Emergent Control of MPSoC Operation by a Hierarchical Supervisor / Reinforcement Learning Approach

Florian Maurer*, Bryan Donyanavard†, Amir M. Rahmani†, Nikil Dutt†, Andreas Herkersdorf*
*Technical University Munich, Germany
†University of California, Irvine, USA

***Abstract*—MPSoCs increasingly depend on adaptive resource management strategies at runtime for efficient utilization of resources when executing complex application workloads. In particular, conflicting demands for adequate computation performance and power-/energy-efficiency constraints make desired application goals hard to achieve. We present a hierarchical, cross-layer hardware/software resource manager capable of adapting to changing workloads and system dynamics with zero initial knowledge. The manager uses rule-based reinforcement learning classifier tables (LCTs) with an archive-based backup policy as leaf controllers. The LCTs directly manipulate and enforce MPSoC building block operation parameters in order to explore and optimize potentially conflicting system requirements (e.g., meeting a performance target while staying within the power constraint). A supervisor translates system requirements and application goals into per-LCT objective functions (e.g., core instructions-per-second (IPS). Thus, the supervisor manages the possibly emergent behavior of the low-level LCT controllers in response to 1) switching between operation strategies (e.g., maximize performance vs. minimize power; and 2) changing application requirements. This hierarchical manager leverages the dual benefits of a software supervisor (enabling flexibility), together with hardware learners (allowing quick and efficient optimization). Experiments on an FPGA prototype confirmed the ability of our approach to identify optimized MPSoC operation parameters at runtime while strictly obeying given power constraints.**

***Index Terms*—Backup-based reinforcement machine learning, MPSoC runtime management, hierarchical reflective control**

I. Introduction

Nature is a rich source of examples where the collective behavior of individual entities can accomplish astonishingly complex tasks in an optimal or near optimal manner. This phenomenon is known as emergence [1] and results from hidden causal relationships among individuals that follow elementary rules and actions. Consider, for example, an ant colony finding the shortest path to food and other resources by following the strongest pheromone deposition of their peers [2]. Or, flock of birds during migration, flying with slight lateral offset, yielding an overall aerodynamic and energy efficient formation for the entire bird community [3]. Fish schools cluster to form assemblies in order to protect themselves from predators by controlling only their relative position, speed and orientation to neighbors [4]. Emergent control is mimicked in synthetic environments such as Conway's game of life [5], emulating birth and death of artificial cellular automata. In all mentioned examples, individuals follow specific, simple rules to take actions that result in beneficial outcomes such as shortest paths, least energy consumption, or improved safety for the overall community, at a higher-level of representation. We draw parallels to scenarios in which many distributed decision-making mechanisms are deployed in complex computer systems for runtime management, e.g., dynamic power management of many-core processors using per-core frequency throttling. Therefore, a strong case can be made for technical systems to exploit emergent behavior for managing complex multi-agent systems in a distributed manner with simple means of control at the individual constituent level. However, if not managed carefully, emergent behavior may also result in uncontrolled oscillations [6] and chaotic actions. Furthermore, the borderline between desirable and unwanted outcomes can be very narrow.

In nature, the differentiation between beneficial and detrimental rules and actions are determined over centuries of multi-generation Darwinistic evolution. Time to market and economic pressure in system design do not allow for such a survival-of-the-fittest design methodology. Hence, two critical challenges for adopting emergent control in technical systems can be formulated: 1) how to overcome multi-generation evolution in order to adhere to state of the art development cycle times; and 2) how to systematically avoid unproductive or non-constructive emergent control leading to overall chaotic system behavior.

In this paper, we present an approach as well as an architecture to tackle these two challenges in a systematic manner for optimizing configurable parameters in multi-core processor systems. We envisage reinforcement-based machine learning to replace natural evolution according to Darwinasian selection. Our approach to replace natural selection proactively explores the solution space by applying different operation parameter settings and assesses the corresponding outcomes with fitness values. We apply learning classifier tables (LCTs) [7], which are interpretable rule-based hardware machine learning entities, for this purpose. As rule application and assessments with LCTs can be done in the order of milliseconds, the evolutionary exploration of a given solution space corresponding to dozens or even hundreds of generations can now be accomplished in seconds. Besides evolution speed, the quality of the identified condition-action pairs is a decisive crite-

rion. It is known that machine learning-based solution space exploration cannot guarantee finding stable behaviors at all times. Hence, we deploy a reflective supervisory control layer (e.g., SPECTR [8]) to monitor and, when necessary, guide the emergent behaviors by applying tighter constraints to the LCT leaf controllers. An archive-based backup policy provides the ability to adhere to constraints. The archive returns the system to valid configurations once it is approaching constraints.

II. Background and Related Work

The literature on autonomous complex systems engineering relates the terms bio-inspired control and emergent behavior frequently to so-called self-x properties, such as self-adaptiveness, self-organization, self-healing or computational self-awareness, as well as to fields like autonomic or organic computing [9].

Corresponding reference architectures, such as Observe-Decide-Act (ODA) or Monitor-Analyze-Plan-Execute (MAPE), have been developed to control configurable systems at runtime in order to achieve a specified goal [10], [11]. In such architectures, an observer/monitor feeds information on the functional system to an observer/analyzer/planner which reasons on the observed parameters with respect to the specified goal. The control loop is closed by a subsequent planning and actuation step that may take additional knowledge sources into account. In Learning Classifier Systems (LCS) such as XCS [12] or LCT [7], the reasoning and action derivation is made on the basis of conditional rule tables and fitness assessments as part of a reinforcement learning process. The learning in these systems adapts to changing workloads, but the process may explore undesired configurations, or, in worst case, never converge. The German Research Foundation (DFG) research unit OC-Trust generally addressed the question of trust in emergent, self-adaptive systems and, in particular, the convergence issue by enforcing predefined bounds in multi-agent organic computing (OC) software systems [13]. While the constraints in OC-Trust are determined during design time, and checked during runtime, we envision dynamic constraint adjustments by a higher-layer supervisor throughout runtime [14].

A. LCTs

LCTs were introduced by Zeppenfeld et al. [15] as a subset of Wilson's accuracy based classifier (XCS) [12]. LCTs are hardware reinforcement learners that make configuration decisions in order to optimize system parameter settings towards a specific goal. This is enabled by the rule-based structure of LCTs. A single rule in the rule table consists of condition, action and fitness.

The operation of an LCT consists of four periodically applied steps (Figure 1):

1) The LCT builds the match set [M], a subset of all rules in the rule table [P]. A rule is part of the match set if its condition matches the current sensor values.
2) Using the roulette-wheel selection algorithm [17], the LCT selects an action based on the match set [M] and the corresponding fitnesses. All rules of the match set [M] containing the selected action become part of the newly built action set [A]. The roulette-wheel selection algorithm enables the LCT to overcome local optima, as it also occasionally selects actions with low fitness.
3) The selected action is applied to the system by actuators, and the action set is saved in $[A]_{t-1}$.
4) The effect of an action on the system is observed by sensors and graded based on the effectiveness towards achieving the specified goal (using the objective function). Based on the objective function value, a reward is calculated, which is used to update the fitnesses in the rule table [P] of all rules in the saved action set $[A]_{t-1}$. In XCS, for the fitness update a modified version of Q-learning proposed by Wilson is used [17].

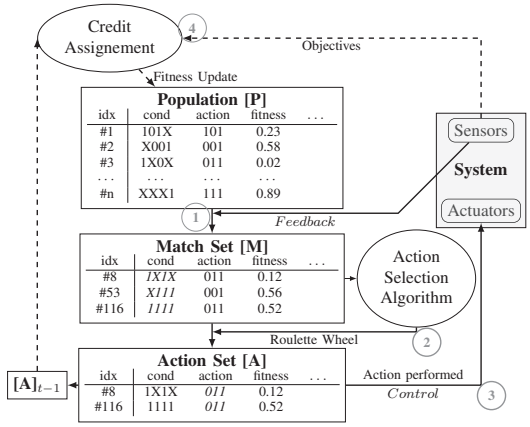


Fig. 1: Overview of the LCT logic from [16] (dashed lines correspond to the fitness update path).

B. Hierarchical Decomposition

Research in intelligent management of computer systems has established hierarchy as an effective way to provide coordination and adaptivity to controllers in a scalable manner [8]. SOSA [16] specifically implements a resource management hierarchy with a supervisor that coordinates distributed LCTs to achieve a global goal. The physical system is horizontally decomposed into subsystems in order to simplify the controllers (i.e., LCTs) and provide scalability. The supervisor oversees each LCT module, guiding its behavior through objective function and rule definition, and coordinating with other LCTs. We propose a similar supervisor in [14] to specify constraints and bounds for LCT objective space, as shown in Figure 2.

III. Managing Emergent Behavior

A key optimization challenge in system management lies in the fact that the optimization is performed for metrics that cannot directly be influenced by changing different configurable

parameters. That is, LCTs act on the *solution space* defined by configuration parameters (e.g., core operating frequency, and cache capacity) whereas the actions have an impact on the *objective space* defined by measurable metrics (e.g., throughput, temperature, and power). Moreover, in LCTs, the metrics from the objective space are used to grade the actions applied on the solution space. Figure 2 shows the relationship between objective and solution space. Configuration **c1** results in the system state **s1**. Due to variability sources such as ambient temperature, shared resource contention, and software complexity, this state is not precisely predictable for a specific configuration. After observing state **s1**, the LCT decides on a new configuration **c2**, which in turn results in the new state **s2**. State **s2** is used to evaluate the effectiveness of the previously applied rule. [14]

A. Problem Definition

In [7] and [18] LCTs have the freedom to explore the entire solution space to achieve a goal following a mathematical function. In consequence, the resulting optimization mechanism does neither adhere to physical constraints, e.g., thermal design power (TDP), nor to application performance requirements. For example, a video playback which has a target of $60\,\mathrm{frames-per-second(fps)}$ should by no means undershoot a lower bound of $30\,\mathrm{fps}$.

So in contrast to the optimization problem tackled in [7] and [18], our optimization formulation consists of a performance target, corresponding bounds plus a power constraint. Hereby, constraints are more severe than bounds. Therefore, we distinguish between the following zones within the objective space represented in Figure 2:

- *prohibited*: This zone is defined by physical constraints. This is the most severe zone and is therefore also refereed to as *constraint* for the remaining text.
- *avoided*: This zone is derived from application performance requirements and framed by the introduced *bounds*.
- *allowed*: This is the area of the objective space which the LCTs should actively explore in order to identify an optimal state by trying different configurations.

B. Backup Policy

Adhering to a constraint means preventing any violation of it. In state of the art approaches this is achieved by backup policies [19]. To prevent constraint violations, backup policies must act before a violation occurs. We propose a margin zone to act as a buffer in the allowed zone in order to avoid the prohibited zone completely (see orange zone in Figure 2). This allows us to recognize if the LCT's exploration is approaching a constraint, in which case the maximum action step in the solution space does not result in a change in the objective space that is larger than the size of the margin.

A countermeasure is triggered any time the LCT's exploration hits the margin zone by activating the corresponding backup policy. The backup policy's responsibility is to return

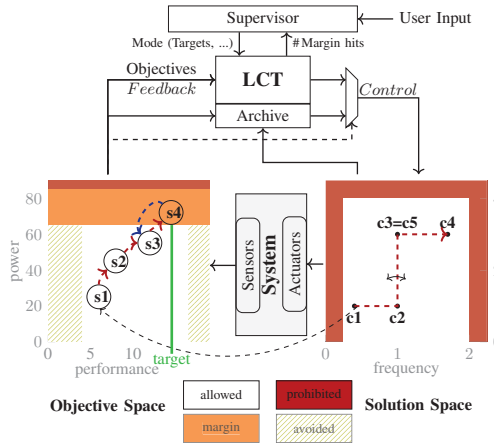


Fig. 2: Bounding the LCT's *objective space* to avoid undesirable or dangerous configurations

the system to the allowed zone. A deterministic action selection, e.g., maximum fitness within the match set, will only work for rulesets already trained for the currently executing workload. This might not necessarily apply, e.g., shortly after starting a newly installed application. Assuming each application initially begins in the allowed zone of the objective space, for any point in time there is a previous configuration in the solution space which results in a valid point in the objective space. So a backup policy reverting the latest executed action on the solution space will return to the allowed zone. However, because LCTs will exercise sub-optimal configurations while learning, there might have been a better configuration earlier than the most recent one. Bringing the system back to the best experienced configuration causes less deviation from the target than reversing the last action, which might result in oscillation between the allowed and margin zones.

Our backup policy is implemented as an archive, tracking the best so far experienced configuration for a workload. The archive applies the saved configuration to the overall system once it is activated by a margin hit.

Figure 2 shows an example sequence of LCT actions and archive resets. The initial configuration **c1** results in a performance **s1** far away from the objective. The LCT increases the frequency (configuration **c2**), which results in a higher performance (state **s2**) and hence gives a high reward which increases the fitness of this rule. As the new configuration is better than the previous one, the archive saves the current configuration and its corresponding objective value. Subsequently, the LCT distributes the workload over some more cores resulting in configuration **c3**. This action brings the system closer to the objective, but also nearer to the margin zone (state **s3**). Because a rule is only graded on the distance to the performance goal, the LCT is not aware of the proximity to the margin. In this case, **c3** replaces **c2** in the archive due to the

state improvement. Previously, a frequency increase resulted in a positive reward, so the LCT applies this action again (configuration **c4**)[1]. This time the frequency increase results in state **s4** in the margin zone of the objective space. This is indicated to the multiplexer in figure 2, which forwards the configuration saved in the archive (**c3**) to the actuators, which apply it to the system. The system returns to state **s3**, which represents the best so far experienced state. The LCT takes over again and the learning is regularly continued by further exploring the solution space.

C. Archive-Assisted Learning

Activating the archive once the margin is hit avoids constraint violations, by returning the system to a stable state using the backup policy. However, the backup policy does not prevent the LCT from repeatedly applying the rule that caused the margin hit. Even worse, if the reward assigned to the rule causing the margin hit does not penalize for triggering the backup policy, the rule might be considered desirable (i.e., high fitness). In Figure 2 the state **s4** is closer to the target than any other state, but it resides in the margin. Rule (**r3**) brings the system from state **s3** to **s4**, and receives a high reward. In this example, the learned behavior can result in oscillation, as rule **r3** will be applied repeatedly, alternating with archive triggers every other cycle. To discourage repeated margin hits, a rule causing an archive activation is assigned the minimum reward value, eventually preventing it from being preferred for its given state.

The purpose of margins and the archive are to avoid violations of constrains. Bounds, on the other hand, are not considered critical (as constraints are), and can tolerate some violation. In fact, allowing bound violations in the exploration phase gives LCTs the opportunity to explore new paths to beneficial configurations, which may not be discoverable otherwise. We support this by rewarding a rule with zero each time it hits an undesired zone, reducing the fitness, and hence the probability to be selected long-term. Therefore, our approach uses this method to adhere to bounds. It does not require additional hardware, and is not as rigid as the archive mechanism, but does not completely prevent violations of bounds when learning.

D. Benefits of Archive

The LCT augmented with the archive always applies configurations appropriate for the current system, because the configurations are derived from previous experiences. The archive overcomes issues caused by the probabilistic behavior of the roulette wheel selection algorithm of an LCT, as well as issues caused by unlearned models/behavior. Violations of bounds are allowed while exploring the solution space, but are prevented in the long term by assigning low rewards to rules that cause violations. We achieve all of this with simple dedicated hardware.

[1]This example assumes rules where the conditions are so general to match under all states mentioned.

IV. Case Study: DVFS Power Management

To evaluate the effectiveness of LCTs extended with an archive and a margin to prevent constraint violations, we deploy a hierarchical controller consisting of a supervisor and LCT to control core DVFS. The CPU core executes a multithreaded focus application with background tasks starting and stopping unpredictably to invoke disturbance. This mimics a typical use-case where a focus application requires some performance, while periodic tasks get executed in the background for synchronization and updates.

The application's performance requirement is represented by the objective function

$$\delta = \frac{|IPS - IPS_{target}|}{IPS_{max}}, \text{ subject to } P \leq P_{constraint}$$

with measured IPS (IPS) and power (P), the performance target (IPS_{target}), the maximal possible IPS value (IPS_{max}) for normalization and the power constraint ($P_{constraint}$). The objective function also contains a power constraint that represents a system requirement. The metrics necessary to evaluate the objective function are measured by hardware monitors.

The supervisor guides LCT behavior by defining the prohibited, avoided, and allowed zones. This is done by providing constraints, targets, and margin values for the objective function. By changing the LCT parameters at runtime, the supervisor can provide adaptivity, continuously coordinating the LCTs through their objective function such that their emergent behavior is aligned with the overall system goals.

The LCT's actions consist of increasing or decreasing the core frequency in variable step sizes. The conditions of the rules used to build the match set are performance in IPS, and utilization and frequency in % of their maximum possible values. These metrics are also measured by hardware monitors.

If the power constraint of the objective function is violated, the frequency is set to the value stored in the archive. The archive maintains the observed frequency setting that yielded the so far best objective function value most recently.

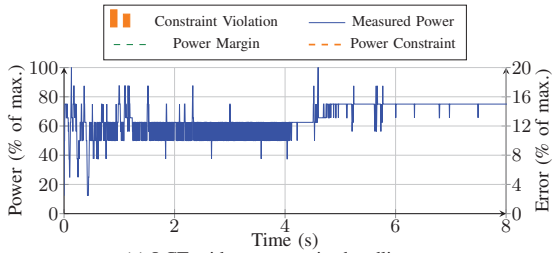
V. Evaluation

We evaluate the efficacy of our LCT augmented with an archive compared to the original LCT design from [15].

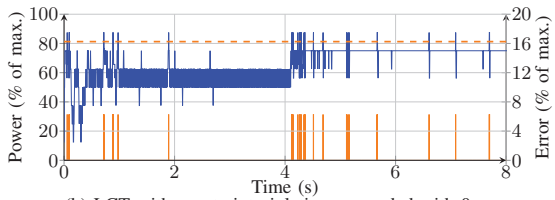
A. Experimental Setup

The platform described in our case study is implemented on a *Xilinx* Virtex®-7 FPGA. *Gaisler's* GRlib[2] is used as basis for the MPSoC running at 100 MHz and consisting of three Leon3 cores. One of the cores reports the measurement data to a host computer for later analysis, and another generates synthetic IP packets (emulating IO). The third core controlled by an LCT augmented by the previously introduced archive and margin mechanism. It handles three threads of the focus application, which mimics an IP-packet forwarding function. All monitors, as well as the LCT with archive are implemented using VHDL. The LCT has an invocation rate of 5 ms, whereas

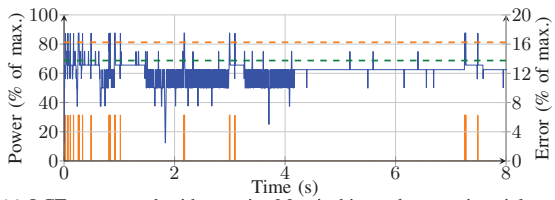
[2]https://www.gaisler.com/index.php/downloads/leongrlib



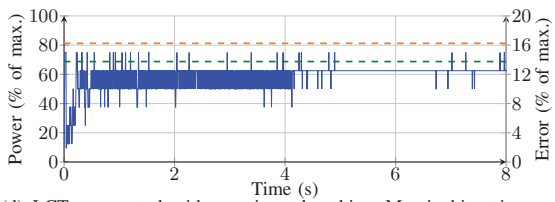
(a) LCT without constraint handling.



(b) LCT with constraint violations rewarded with 0.



(c) LCT augmented with margin. Margin hits and constraint violations both rewarded with 0.



(d) LCT augmented with margin and archive. Margin hits trigger backtrack.

Fig. 3: Ability of LCT controllers to manage a power constraint given a target IPS. In all cases, the target IPS is achievable with ~60 % power.

the fitness update and action selection together take only $(\#rules \cdot 2 + 2)$ clk cycles $= 2.58\,\mu s$ in our setup. The DVFS actions are carried out by pausing the processor's pipeline with a pulse density modulated signal. The IPS performance target is determined based on the packet throughput of the focus application. The supervisor sets a power constraint, but no bounds. After four seconds the supervisor adapts to the packet drop rate by increasing the IPS target. This behaviour represents a context switch.

In an LCT-per-core setup, the hardware necessary for our approach requires 10.6 % additional slices compared to an unaugmented MPSoC. Whereas, only one third of this overhead is necessary for learning and action decisions. The remaining slices are due to monitors and our DVFS actuator implementation, which are typically included in state-of-the-art MPSoCs.

B. Archive Evaluation

Figure 3 shows the ability of the LCT DVFS controller to honor a power constraint for four different controller implementations.

Figure 3a shows a basic LCT without a power constraint, or any restriction on its exploration of the solution space. This allows the controller to track IPS (purple dotted line in Figure 4) in the first four seconds. After the IPS target changes, the controller aims to increase the performance by maximizing the frequency, however, the IPS target is no longer achievable.

Figure 3b shows an LCT controller that applies a minimum reward to constraint violations. Observe that constraint (dashed orange line) violations occur throughout execution. This is because constraint violations are *required* for this controller to learn bad behavior, and subsequently avoid it. After the context switch by the supervisor, the LCT must learn again, requiring more constraint violations.

Figure 3c shows an LCT controller augmented with a margin (dashed green line) that applies a minimum reward to both constraint violations and margin hits. Observe that, although there are a number of constraint violations at the start, the more conservative reward assignment discourages the LCT from approaching the constraint, eventually preventing constraint violations. However, there are numerous constraint violations during initial exploration because, although the margin prevents any single-step actions from violating the constraint, the lack of an archive does not prevent the LCT from actuating toward the constraint after a margin hit. The IPS target increase does not cause any violations. However, the violations appearing near 7 seconds are caused by the roulette wheel selection. Graph 4 (red dash dotted line) shows how the introduced margin prevents the core from achieving the increased target IPS.

Figure 3d shows an LCT controller augmented with both a margin and archive. Observe that no constraint violations occur. This is because in addition to the margin preventing any single-step actions from violating the constraint, the archive additionally guarantees that the action immediately following a margin hit will revert *away* from the constraint. The context switch does not cause any violations, because the augmented LCT prioritizes the power constraint over the IPS target. Again, the margin prevents the core from achieving the IPS target.

Figure 4 shows the measured IPS (moving average) of all four LCT implementations. Observe that the LCT augmented with archive and margin not only approaches the reference quickly, but also continues to stay the closest compared to the other setups. Furthermore, the distance to the reference decreases constantly, whereas the other setups continue exploring disadvantageous configurations intermittently.

Our experiments demonstrate that an LCT augmented with an archive and a margin zone allows us to recognize ap-

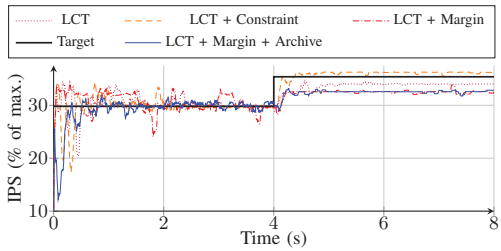


Fig. 4: The LCT + Margin + Archive (blue line) approaches the target faster and stays closer to it than the other controllers.

proaching constraints, and lets us counteract in an appropriate manner to prevent any constraint violations. Further, our approach deviates least from the target once in steady-state. The experiments show that preventing constraint violations comes at the cost of sacrificing the performance target. This is due to correctly prioritizing constraints over achieving performance targets. The comparison in Figure 3a and Figure 3b shows the possible necessity of violating constraints in order to discover desirable configurations.

VI. Conclusion and Future Work

We demonstrate the ability to dynamically manage multicore processor performance targets while adhering to a given power constraint by means of a hierarchical controller exploiting machine learning and reflective supervision. Experimental use cases revealed that archive-based backup policies predictively cope with potentially critical actions and, thus, represent trustworthy and effective means for obeying to critical system constraints. These experiments were conducted for a packet forwarding application running on an FPGA evaluation board containing three SPARC-V8 cores. In general, the fusion of rule-based reinforcement learning with reflective supervision allows the hardware/software control layers of multicore processors to self-adapt under changing environmental or workload dynamics. This is comparable to how biological organisms or organizations adapt through evolution. However, our multicore processors do this within seconds rather than centuries, which is the typical time period in nature.

Current work focuses on extending the applicability of LCT-based reinforcement learning for controlled memory approximation on a system with multiple on-chip cache levels and off-chip main memory. Furthermore, we investigate the opportunity for the evolutionary generation of new condition-action pairs during runtime through refinement of existing rules with proven track records.

VII. Acknowledgments

We would like to thank our partners from TU Braunschweig, Rolf Ernst and his team, for the excellent cooperation within the IPF project and acknowledge the financial support from the DFG under Grant HE4584/7-1 as well as the NSF under Grant CCF-1704859.

References

[1] J. Fromm, in *The Emergence of Complexity*. Kassel University Press, 2004.
[2] P. Valckenaers *et al.*, "Emergent short-term forecasting through ant colony engineering in coordination and control systems," *Advanced Engineering Informatics*, vol. 20, no. 3, pp. 261–278, 2006.
[3] A. Attanasi *et al.*, "Emergence of collective changes in travel direction of starling flocks from individual birds' fluctuations," *Journal of The Royal Society Interface*, vol. 12, no. 108, p. 20150319, 2015.
[4] C. K. Hemelrijk *et al.*, "Emergence of oblong school shape: models and empirical data of fish," *Ethology*, vol. 116, no. 11, pp. 1099–1112, 2010.
[5] M. Gardner, "Mathematical Games - The fantastic combinations of John Conway's new solitaire game 'life'," vol. 223, no. 4, 1970, pp. 120–123.
[6] A. J. Lotka, "Elements of physical biology," *Science Progress in the Twentieth Century (1919-1933)*, vol. 21, no. 82, pp. 341–343, 1926.
[7] J. Zeppenfeld *et al.*, "Applying autonomic principles for workload management in multi-core systems on chip," in *Proceedings of the 8th ACM international conference on Autonomic computing*. ACM, 2011, pp. 3–10.
[8] A. M. Rahmani *et al.*, "SPECTR: Formal Supervisory Control and Coordination for Many-core Systems Resource Management," in *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2018.
[9] A. Jantsch *et al.*, "Self-awareness in systems on chip—a survey," *IEEE Design & Test*, vol. 34, no. 6, pp. 8–26, 2017.
[10] J. O. Kephart *et al.*, "The vision of autonomic computing," *IEEE Computer*, vol. 36, no. 1, pp. 41–50, 2003.
[11] H. Hoffmann *et al.*, "A generalized software framework for accurate and efficient management of performance goals," in *Proceedings of the International Conference on Embedded Software, EMSOFT 2013, Montreal, QC, Canada, September 29 - Oct. 4, 2013*, 2013, pp. 19:1–19:10.
[12] S. W. Wilson, "Classifier fitness based on accuracy," *Evolutionary computation*, vol. 3, no. 2, pp. 149–175, 1995.
[13] S. Edenhofer *et al.*, *Trust Communities: An Open, Self-Organised Social Infrastructure of Autonomous Agents*. Springer International Publishing, 2016, pp. 127–152.
[14] E. A. Rambo *et al.*, "The information processing factory: A paradigm for life cycle management of dependable systems," *ESweek*, 2019.
[15] J. Zeppenfeld *et al.*, "Learning classifier tables for autonomic systems on chip," *INFORMATIK 2008. Beherrschbare Systeme-dank Informatik. Band 2*, 2008.
[16] B. Donyanavard *et al.*, "Sosa: Self-optimizing learning with self-adaptive control for hierarchical system-on-chip management," in *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2019.
[17] S. W. Wilson, "Zcs: A zeroth level classifier system," *Evolutionary computation*, vol. 2, no. 1, pp. 1–18, 1994.
[18] J. Zeppenfeld *et al.*, "Autonomic workload management for multi-core processor systems," in *International Conference on Architecture of Computing Systems*. Springer, 2010, pp. 49–60.
[19] J. Garcia *et al.*, "Safe exploration of state and action spaces in reinforcement learning," *Journal of Artificial Intelligence Research*, vol. 45, pp. 515–564, 2012.