal verification* and validation. This paper outlines the progress of EU project CONVINCE (CONtext-aware Verifiable and adaptIve dyNamiC dEliberation), which aims to develop a software *toolchain* that aids developers in designing, developing, and deploying robot deliberation systems that are fully verified. We describe our modelling approach, each of the toolchain components, and how they interact. We also discuss survey results which demonstrate the demand for a verifiable toolchain among the robotics community.

Introduction

Modern robotic systems are increasingly deployed in unstructured, unseen, and dynamic environments. For example, a mobile manipulator may be used to assemble structures where previously unseen blocks are damaged or vary in size, and humans move through the environment. The ultimate goal of robotics research is to attain *robust* autonomy where robots interact intelligently with their environment and respond effectively to *anomalies*, i.e. unexpected events which are either known or unknown to the system *a priori*. However, robots are still unable to autonomously complete complex tasks in real-world environments. High levels of autonomy can only be achieved for simple tasks in controlled environments. In recent years, the performance of individual robot components such as deliberation, perception, and control have improved greatly. To robustly complete complex tasks and handle unexpected changes in the environment, these components must be comprehensively integrated into a software *toolchain*.

The robustness of autonomous systems can be guaranteed through validation and verification techniques (Espiau, Kapellos, and Jourdan 1996). Research effort within robotics and AI has largely focused on testing single software components to develop safe and dependable autonomous robots (Ingrand 2019). This ignores interactions between components, and how the robot interacts with the environment. We argue that to guarantee robustness the entire robotic system must be verified in its environment. This ensures at design time that the system can complete its task while handling anomalies and guaranteeing that properties describing correct operation are not violated. Such guarantees cannot be provided through testing alone.

Current robotic development is limited by the lack of comprehensive software frameworks for integrating single components into complex systems. The most common approach is to develop independent components and rely on a communication layer such as the robot operating system (ROS) (Quigley et al. 2009) to interconnect them. There exist frameworks which expand on this to provide additional scaffolding for building coherent systems (Bruyninckx 2001; Nordmann, Hochgeschwender, and Wrede 2014; Brugali 2015). However, these approaches lack any formal modelling or verification, which help achieve levels of confidence in the system behaviour that are unreachable through other techniques (Ingrand 2019). Some toolchains explicitly support robot behaviour design and verification (Colledanchise et al. 2021b,a; Meywerk et al. 2020). The cognitive interaction toolkit (CITK) (Lier et al. 2014), Papyrus for robotics (Radermacher et al. 2021), and SmartSoft (Schlegel et al. 2009) cover a large portion of the robotic development cycle. Papyrus (Radermacher et al. 2021) and SmartSoft (Schlegel et al. 2009) in particular provide an integrated set of robotic meta-models which cover common aspects of robotic applications. These meta-models have been extended to support verification in CARVE (Colledanchise et al. 2021b), SCOPE (Colledanchise et al. 2021a), and SafeCC4Robotics (Martinez et al. 2021). However, none of these solutions provide a consistent development tool for integrating deliberation, learning, perception, and control while also providing modelling languages and verification tools. The lack of appropriate tools for verifying autonomous deliberation prevents the introduction of higher cognitive abilities in industry, limiting the development of truly adaptable commercial applications and products.

In this paper, we outline the progress and position of EU project CONVINCE (CONtext-aware Verifiable and adaptIve dyNamiC dEliberation). The key contribution of CONVINCE is to develop and verify cognitive deliberation capabilities that ensure safe robot operation over extended periods of time, i.e. ensure that robots avoid undesirable states or behaviours with a very high probability. In this paper, we describe a model-driven software toolchain which allows developers to build application-specific deliberation systems capable of i) determining the robot behaviours required for a given task; ii) deploying and configuring the components required for these behaviours; and iii) automating the analysis of these behaviours using formal modelling and verification to ensure the system is safe and robust. We also discuss the use cases we will use to evaluate our toolchain, and survey results which demonstrate the demand for a verifiable robotic toolchain among the robotics community.

Toolchain Use Cases

In this section, we describe the real-world robotic use cases we will use to evaluate our toolchain.

Vacuum Robot. Autonomous vacuum cleaners must periodically clean a space within a reasonable duration without getting stuck. Domestic environments feature multiple sources of uncertainty. First, the environment is dynamic due to moving humans, animals, furniture, and other household objects, where dynamics occur over multiple timescales. This makes it challenging for a robot to predict when a location will be free, and to generate a complete list of anomalies it may encounter during execution. For example, small items such as toys and shoe laces may get sucked into the robot, causing it to fail. The robot must also remain small enough to drive under furniture and inside narrow spaces which imposes hardware limitations. We will use our toolchain to improve robot coverage by robustly handling anomalies.

Assembly Robot. In this use case, a mobile manipulator assists a human in assembling a kiln car for heavy clay/ceramic production. Though similar to a classical pick-and-place task, there are multiple challenges to consider. Blocks must be placed with millimetre precision to ensure stability and safety. Blocks may also be occluded, and must be uncovered to complete assembly. Sources of uncertainty include the chance of a human blocking the robot's path, and blocks falling due to possible damage or slight variations in size. Our toolchain will improve the efficiency of assembly while successfully handling known and unknown anomalies.

Museum Tour Guide. In this use case, an autonomous mobile robot must provide guided tours in an art museum. Here, the robot must interact with humans in terms of verbal communication and movement within a crowded environment. This makes the task unpredictable and subject to failures. For example, some areas of the museum may be too crowded for a robot to safely navigate, artworks may be moved, visitor questions may be hard to comprehend, and visitors may leave a tour without notice. Using our toolchain, we will mitigate these issues and improve the quality of robotic tours.

The CONVINCE Toolchain

In Fig. 1, we present an overview of the CONVINCE toolchain. A more detailed overview is available online alongside the implementation status of each toolchain component and an initial open source implementation[1]. The CONVINCE toolchain uses a layered architecture similar to Colledanchise et al. (2021b), where higher layers are more abstracted from the robot hardware. The robot system and environment are modelled formally to admit offline planning and verification, as well as online planning and monitoring. While this implies a model-driven approach, where models are used to generate code, the CONVINCE toolchain is also designed to be used alongside existing codebases. To achieve this, our toolchain uses two types of model. *Concrete models* are directly executable, such as the description of a behaviour tree (BT) (Colledanchise and Ögren 2018), which acts as a task model but can also be executed by an interpreter. *Abstract models* capture the behaviour of a software component but are not directly executable. Abstract models can be used for verification, but there is no formal guarantee that the model matches the behaviour of the component it describes. Our toolchain layers are as follows:

- The *functional layer* interfaces with the robot hardware, including sensors, user interfaces, drives, manipulators and communication in a hardware-specific way.
- The *skill layer* implements basic robot behaviours in a modular way using the functional components, such as object picking and navigating to a charging station.
- The *deliberation layer* orchestrates skills to achieve a robotic task or mission, such as giving a guided museum tour, assembling a structure, or cleaning a house.

Functional components define core robot capabilities, and are often developed by the robot provider or provided through open source libraries. We model functional components using an extension of SCXML[2] developed within CONVINCE. Our extended SCXML format captures robot components and behaviours as finite state machines (FSMs). Functional components are represented as abstract models, as their functionality exceeds what is required for the high-level task. Continuous values in functional components such as a robot's battery level are discretised to reduce the complexity of model checking. Skills in the skill layer can be either concrete or abstract models, again represented using our extended SCXML format. Prior to execution, concrete skill SCXML files are translated into executable robot code using the model to code generator in Fig. 1. We model the deliberation layer using BTs (Colledanchise and Ögren 2018), a reactive, modular, and flexible formalism for high-level robot behaviours. BTs are concrete models

[1] https://convince-project.github.io/overview/

[2] https://www.w3.org/TR/scxml/

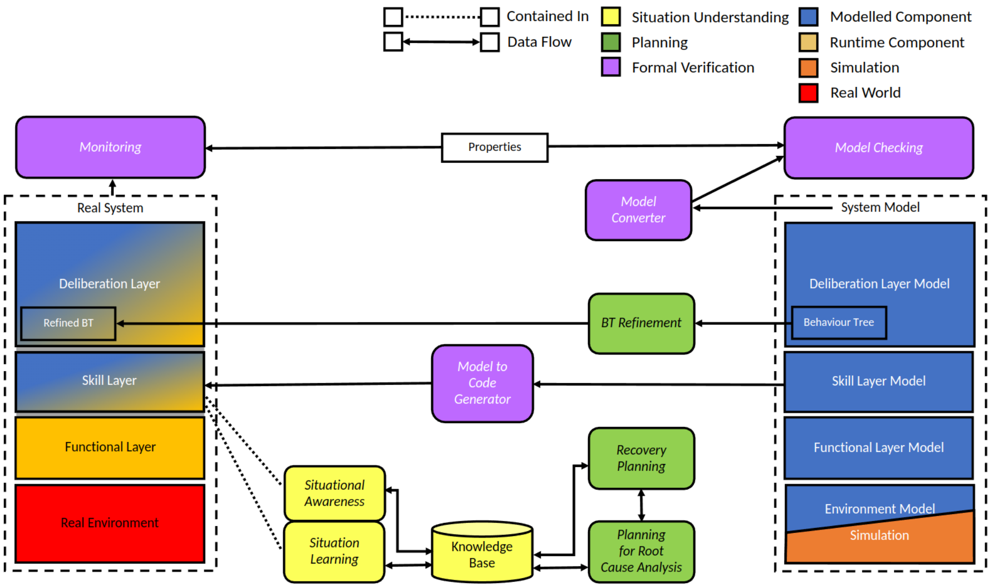

Figure 1: An overview of the CONVINCE toolchain.

specified in the `BehaviorTree.cpp` V3.8[3] XML format, from which they can be directly executed. The BT leaf nodes are defined in the CONVINCE SCXML format as ‘plugins’. Plugins are often associated with a single skill, and act as an interface between the deliberation and skill layers, i.e. they convert ticks received by the BT leaf nodes into function calls for the skill, such as start, stop, or abort. Our toolchain allows models which are tied to a particular middleware, e.g. where ROS interfaces are defined, and models which are middleware agnostic. In fact, a key advantage of the CONVINCE SCXML format is that it allows easier modelling of middleware-specific interfaces, but translates to plain SCXML internally. Properties for system-level verification are specified in metric temporal logic (Koymans 1990) and written in XML. Properties represent logical conditions that should be satisfied during execution. For example, we may want to guarantee that the charge of a vacuum robot never falls below a certain threshold. The complete CONVINCE toolchain data model is available online[4]. Defining formal models for each layer admits verification of the entire robotic system.

Given a set of models, we now describe how the CONVINCE toolchain operates in the context of a museum tour guide. Toolchain components are categorised into either situation understanding, planning, or verification. Some components are run offline prior to execution, and others are run online. Before execution, formal models are verified using offline model checking to evaluate system robustness, e.g. whether tours are consistently completed within a certain duration. A set of execution monitors are also generated to monitor the models, components, and properties during execution. The initial BT model supplied by the designer is also refined using offline planning to improve robustness under uncertainty. BT refinement will allow the robotic tour guide to effectively adjust its tour based on crowd levels. Finally, concrete models are translated into executable code. During execution, the robot receives sensor data which is input to multiple components. Sensor inputs update the knowledge model for situation understanding and planning as well as the execution monitors. In the museum, the knowledge model may capture crowd levels and the tour group location. Monitors inspect messages exchanged by the skills and functional components to ensure that system behaviour matches their models. This is particularly important for abstract models. Monitors also detect property violations caused by anomalies. Upon firing, the anomaly is identified using situational awareness, which reasons over the knowledge model and sensor data. Active planning then identifies the root cause of the anomaly and recovers from it. For example, a robotic tour guide may be unable to see an artwork, and use active planning to identify that it has moved. We proceed by describing each toolchain component in more detail.

Situation Understanding

Our tools for situation understanding allow robots to reason over their perception and prior knowledge to understand the context of their execution and adapt accordingly.

Situational Awareness and Situation Learning. Situational awareness and situation learning form a pipeline for handling known and unknown anomalies. This ensures robustness against unexpected events during execution. At the core of this pipeline is a knowledge base represented as an ontology (Olivares-Alarcos et al. 2019). The first step of situational awareness is to combine data from robot sensors and a digital twin to infer the symbolic system state. If this state contains anomalous predicates or causes a monitor to fire (described below), an anomaly has occurred. A

[3] https://www.behaviortree.dev/

[4] https://github.com/convince-project/data-model/

description of the anomaly is then constructed using feature extraction techniques such as deep learning or symbolic approaches. This description is compared against existing anomaly descriptions in the knowledge base and classified as known or unknown. Known anomalies have existing, executable recovery behaviours. For unknown anomalies, we call our active planner for root cause analysis and recovery (described below). After unknown anomaly recovery, situation learning updates the knowledge base with the new anomaly description and corresponding recovery behaviour.

Planning

Our planning tools synthesise robust robot behaviour in uncertain, dynamic, and unstructured environments.

BT Refinement. BTs are popular within robotics for their reactivity, modularity, and flexibility. However, it is challenging for a human designer to fully account for all sources of uncertainty which affect a robot, such as human movement. Therefore, we include a tool which uses Markovian planning techniques (Puterman 2014; Sutton, Precup, and Singh 1999) to refine the initial BT provided to the deliberation layer. Refinement modifies the logical structure of the BT to achieve robustness under the effects of uncertainty.

Planning for Root Cause Analysis and Recovery. During execution, a robot may experience an unknown anomaly which causes the system to halt, i.e. an anomaly the robot has never experienced or modelled. Anomaly detection is handled by the execution monitors and situational awareness. We present a planning tool which collects information about an unknown anomaly to identify its root cause, and another for anomaly recovery. This is achieved through answer set programming (Lifschitz 2019) and scene graph analysis (Jiao et al. 2022). After recovery, the robot can continue to execute the task plan encoded in the (refined) BT.

Formal Verification

Our verification tools formally evaluate the robustness of the complete robotic architecture offline and online.

Model Checking. Robotic architectures are large concurrent systems which are challenging to verify using traditional numerical model checking techniques. Therefore, we introduce statistical model checkers which verify metric temporal logic properties on the formal system model (Legay, Delahaye, and Bensalem 2010). Our model checkers can be used offline and one of them also online to find violating traces, or the probability of property satisfaction alongside a corresponding confidence interval. Confidence intervals tighten as the time for verification increases. This provides a confidence level over system robustness for a given guarantee. Our toolchain also integrates the Storm model checker (Hensel et al. 2022). Prior to model checking, we use the model converter in Fig. 1 to convert the system model into a format accepted by our model checkers, such as JANI (Budde et al. 2017).

Monitoring. Statistical model checking improves on the complexity of exhaustive numerical model checking, but complexity can still be high for large systems. The models verified offline also do not perfectly reflect reality. This motivates the need for fast, online verification. We achieve this by creating a set of execution monitors offline. Monitors receive input from the real system during execution, and fire if a property is violated, or if we reach a state that is inconsistent with our models. We create and execute monitors using ROSMonitoring (Ferrando et al. 2020).

Developer Survey

To support toolchain development, we surveyed 60 software developers, system/test engineers, researchers, and managers who are building autonomous systems across academia and industry. The survey was conducted during toolchain development and considered robot deliberation models and the use of formal verification for robot testing. Full survey results are available online (Chen et al. 2024).

A substantial number of respondents (30%) are developing autonomous robots for outdoor, unstructured environments. Our toolchain improves the robustness of robot deliberation in these often dynamic and uncertain environments. With regards to modelling, 78% of respondents have used BTs regularly in the past year, and 64% have used FSMs. Further, 53% of respondents chose BTs as their preferred deliberation model. Our toolchain uses BTs and FSMs during modelling, making it familiar and accessible to developers.

Few respondents currently include model checking in their test pipeline (15%). Instead, developers largely rely on simulations (88%), manual testing (78%), and unit tests (44%). However, when asked whether they feel the need for a more systematic testing or verification approach for their system, 78% of respondents answered 'definitely yes' or 'rather yes'. Developers want to 'make [their] system more robust' and handle the complexity of robot systems, as 'it's very difficult to abstract away or mock all the necessary pieces to thoroughly test behaviours'. Respondents are generally willing to expend effort writing formal models if it admits more systematic testing or verification; if a score of one is definitely yes to expending effort, and five is definitely no, the average score was 2.67. This suggests there is demand among the robotics community for tools that support the use of formal methods to increase robustness and allow for the verification of robot deliberation systems. Our toolchain addresses this problem by formally modelling the complete robotic architecture using familiar formalisms, lowering the entry barrier for developers.

Conclusion

In this paper, we argued that a verifiable robotic toolchain is essential for handling the complexities of unstructured, dynamic, and uncertain environments. We have proposed a multi-layer toolchain which utilises formal models at each layer to admit verification for evaluating robot robustness. This is supported with state-of-the-art solutions for situation understanding and planning. The success of our toolchain is dependent upon its adoption in the robotics community. Our toolchain uses familiar modelling formalisms, and will be open sourced to allow for easy integration and extension.

Acknowledgments

This work was funded by the European Union under the Horizon Europe grant 101070227 (CONVINCE). As set out in the Grant Agreement, beneficiaries must ensure that at the latest at the time of publication, open access is provided via a trusted repository to the published version or the final peer-reviewed manuscript accepted for publication under the latest available version of the Creative Commons Attribution International Public Licence (CC BY) or a licence with equivalent rights. CC BY-NC, CC BY-ND, CC BY-NC-ND or equivalent licenses could be applied to long-text formats. Charlie Street, Yazz Warsame, and Masoumeh Mansouri are UK participants in Horizon Europe Project CONVINCE and supported by UKRI grant number 10042096.

References

Brugali, D. 2015. Model-Driven Software Engineering in Robotics: Models are Designed to Use the Relevant Things, Thereby Reducing the Complexity and Cost in the Field of Robotics. *IEEE Robotics & Automation Magazine*, 22(3): 155–166.

Bruyninckx, H. 2001. Open Robot Control Software: The OROCOS Project. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, volume 3, 2523–2528. IEEE.

Budde, C. E.; Dehnert, C.; Hahn, E. M.; Hartmanns, A.; Junges, S.; and Turrini, A. 2017. JANI: Quantitative Model and Tool Interaction. In *Proceedings of Tools and Algorithms for the Construction and Analysis of Systems*, 151–168. Springer.

Chen, G. I.; Klauck, M.; Lampacrescia, M.; Henkel, C.; and Wallis, D. 2024. Results of the CONVINCE Autonomous Systems Developer Survey [Data Set]. https://doi.org/10.5281/zenodo.13382222.

Colledanchise, M.; Cicala, G.; Domenichelli, D. E.; Natale, L.; and Tacchella, A. 2021a. Formalizing the Execution Context of Behavior Trees for Runtime Verification of Deliberative Policies. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 9841–9848. IEEE.

Colledanchise, M.; Cicala, G.; Domenichelli, D. E.; Natale, L.; and Tacchella, A. 2021b. A Toolchain to Design, Execute, and Monitor Robots Behaviors. *arXiv preprint arXiv:2106.15211*.

Colledanchise, M.; and Ögren, P. 2018. *Behavior Trees in Robotics and AI: An Introduction*. CRC Press.

Espiau, B.; Kapellos, K.; and Jourdan, M. 1996. Formal Verification in Robotics: Why and How? In *Proceedings of Robotics Research: The Seventh International Symposium*, 225–236. Springer.

Ferrando, A.; Cardoso, R. C.; Fisher, M.; Ancona, D.; Franceschini, L.; and Mascardi, V. 2020. ROSMonitoring: A Runtime Verification Framework for ROS. In *Proceedings of the Annual Towards Autonomous Robotic Systems Conference (TAROS)*, 387–399. Springer.

Hensel, C.; Junges, S.; Katoen, J.-P.; Quatmann, T.; and Volk, M. 2022. The Probabilistic Model Checker Storm. *International Journal on Software Tools for Technology Transfer*, 1–22.

Ingrand, F. 2019. Recent Trends in Formal Validation and Verification of Autonomous Robots Software. In *Proceedings of the IEEE International Conference on Robotic Computing (IRC)*, 321–328. IEEE.

Jiao, Z.; Niu, Y.; Zhang, Z.; Zhu, S.-C.; Zhu, Y.; and Liu, H. 2022. Sequential Manipulation Planning on Scene Graph. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 8203–8210. IEEE.

Koymans, R. 1990. Specifying Real-Time Properties with Metric Temporal Logic. *Real-Time Systems*, 2(4): 255–299.

Legay, A.; Delahaye, B.; and Bensalem, S. 2010. Statistical Model Checking: An Overview. In *Proceedings of the International Conference on Runtime Verification*, 122–135.

Lier, F.; Wienke, J.; Nordmann, A.; Wachsmuth, S.; and Wrede, S. 2014. The Cognitive Interaction Toolkit–Improving Reproducibility of Robotic Systems Experiments. In *Proceedings of the International Conference on Simulation, Modeling, and Programming for Autonomous Robots*, 400–411. Springer.

Lifschitz, V. 2019. *Answer Set Programming*, volume 3. Springer Heidelberg.

Martinez, J.; Ruiz, A.; Radermacher, A.; and Tonetta, S. 2021. Assumptions and Guarantees for Composable Models in Papyrus for Robotics. In *Proceedings of the IEEE/ACM International Workshop on Robotics Software Engineering (RoSE)*, 1–4. IEEE.

Meywerk, T.; Walter, M.; Herdt, V.; Kleinekathöfer, J.; Große, D.; and Drechsler, R. 2020. Verifying Safety Properties of Robotic Plans Operating in Real-World Environments via Logic-Based Environment Modeling. In *Proceedings of the International Symposium on Leveraging Applications of Formal Methods*, 326–347. Springer.

Nordmann, A.; Hochgeschwender, N.; and Wrede, S. 2014. A Survey on Domain-Specific Languages in Robotics. In *Proceedings of the Conference on Simulation, Modeling, and Programming for Autonomous Robots*, 195–206.

Olivares-Alarcos, A.; Beßler, D.; Khamis, A.; Goncalves, P.; Habib, M. K.; Bermejo-Alonso, J.; Barreto, M.; Diab, M.; Rosell, J.; Quintas, J.; et al. 2019. A Review and Comparison of Ontology-Based Approaches to Robot Autonomy. *The Knowledge Engineering Review*, 34.

Puterman, M. L. 2014. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons.

Quigley, M.; Conley, K.; Gerkey, B.; Faust, J.; Foote, T.; Leibs, J.; Wheeler, R.; Ng, A. Y.; et al. 2009. ROS: An Open-Source Robot Operating System. In *Proceedings of the ICRA Workshop on Open Source Software*, volume 3, 5.

Radermacher, A.; Morelli, M.; Hussein, M.; and Nouacer, R. 2021. Designing Drone Systems with Papyrus for Robotics. In *Proceedings of Drone Systems Engineering and Rapid Simulation and Performance Evaluation: Methods and Tools*, 29–35.

Schlegel, C.; Haßler, T.; Lotz, A.; and Steck, A. 2009. Robotic Software Systems: From Code-Driven to Model-Driven Designs. In *Proceedings of the International Conference on Advanced Robotics*, 1–8. IEEE.

Sutton, R. S.; Precup, D.; and Singh, S. 1999. Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning. *Artificial intelligence*, 112(1-2): 181–211.