Explainability and Dependability Analysis of Learning Automata based AI Hardware

Rishad Shafik, Adrian Wheeldon and Alex Yakovlev
Microsystems Research Group, School of Engineering, Newcastle University, NE1 7RU, UK.
E-mail: Rishad.Shafik@ncl.ac.uk; Adrian.Wheeldon@ncl.ac.uk; Alex.Yakovlev@ncl.ac.uk

***Abstract*—Explainability remains the holy grail in designing the next-generation pervasive artificial intelligence (AI) systems. Current neural network based AI design methods do not naturally lend themselves to reasoning for a decision making process from the input data. A primary reason for this is the overwhelming arithmetic complexity.**

Built on the foundations of propositional logic and game theory, the principles of learning automata are increasingly gaining momentum for AI hardware design. The lean logic based processing has been demonstrated with significant advantages of energy efficiency and performance. The hierarchical logic underpinning can also potentially provide opportunities for by-design explainable and dependable AI hardware. In this paper, we study explainability and dependability using reachability analysis in two simulation environments. Firstly, we use a behavioral SystemC model to analyze the different state transitions. Secondly, we carry out illustrative fault injection campaigns in a low-level SystemC environment to study how reachability is affected in the presence of hardware stuck-at 1 faults. Our analysis provides the first insights into explainable decision models and demonstrates dependability advantages of learning automata driven AI hardware design.

I. Introduction

Tsetlin machine (TM) is a promising new machine learning (ML) algorithm, recently proposed by Ole-Christoffer Granmo [1]. It is built on Mikhail Tsetlin's original learning automaton based control principles for complex systems [2] as well as contemporary linear tactics based game theory [3]. Discretization of the control states is a major simplification in TM, which allowed for using linear tactics to reinforce the states over time in parallel [4]. As such, TM can define a machine learning problem through hierarchical and powerful propositional logic expressions [1]. These have enabled the design of the first-ever hardware architecture [5], which demonstrated significantly lower energy consumption and resource frugality than state-of-the-art neural networks alike.

Figure 1 depicts a schematic diagram of different structural blocks in the TM hardware. As can be seen, a fundamental property of TM is data encoding at the input as a set of Boolean digits rather than binarized numbers with positional significance. These digits and their complements define a set of literals. The combination of these literals participating in the definition of the output class is controlled by Tsetlin automata (TAs), which are finite automata with linear tactics. Each TA constitutes a set of states that define the discrete action space. During training rewards are used to reinforce the states towards an action and penalties are used to transition the states for weakening automaton confidence in performing an action. Ensemble of TA actions define the output of a clause.

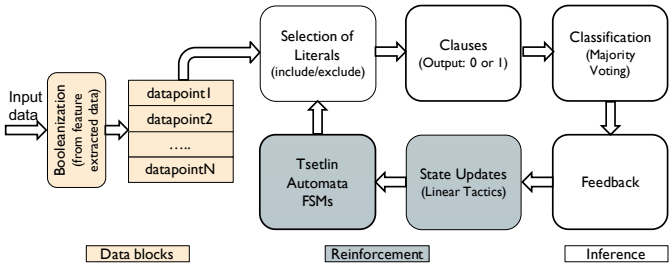


Figure 1. A schematic diagram of TM, showing different structural blocks

These action updates take place in discrete space, rather than in gradient-descent steps, which is another major differentiator when compared to traditional neural networks. This feature can be exploited for discernible and explainable artificial intelligence (AI) hardware design. This requires understanding reachability of TAs states and clause outputs in relation to the Boolean literals during the training and inference exercises.

In this paper, we provide the first insights into explainability of TM using reachability analysis. Additionally, we study dependability of the same in terms of the impact of faults on state reachability. Specifically, we make the following *contributions*: firstly, a state transition based reachability analysis of TM applied to a binary XOR example for demonstration, and secondly, a fault injection campaign led analysis to investigate the impact of stuck-at faults on the state transitions and their reachability.

The paper is organized as follows. Section II provides a state transition based reachability analysis. Section III studies the reachability further in the presence of faults. Finally, Section IV concludes the paper highlighting our future work.

II. Reachability Analysis and Explainability

Commonly, when people refer to reachability analysis they define it as a process of exploring the set of states that a (usually discrete event) system can visit while performing a set of permitted actions. Often this process has a specific aim associated with checking certain properties of the system. In our research, we define reachability slightly more specifically, as the property of the system that allows it to navigate through the finite state-space produced by the composition of finite-state automata, namely TAs. This property is crucial for the hardware to generate the intended and bounded outputs by relating them to sequences of the input data points.

To investigate reachability of the AI hardware using the principles of learning automata (Figure 1), a key hardware block is the team of TA within the reinforcement part. The TA use their internal states to facilitate the selection (i.e.

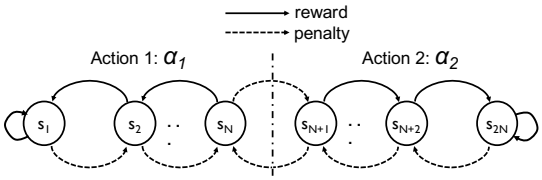


Figure 2. A Tsetlin automaton for 2-action environment with $2N$ states

inclusion or exclusion) of Boolean literals for clauses, thereby defining the clause outputs. The clause outputs then govern the feedback mechanism in the inference part, which subsequently generates the reward/penalty signals sent to the TAs. Hence, the overall operational cycle involves the work of both sequential part (TAs) and combinational part (clauses, classifiers and feedback). As input data sequences are applied, the whole system evolves in the TA state-space and eventually reaches the subset of states (trained states) where the system can perform its most advantageous classification decisions. The latter property, convergence to the stable trained state, is crucial for the accuracy and efficiency of the TM in terms of performance and energy. Besides, reachability becomes a measure of explainability because the trajectories of states through which the system converges can be easily traced.

Figure 2 shows a high-level state transition diagram of each automaton with $2N$ internal states in a 2-action environment. We denote the TA states as $\mathbf{S}=\{s_1, s_2, \ldots s_n \ldots s_{2N}\}$, where s_n is the n-th state. Each automaton initially starts with a random state near the action boundary, i.e. either s_N or s_{N+1}. This allows for the TA to make minimum number of state transitions to reinforce an action. After each reinforcement step, a reward is used to strengthen an action or a penalty is used to weaken the automaton confidence in performing the current action [1]. Since state transitions take place in discrete single steps, s_n is the TA state resulting from a transition from either of s_{n-1} or s_{n+1}. For a given state of s_n, the action performed by the automaton is given as:

$$G(s_n) = \begin{cases} \alpha_1; & \text{if } 1 \leq n \leq N \\ \alpha_2; & \text{if } (N+1) \leq n \leq 2N \end{cases} \quad (1)$$

To demonstrate the number of reinforcement steps needed to fully converge to the final state as well as the corresponding action, we consider an automaton with $2N = 6$ internal states and 2 actions. The state transition equations of all automaton states are given as below:

$$s_1 = (s_1 \text{ AND } R) + (s_2 \text{ AND } R); \quad (2)$$

$$s_2 = (s_1 \text{ AND } P) + (s_3 \text{ AND } R); \quad (3)$$

$$s_3 = (s_2 \text{ AND } P) + (s_4 \text{ AND } P); \quad (4)$$

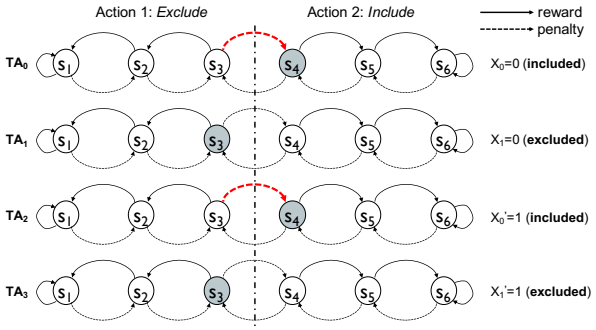
$$s_4 = (s_3 \text{ AND } P) + (s_5 \text{ AND } P); \quad (5)$$

$$s_5 = (s_4 \text{ AND } R) + (s_6 \text{ AND } P); \quad (6)$$

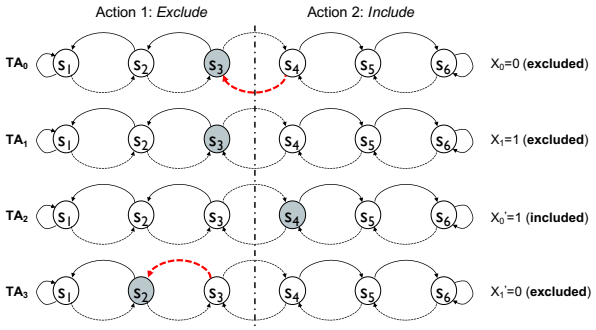
$$s_6 = (s_5 \text{ AND } R) + (s_6 \text{ AND } R), \quad (7)$$

where R and P are the reward and penalty signals generated by the state update circuit (Figure 1). From Eqns. (2)-(7), given the random initial state of either s_3 or s_4, the automaton needs minimum 3 or 4 reinforcement steps.

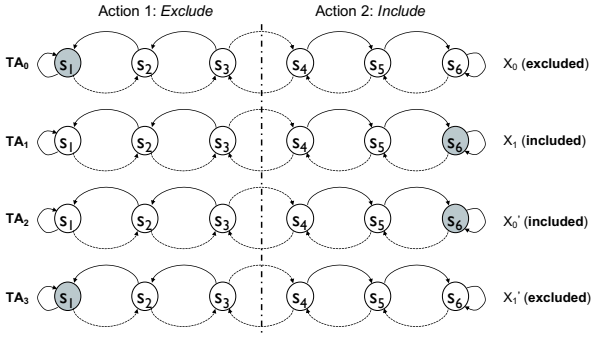
The states of the whole TM are formed as Cartesian products



(a) Reinforcing 4 TA with datapoints (X_0, X_1): (0, 0)



(b) Reinforcing 4 TA with datapoints (X_0, X_1): (0, 1)



(c) TA states After 11 reinforcement steps (i.e. 44 datapoints)

Figure 3. Illustrative example of TA state changes in a 2-input binary XOR

of the states of individual TAs. To illustrate how bounded TA state transitions contribute to reachable learning formulation in the TM algorithm, we simulate a 2-input XOR using a behavioral SystemC description of the same. The inputs and their complements constitute 4 literals and as such 4 TA are used in each clause. Each automaton consists of 6 states as exemplified above. A total of 4 clauses are used in the inference circuit, of which 2 are positive clauses and 2 are negative clauses into the majority voting (i.e. classification) circuit. Figure 3 shows the internal states of 4 TA, defining one clause output only.

The state transitions in a training step correspond to 4 datapoints (which are the set of literals), but only 2 are shown. The TA start with the same initial states of $s3$. After the first datapoint ($\mathbf{X}$:$[X_0, X_1]$=$[0, 0]$) reinforcement, the clause sees an output of 1 as all TA states suggest no inclusion of 0 literals. Overall, this results in an erroneous classification and as such 2 penalties in TA_0 and TA_2, causing them to transition

to s_4 (Figure 3(a)). After the second datapoint (**X**:[0, 1]), the clause output is 0 as the TA_2 state favors the inclusion of a 0 literal ($X_0^{'}$). However, as the clause output generates a wrong classification but with a lower error, TA_0 is penalized to s_3 and TA_3 is rewarded to s_2 (Figure 3(b)). With more datapoints and their associated single-step reinforcements (Eqns.(2)-(7)), the TA continue to settle for states with higher reward probabilities, e.g. s_1 and s_6 (Figure 3(c)). This guarantees convergence during training.

The above analysis of reachability for the XOR example shows an important property of the TM, where the (integer) vector of states of TAs is effectively mapped (contracted) onto the (binary) vector of actions include/exclude. This mapping allows us to define the notion of equivalence between the states of TAs, and hence define the conditions for detecting convergence to the trained state as soon as possible, thus improving the efficiency of the system and its performance.

III. Dependability Analysis

We continue our reachability analysis further in this section and study how dependability of the system is affected in the presence of faults. For these, we use an RTL SystemC model of a 2-input XOR with fault injection handles using [6]. Our fault injection campaign includes a stuck-at 1 fault model, applied to the reinforcement part, i.e. TA. Our future research includes comprehensive fault injection in the TM.

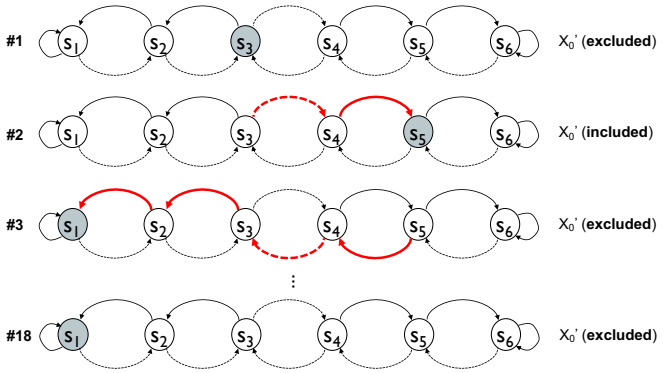
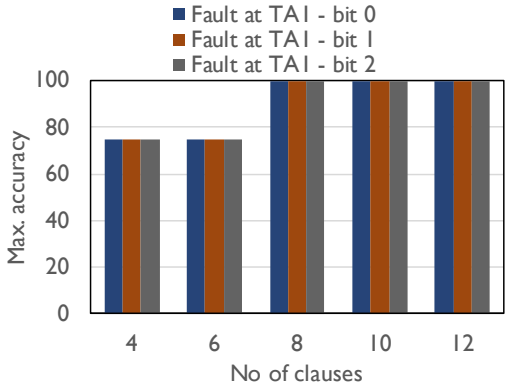


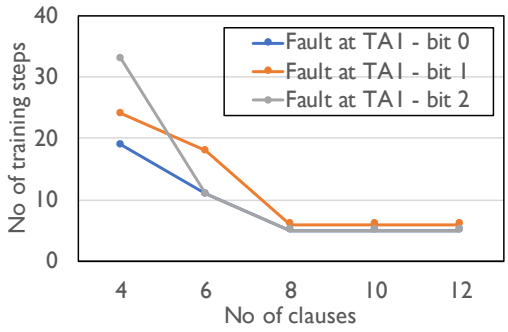
Figure 4. The impact of a stuck-at 1 fault in TA_1's state transition and hence learning. Notice how originally included literal is now excluded because of fault in the TA.

For demonstration purposes, we inject a stuck-at 1 fault in the least significant bit (i.e. bit position 0) of automaton 1 (i.e. TA_1) within the first clause. This is done to observe how this fault can change TA_1 state transitions (see Figure 4) when compared with the same in Figure 3. As can be seen, the automaton assumes an initial state of s_3 and does not change the state after the iteration step 1. This is equivalent to a no-action reinforcement of 4 datapoints. In the iteration step 2, the automaton state is penalized towards s_4 through an increment operation (i.e. from register value of 011 to 100). However, due to the fault the automaton transitions to s_5 (i.e. a register value of 101). After iteration step 3, the automaton is rewarded towards $s6$. However, the faulty automaton state tries to transition to an unreachable state of s_7. As the state bounds are protected through a [*modulus 6 +1*] operation internally, the automaton changes the state to s_1. The automaton retains this state until automata in all clauses are converged (after 18 iteration steps). Note that unlike the TA_1 state in the first clause of the fault-free TM (Figure 3), the faulty automaton excludes the associated Boolean literal, $X_0^{'}$.

From Figure 4 it is evident that a fault in an automaton can influence its state transitions significantly. For example, the state values of TA_1 are constrained to only 3 out of 6 states: s_1, s_3 and s_5, 2 of which are inclined towards the exclude action. This affects the reinforcement as well as inference for the first clause, resulting in a maximum achievable accuracy of 75%. Indeed, defining the relationship between input datapoints and output classes can be challenging with limited state transitions if there are no other means of fault masking or mitigation.



(a) Max. accuracy for different number of clauses

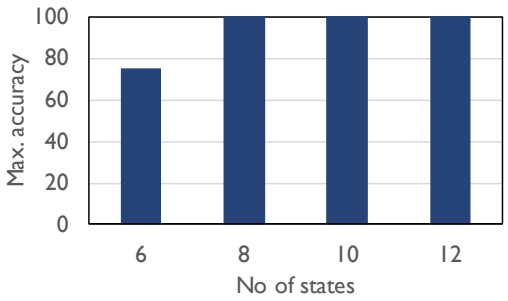


(b) No of iteration steps for max. accuracy

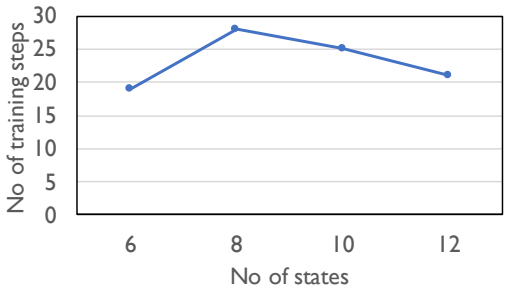
Figure 5. The impact of stuck-at 1 faults in TA_1 at different bit positions in terms of accuracy and performance; number of clauses are varied to observe how clause redundancy naturally masks the faults.

Next, we will explore if TM design allows for fault masking if resource provisions are relaxed in terms of number of clauses in the inference part (Figure 1). For this, we carried out another experiment with variable number of clauses from 4 to 12, each with 6 TA states. Figure 5 presents the results in terms of the maximum training accuracy and the corresponding number of iteration steps to convergence. To observe the significance of fault positions, we injected stuck-at 1 faults in different positions of the TA_1 register: at bit positions 0, 1 and 2. As expected, when the number of clauses are increased to 8 or more, the training accuracy increases to 100% for all fault injection campaigns (Figure 5(a)). Provisioning more clauses in TM allows for further state transition variations.

More variations, in turn, provide masking of the stuck-at fault completely. This observation is akin to traditional fault-tolerant design principles [7], where redundant hardware resources together with majority voting mitigate the impact of faults. TM already features majority voting in the classification circuit and as such it allows for more clauses to independently process the different automaton states internally and yet find the team of automata that correctly define the relationship between Boolean literals and output classes.



(a) Max. accuracy for different number of TA states



(b) No of iteration steps for max. accuracy

Figure 6. Impact of stuck at faults in TA_1 in terms of accuracy and performance with variable number of TA states.

Although fault positions do not affect the accuracy as clauses are increased, they influence the training times (Figure 5(b)). This is because the fault positions can constrain the number of state transitions available to an automaton, often with an action bias. This can increase the number of reinforcement steps needed to increase the automaton action confidence. For example, a stuck-at fault in bit position 2 is more challenging to mask as it only allows for the include action states: s_4 (100), s_5 (101) and s_6 (110). The other automata within the clause take more reinforcement steps to converge their states diverging from this bias. This also explains the longer convergence time with lower number clauses. However, as the number of clauses is increased, the training convergence times decrease due with more redundancy and diversity between clauses.

Finally, we study the impact of number of states (i.e. state register sizes as well as their values) on the reachability of TM states under fault scenarios. For this, we repeat the stuck-at 1 fault injection in TA_1 register in bit position 0 for 4 different state sizes: from 6 to 12, each with a 4-clause configuration. Figure 6 shows the maximum training accuracy as well as their convergence times. As can be seen, the accuracy increases from 75% to 100% when the number of states is increased from 6 to 8, corresponding to a 1-bit increase in the automaton register size from 3 (Figure 6(a)). The increase in the register size as well as the state values allow each automaton to explore a larger state-space. In a 6-state (3-bit) automaton register, a stuck-at 1 fault in bit position 0 can result in 3 allowable states. Conversely, in an 8-state (4-bit) automaton register the same can results in 4 allowable states. Note that, with one clause unable to provide correct outcomes, the 6-state automaton converges faster than the 8-state automaton. However, as more state values are allowed in the automaton, the learning converges faster to the maximum accuracy of 100% (Figure 6(b)).

IV. Summary and Conclusions

We presented the first insights into explainability and dependability of learning automata based AI hardware design using reachability analysis. Our key findings are as follows. Firstly, with a bounded state-space, TM can start from random initial TA states and yet reach a learnt state with incremental reinforcements. As the initial training datapoints generate erroneous outcomes, the randomization-enhanced feedback mechanism continues to navigate to and strengthen an action with higher reward probabilities when it reduces errors [1]. This guarantees convergence. Secondly, with suitably chosen redundant clauses and thereby more state transition variations, stuck-at faults can be fully masked without requiring any additional fault mitigation strategy. The TM can achieve the maximum accuracy faster during training with higher number of clauses. Thirdly, by allocating more TA states and as such expanding the valid state-space, stuck-at faults can also be completely masked. Under fault scenarios, higher number of states allows for faster learning convergence. Compared with clause redundancy approach, expanding the state register sizes provides more energy-frugality. Our future work includes reachability analysis using formal checking tools and theory with comprehensive fault injection campaigns.

Acknowledgement

The authors gratefully acknowledge the inputs from Jie Lei as well as funding from EPSRC IAA project "Whisperable" and EPSRC grant STRATA (EP/N023641/1).

References

[1] O.-C. Granmo, "The Tsetlin Machine - A Game Theoretic Bandit Driven Approach to Optimal Pattern Recognition with Propositional Logic," *arXiv e-prints*, Apr. 2018. [Online]. Available: http://arxiv.org/abs/1804.01508

[2] I. M. Gel'fand and M. L. Tsetlin, "Some methods of control for complex systems," *Russian Mathematical Surveys*, vol. 17, no. 1, p. 95, 1962.

[3] J. Von Neumann and O. Morgenstern, *Theory of games and economic behavior (commemorative edition)*. Princeton university press, 2007.

[4] O.-C. Granmo, "Introduction to the Tsetlin Machine," University of Agder, Norway, Tech. Rep., 2019.

[5] A. Wheeldon *et al.*, "Learning automata based AI hardware design for IoT," *Philosophical Trans. A of the Royal Society*, vol. (in press), 2020.

[6] R. A. Shafik, P. Rosinger, and B. M. Al-Hashimi, "SystemC-based minimum intrusive fault injection technique with improved fault representation," in *14th IEEE Intl. On-Line Testing Symposium*, 2008, pp. 99–104.

[7] J. Mathew, R. Shafik, and D. K. Pradhan, *Energy-efficient fault-tolerant systems*. Springer USA, 2014.