Reinforcement Learning State Space Properties for Model Checking of Interlockings*

Ben Lloyd-Roberts[1], Phil James[1], and Michael Edwards[1]

Swansea University, Swansea, UK
{ben.lloyd-roberts, p.d.james, michael.edwards}@swansea.ac.uk

Abstract. Railway interlockings serve as safety layer within railway signalling systems by ensuring proposed signalling requests are safe given the current state of the railway. As a vital part of any railway signalling system, interlockings are critical systems developed with the highest safety integrity level (SIL4) according to the CENELEC 50128 standard. The application of formal methods, in particular model-checking, in order to verify interlockings operate correctly is well established within academia and is beginning to see real applications in industry. However, the uptake of formal methods research within the UK rail industry has yet to make a substantial impact due to current approaches often producing false positives that require manual analysis during verification. Here, it is accepted that so-called invariants can be added to reduce the number of such false positives, however automatically computing these invariants remains a challenge. In this work we present a first approach illustrating how reinforcement learning can be used to learn properties of a state space for a given railway interlocking. Understanding how a verification problem can be formulated as one where machine learning can be successfully applied is the first step towards automated learning of invariants.

Keywords: Reinforcement Learning · Interlocking · Model Checking.

1 Introduction

Interlockings serve as a filter or 'safety layer' between inputs from operators, such as route setting requests, ensuring proposed changes to the current railway state avoid safety conflicts. As a vital part of any railway signalling system, interlockings are critical systems regarded with the highest safety integrity level (SIL4) according to the CENELEC 50128 standard. The application of model-checking to Ladder Logic programs in order to verify interlockings is well established within academia and is beginning to see real applications in industry. Bounded model-checking is an efficient means of verifying a system through refutation. Given an abstracted model of a target system M and some safety properties ϕ, BMC searches for counterexamples up to some precomputed bound k. Search terminates when either an error trace is produced or the bound k is reached.

* Supported by Siemens Mobility UK & EPSRC

Determining this completeness threshold to sufficiently cover all states is often computationally intractable given it's true value will depend on model dynamics and size of the search space. Additionally, the k-induction rule, which checks if the property ϕ holds inductively for k sequences of states, is constrained to acyclic graphs for guarantees of completeness.

An invariance rule allows us to establish an invariant property ψ which holds for all initial states and transitions up to a given bound. Invariants may hold for sub-regions of the state space, meaning complete coverage isn't necessary to learn them. Supporting k-induction with strengthening invariants helps reduce the overall search space for bounded model checking, proving that invariant aspects of the system need not be considered. This can help filter cases where false negative counter examples are triggered by unreachable states. Generating sufficiently strong invariants is a non-trivial process given their construction is heavily program dependent. Usually such invariants require domain knowledge, typically devised by engineers responsible for the program implementation.

We infer two principle challenges to address. First, formulating theoretical and practical frameworks in an academic-industry partnership with Siemens Rail Automation UK to represent interlocking verification as a goal-orientated reinforcement learning task. Second, devising an appropriate strategy for learning invariants within those frameworks.

The aim of this paper is to address this first challenge. Reinforcement learning is a popular machine learning paradigm with a demonstrably impressive capacity for learning near optimal strategies for goal-orientated tasks. Such approaches are well suited to problems where one need systematically learn the behaviour of a deterministic system, record observations regarding different states of being and identify patterns across those states. Generalisation of learned policies across different 'environments' being one of the key challenges in RL, we first aim to learn conceptually simpler goals, such as finding the progressively larger upper bounds for BMC. Understanding how a verification problem can be formulated as one of where machine learning can be successfully applied suggests the promise of learning invariants this way.

2 BMC & Program Invariants

2.1 Reinforcement Learning

Reinforcement learning is a principle machine learning paradigm which models sequential decision making problems with respect to some goal-orientated task as the optimal control of some incompletely-known MDP, known as the environment [1]. Given a permitted set of actions to perform over a series of discrete time steps, a software agent is trained to interact with and observe changes in its environment based on intermittent reward signals. Over a process of accelerated trial-and-error agents learn a function, or policy, mapping states to optimal

actions likely to return the greatest cumulative future reward.

For purposes of invariant finding, we aim to maximise state space, or environment, coverage. In order to reinforce this behaviour, we motivate agents to pursue the longest loop free path where a reward scheme positively rewards sequences of novel observations. Inversely, we issue a negative reward for observing similar states within a single episode.

We select asynchronous advantage actor-critic which estimates both the value function and behaviour policy when exploring an environment. The asynchronous nature of the algorithm facilitates distributed exploration of state-action pairs via separate workers with a shared global model of their environment.

2.2 Environment Generation

Given exhaustive search of large state spaces is often computationally intractable, we generate a set of ladder logic programs where the number of reachable states and recurrence reachability diameter are known. Using the well established pelican crossing example as a base template, we introduce a single contact and coil for each additional rung during program generation. Where $|S(p_i)|$ represents the number of reachable states for a program p_i, a subsequently generated program p_{i+1} with one additional rung, has $2|S(p_i)| + 1$ reachable states. Through a series of training runs on each environment we record the number of states observed by workers to gauge the overall state space coverage.

3 Learning Framework

For each environment we set a max number of episodes $T_{\max}$ to constrain the duration of training. Additionally we introduce two forms of early termination to avoid superfluous interaction with the environment. First, if the performance curve of a given agent converges to some local minima. Second, if all reachable states have been observed at least once. Given agents accumulate experience within separate environment instances, they update shared sets of observations and rewards intermittently. This update frequency depends on both the number of workers and precomputed size of the environment.

4 Results

5 Conclusion & Future Work

References

1. Sutton, R.S., Barto, A.G.: Reinforcement learning: An introduction. MIT press (2018)

Environment Metrics			Training Metrics			
States	**Reachable States**	**Actions**	**Max K**	**States Observed**	**Coverage**	**Total Episodes**
2^{12}	7	2	6	7	100	1.00E+04
2^{14}	15	4	14	15	100	1.00E+04
2^{16}	31	8	28	31	100	1.00E+04
2^{18}	63	16	48	63	100	1.00E+05
2^{20}	127	32	33	127	100	1.00E+05
2^{22}	255	64	76	255	100	1.00E+04
2^{24}	511	128	49	377	100	1.00E+05
2^{26}	1023	256	306	1023	100	3.00E+05
2^{28}	2047	512	538	2047	100	3.00E+05
2^{30}	4095	1024	1418	4084	99.731	3.00E+05
2^{32}	8191	2048	1712	7907	96.532	3.00E+05
2^{34}	16383	4096	1498	15654	95.550	3.00E+05
2^{36}	32767	8192	2879	27752	84.694	3.00E+05
2^{38}	65535	16384	1969	58373	89.071	3.00E+05
2^{40}	131071	32768	2692	108638	82.884	2.00E+05
2^{42}	262143	65536	1406	5199317	76.033	3.00E+05
2^{44}	524287	131072	1782	325781	62.137	2.00E+05
2^{46}	1048575	262144	1593	671645	64.053	3.00E+05
2^{48}	2097151	524288	1598	1206867	57.547	3.00E+05
2^{50}	4194303	1048576	2527	1625338	38.751	5.00E+05