SCIENCE CHINA
Information Sciences

January 2024, Vol. 67, Iss. 1, 111101:1–111101:15
https://doi.org/10.1007/s11432-022-3826-6

• REVIEW •
From CAS & CAE Members

# Categorizing methods for integrating machine learning with executable specifications

David HAREL[1*], Raz YERUSHALMI[1], Assaf MARRON[1] & Achiya ELYASAF[2]

[1]*Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot* 76100*, Israel;*
[2]*Department of Software and Information Systems Engineering, Ben-Gurion University of the Negev, Beer Sheva* 8410501*, Israel*

Received 30 September 2022/Revised 11 April 2023/Accepted 7 July 2023/Published online 17 October 2023

**Abstract** Deep learning (DL), which includes deep reinforcement learning (DRL), holds great promise for carrying out real-world tasks that human minds seem to cope with quite readily. That promise is already delivering extremely impressive results in a variety of areas. However, while DL-enabled systems achieve excellent performance, they are far from perfect. It has been demonstrated, in several domains, that DL systems can err when they encounter cases they had not hitherto encountered. Furthermore, the opacity of the produced agents makes it difficult to explain their behavior and ensure that they adhere to various requirements posed by human engineers. At the other end of the software development spectrum of methods, behavioral programming (BP) facilitates orderly system development using self-standing executable modules aligned with how humans intuitively describe desired system behavior. In this paper, we elaborate on different approaches for combining DRL with BP and, more generally, machine learning (ML) with executable specifications (ES). We begin by defining a framework for studying the various approaches, which can also be used to study new emerging approaches not covered here. We then briefly review state-of-the-art approaches to integrating ML with ES, continue with a focus on DRL, and then present the merits of integrating ML with BP. We conclude with guidelines on how this categorization can be used in decision making in system development, and outline future research challenges.

**Keywords** machine learning, artificial intelligence, grey box learning, domain knowledge, rules, behavioral programming, deep reinforcement learning, survey

## 1 Introduction

Our world depends increasingly on complex systems in medicine, transportation, security, manufacturing, and many more fields. These systems have to make sensitive and safety-critical decisions, often autonomously, with little or no human intervention. A game-changing breakthrough in system development was afforded with the advent of deep learning (DL); see, for example, papers surveying the application of machine learning (ML) in autonomous vehicles (AVs) [1, 2]. Despite the many advances, both in ML and in software and systems engineering in general, there are major issues to be addressed before the techniques for developing trustworthy systems can become routine engineering practice [3, 4].

In [5], additional challenges associated with learning real-world behavior are discussed, associated, among other things, with (i) the difficulty of obtaining relevant real-world data and behavior from which to learn, and (ii) the tight real-time constraints on both the learning process and the runtime operation.

Some systems built with ML deal with such challenges by adding rules and procedures anchored in the domain knowledge of human experts. A central issue that thus arises involves the various ways to design hybrid decision systems that consider both observed data and domain-expert knowledge [3, 4, 6]. More specifically, on the one hand, data-driven solutions that rely on deep machine learning (DML) are becoming ever stronger for solving hard problems in real-world applications, but they present challenges

ORCID IDs: David Harel: 0000-0001-7240-3931; Raz Yerushalmi: 0000-0002-0513-3211; Assaf Marron: 0000-0001-5904-5105; Achiya Elyasaf: 0000-0002-4009-5353.

* Corresponding author (email: dharel@weizmann.ac.il)

in explainability, verification, and maintainability. On the other hand, domain-expert knowledge, represented as executable specifications (ES), does offer explainability, ease of enhancement, and amenability to formal analysis. ES can be defined using classical procedural programming, structural and behavioral modeling, rule-based systems, specialized formalisms (such as Statecharts [7], scenario-based programming [8–10], temporal logic [11]), and more. Still, the required expertise in the problem domain is costly and hard to acquire.

Working with ES is also challenged by the fact that even when the expert knowledge is available, translating requirements specified in natural language by domain specialists into computer-executable languages is a difficult and error-prone engineering task. When compared, some DML-based solutions achieved better results than algorithms crafted carefully by domain experts using ES techniques [12,13]. Some of the difficulties are elucidated by the limited success of early knowledge-based expert systems. In [14], LeCun, Bengio, and Hinton wrote: "Ultimately, major progress in artificial intelligence will come about through systems that combine representation learning with complex reasoning...".

Despite agreement on the desirability of the direction, best practices for such hybrid ML-and-ES designs are not yet commonly considered to be part of the software-engineering toolbox for AI-based systems [15].

This paper aims to contribute to a foundation for such methods and design approaches. Specifically, our main contributions are:

(1) A categorization and analysis of ML-ES hybridization approaches.

(2) Guidelines for utilizing this categorization in system development and maintenance.

(3) A spotlight on the relevance of the choice of an ES language to some ML-ES integration approaches.

The paper is structured as follows. We begin by describing the terminology in Section 2. In Sections 3 and 4, we categorize and briefly review approaches to combining ES with ML and deep reinforcement learning (DRL). In Section 5, we review recent contributions and work in progress in hybrid DRL and ES designs. In Section 6, we provide guidelines for utilizing this framework in new/existing systems. We conclude in Section 7.

## 2 Taxonomy

To differentiate the approaches for combining ML with ES, we use the following taxonomy.

**ES.** As previously explained, we consider any domain-specific specifications that can be executed, including classical procedural programming, structural and behavioral modeling, rule-based systems, and specialized formalisms. In this paper, we focus on the integration of domain knowledge, represented using ES, with ML. Thus, domain-independent improvements to ML algorithms, such as a novelty-based intrinsic reward for improving DRL explorations [16,17], are outside its scope. Another field that is out of the scope of this paper is imitation learning (IL), where given a dataset of the expert demonstrations, IL tries to imitate the expert behavior for the range of presented scenarios. Although IL integrates domain knowledge, it does not utilize ES for this task. A survey on IL can be found at [18,19].

**Core technology.** The central, core technology of the system, whether it is based on ML, ES, or both. Consider, for example, an ML-based image recognition system; its core technology is, of course, ML. Nevertheless, if this system is integrated as a module in the software of a robot developed by programmers, we would say that the core technology of the robot is an ES. The exact boundaries of this definition are not formally defined, and often the nature of the core technology is in the eyes of the beholder. Defining the core technology this way allows us to describe the integration types from several views, thus supporting different types of systems. This observation is also true for the following definitions.

**ES/ML role.** The role of the ES or ML in the system. In the example of an ML-based image recognition module that is part of a robot, the role of the ES is the system implementation. More specifically, its role is to assemble the different modules, ES and ML modules. The role of ML is a module implementation. There are other types of roles, as elaborated below.

**Integration phase.** The phase in which the executable specifications are integrated with ML. This can be during the runtime of the system or during the training of the ML model. Some of the runtime integration approaches may have implications on the design/construction of the system. For example, constructing an ES-based system, which utilizes ML at runtime to tune parameters of specific behavior, requires special considerations at design time.

**Table 1** The integration approaches (Sections 3–5) with respect to the taxonomy defined in Section 2

| Category section | Core tech. | ES role | ML role | Integration phase | Integration technique |
|---|---|---|---|---|---|
| 3.1 Domain-specific task decomposition and component assembly | ML/ Hybrid | Assemble ML and ES components | Module impl. | Design, runtime | Assemble the trained model using domain-specific knowledge |
| 3.2 Enforcing behavior during inference | ML | Enforce behavior | System impl. | Runtime | Override the ML inference using, e.g., rules or formal specifications |
| 3.3 Enforcing behavior during training | ML | Enforce behavior | System impl. | Training | Use ES to test and verify that ML meets the specification, retrain otherwise |
| 3.4 Rule-based data enhancement of ML training | ML | Enhance training data | System impl. | Training | Enrich the raw data or interim results |
| 3.5 Enriching model representation using ES | ML | Enhance training model | System impl. | Training | Incorporate the domain knowledge within the model representation |
| 3.6 ML-based enhancement to ES-based systems | ES | System impl. | Improve ES | Design, runtime | Endow local elements with learning capabilities |
| 4.1 Using ES in a DRL reward function | DRL | Reward shaping | System impl. | Training | Incorporate domain knowledge in, e.g., rules or a probabilistic risk model |
| 4.2 Using ES for environment simulation | DRL | Environment simulation | System impl. | Training, runtime | Specify a simulator that utilizes domain knowledge, such as physics |
| 4.3 Using ES for agent training and representation | DRL | Enhance training model | System impl. | Training | Incorporate domain knowledge within the agent model to enhance training |
| 4.4 Applying domain knowledge to state representation | DRL | State representation | System impl. | Training, runtime | Use domain knowledge to abstract the actual state space (e.g., pixels) with higher-level properties |
| 4.5 Applying domain knowledge to safe learning | DRL | Ensure safety | System impl. | Training | Constrain unsafe actions during exploration |
| 4.6 Languages for DRL knowledge specification and reward shaping | DRL | Reward shaping languages | System impl. | Training | Languages (not an integration approach) |
| 5.1 Labor division with movable walls: composing ES with ML | Both | Integration methodology | System impl. | Design, training, runtime | An agile methodology for developing complex intelligent systems |
| 5.2 Using BP with SMT constraint resolution, context, and deep reinforcement learning for a simplified RoboCup-type game | ES | System impl. | Improve ES | Design, runtime | Adapt the specification's parameters with DRL |
| 5.3 Augmenting deep neural networks with scenario-based guard rules | DRL | Enforce behavior | System impl. | Runtime | Override ML inference using BP specifications that are amenable to verification and ensuring both safety and liveness |
| 5.4 Scenario-assisted reward-shaping for deep reinforcement learning | DRL | Reward shaping | System impl. | Training | BP specifications of requirements and safety constraints, used as a context-aware reward shaping function |
| 5.5 Constrained reinforcement learning for robotics via scenario-based programming | DRL | Ensure safety | System impl. | Training | The ES-imposed costs (like Subsection 5.4) is separated from the DRL reward function |

**Integration technique.** The technique used for the integration. For example, the ES can assemble ML models using domain-specif knowledge, override the inferences of the ML model.

Table 1 summarizes the various integration approaches described in the following Sections 3–5 with respect to this taxonomy. Figure 1 illustrates high-level examples of integration approaches.

We exclude from the tabular categorization of integration approaches the use of AI and ML for automating the generation of ES from various forms of non-executable specifications, such as natural language requirements or examples of simulation logs. Intuitively, such code generation, and AI-assisted development in general, exists in a different realm, distinct from considerations of architectural integration, and deserves a separate analysis, especially in view of rapid developments in this area in the wake of the proliferation of large language models [20–22].

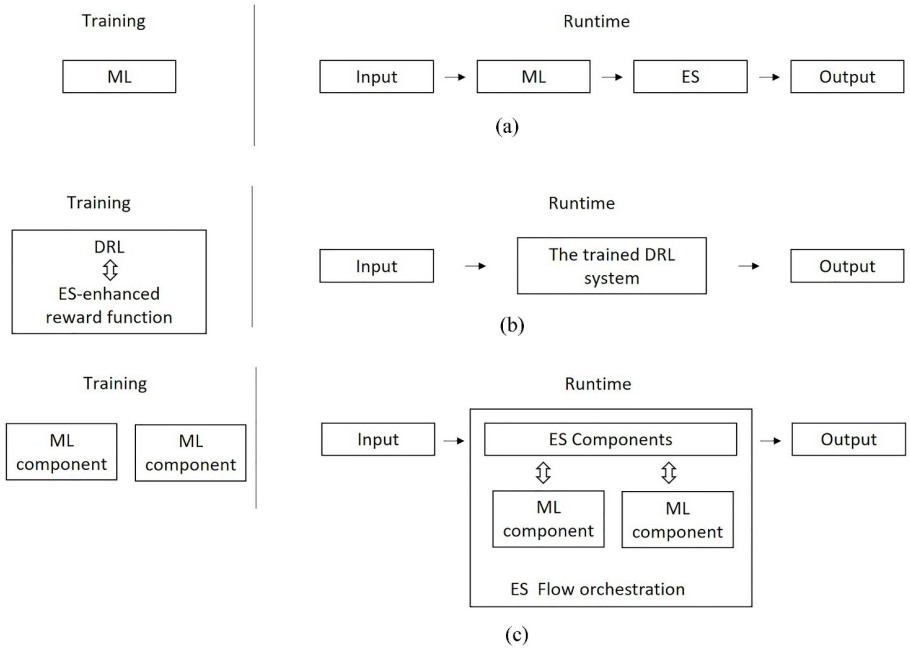



**Figure 1** High level illustration of selected examples of ML-ES integration approaches. (a) 3.2 Enforcing behavior during inference; (b) 4.1 using ES in a DRL reward function; (c) 3.6 ML-based enhancement to ES-based systems.

## 3 ES and ML – a bird's eye view

In this section, we highlight existing techniques for incorporating ES driven by domain knowledge into systems based on ML. As will be clear from the cited examples, the boundaries between the categories listed here are not formally defined; furthermore, several such approaches may be used in a given system.

### 3.1 Domain-specific task decomposition and component assembly

Domain knowledge can be applied as a design heuristic, influencing how the overall problem is divided into sub-tasks, and the flow of data between ML and/or rule-based components. For example: in [23], a complex problem of tracking multiple moving targets is solved by the composition of a number of neural nets, where the division of work and the composition mechanism are driven by domain knowledge. Specifically, certain recurrent neural networks (RNNs) carry out time-related inferences and state management, while others (in this case, long short term memory (LSTM) networks) are used within each camera frame for the combinatorially difficult task of data association – tying observations to objects.

The authors of [24] decomposed the problem into sub-tasks, focusing specifically on human-understandable concepts, towards streamlining the argumentation of safety issues. In [25], a finite state machine is used to control the high-level steps in an AV's lane change and cut-in behavior, and the behavior within each step is learned.

Such a division of work and composition of components may also include redundancy, performing the same task using more than one approach, and employing voting or other interweaving techniques to reach a cohesive system behavior; see, for example, Python Ensembles [26]. The choice of which algorithm (ML or ES) to use for each of the agents in the ensemble can be domain dependent.

### 3.2 Enforcing behavior during inference

An approach that stands out among the ways to incorporate domain knowledge into ML applications is the one in which a "black box" ML-based solution is "wrapped around" by knowledge-based rules that constrain the system's action or even enforce particular outcomes in certain situations.

For example, in [27], the decisions of an ML-based controller of an AV are subjected to overriding domain-specific rules of safety and responsible driving. A sample rule would be "when the AV follows another vehicle, it must make sure that the distance between the two vehicles is greater than $X$", where a formula for the safety distance $X$ is provided.

In [28], a similar technique is used: an ML component controls most of the behavior, a baseline-verified procedural safety controller concurrently computes verified safe behavior, and a third controller dynamically switches between the two, as needed.

### 3.3 Enforcing behavior during training

While override rules may enforce a behavior for a certain input, they do not guarantee the correct behavior for all possible inputs. To verify that a system meets certain requirements, the problem-specific requirements must obviously be specified formally. For example, in [29], such requirements are provided as input to verification tools, and repair actions are taken during training when the ML-based system does not meet a requirement.

In [30, 31], formal scenario-based specifications are used for automated verification and test-case generation for training ML-based systems.

### 3.4 Rule-based data enhancement of ML training

In the preceding sections, the integration of ES and ML was carried out after the inference of the ML model. In other approaches, the integration is in the training phase of developing an ML-based solution.

The most intuitive way to apply rules in training is rule-based data enrichment, where the raw data or interim results are processed and enriched using domain-specific rules. The output is then fed to subsequent learning processes.

For example, in [32], the ML process is broken down into a feature-mapping phase and a global learning phase. Between these two phases, a rule-based step enriches the interim data by domain-specific processing focused on the interaction between feature-based elements; the approach is demonstrated therein by electrocardiogram analysis. In [33], an application for sentiment analysis in text augments the ML data processing using linguistic rules to tag text elements with sentiment-related features. In [34], an application for processing text in images is described; convolutional neural networks (CNNs) are used to identify relevant regions. Candidate locations of the text are determined with the assistance of domain-specific rules, and the text recognition is guided by a vocabulary extracted from image metadata and other available context information. In [35], recognition of text in images is enhanced using a semantic module and a language model for enriching the intermediate "bottleneck code" in an encoder-decoder ML architecture.

Clearly, with or without data enrichment, all supervised learning where the data set used is labeled, involves domain knowledge, by the very fact that data is labeled. To the extent that some of this knowledge can be automated to accelerate the training, this would constitute an additional aspect of the integration of ES and ML. For example, in [36], a system is trained on a labeled data set, but to accelerate the training process, the data set is iteratively and automatically enhanced with examples derived from instances that the evolving ML solution has previously classified incorrectly.

In [37], also in supervised learning, the feedback provided by the teacher to the learner, like the loss function, is enhanced with information resulting from domain-specific rules.

### 3.5 Enriching model representation using ES

Hu et al. [38] transformed expert knowledge, represented using declarative first-order logic rules, into the weights of neural networks. This is achieved by forcing the network to emulate the inferences of a rule-regularized teacher, and evolving both models iteratively throughout the training. The teacher network is constructed in each iteration by adapting the posterior regularization principle in a logical constraint setting that provides a closed-form solution.

We provide more DRL-specific examples in Subsection 4.3.

### 3.6 ML-based enhancement to ES-based systems

An approach dual to that of Subsection 3.4, where ML solutions are enhanced through data enrichment using domain knowledge, is the one where we augment (largely) procedural solutions by endowing low-level or local elements with learning capabilities. For example, in [39], a sophisticated adaptive multi-agent system is enhanced by enabling each individual agent to enhance its own narrow behavior using ML. Also, in [40], a self-adaptive MAPE-K-based system accelerates its exploration of adaptation spaces – namely, the exploration of the possible actions at any state – using ML.

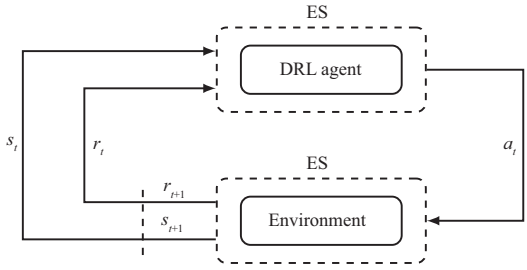



**Figure 2** A combined view of approaches for integrating ES and DRL. With ES wrapping the environment, it can be used for: shaping the reward function (Subsection 4.1), simulating the environment (Subsection 4.2), state representation (Subsection 4.4). With ES wrapping the agent, it can be used for agent representation (Subsection 4.3) and safe learning (Subsection 4.5).

# 4 ES with DRL

In this section, we review select approaches to the integration of ES with DRL. Notably, some of the techniques described in Section 3 are applicable to DRL (e.g., Subsection 3.2), so in this section, we focus on integration types that are unique to DRL, and which also integrate domain-specific knowledge. In Section 5, we further narrow down the review to our own work in this area.

A survey concentrating on the application of expert knowledge in the area of DRL for robotic assembly tasks appears in [41]. This survey states the main goals of such an integration, which can be readily generalized to other application areas: (i) guiding and constraining the exploration process, and (ii) decomposing the tasks and process into human-understandable components. These goals can both accelerate the learning process and yield a more interpretable behavior policy.

Figure 2 depicts possible ways to integrate ES with DRL, as we now discuss.

## 4.1 Using ES in a DRL reward function

By its very nature, training a DRL system requires some incorporation of domain knowledge since the reward function must know how good or bad each action is under every condition or state. For example, in training an AV controller, the reward function may have access to the anticipated actions of a human driver under various conditions, as in "if the distance to leader car is now less than 2 seconds, a human would have pressed the brake by now". Or the reward function may assess how desirable or undesirable each state is, as in "if the distance between two cars is zero and they are moving towards each other (i.e., a collision occurred), this is bad". Another case, as in [42], is where the DRL reward function may compute the reward based on a rich risk model with a variety of road conditions and predictive probabilities of dynamic changes therein.

## 4.2 Using ES for environment simulation

DRL depends on having the agent interact with a realistic environment, augmented by reward-function feedback. Learning in a real-world environment is expensive and risky and is thus often preceded by extensive learning under simulated conditions. In building such simulators, it is only natural to encode human assumptions about the environment's behavior as ES.

For example, in [43], the DRL of an AV agent learns in an elaborate environment simulator (VISSIM)[1)], which offers rich programmable control over the environment's behavior scenarios that are presented to the learning agent.

Additional simulators are listed in [44], which also provides a glimpse into the kind of domain knowledge involved in a particular problem. In this case, negotiating a variety of road intersection configurations.

Not all model-and-knowledge-based DRL environments are programmed with ES. For example, in [45], where an agent learns to play a variety of Atari games, the environment's behavior is first learned in a preceding step, using a CNN, from actual screen images. The learned environment behavior is then executed and simulated in the agent's learning process.

The design of the particular CNN in [45] employs domain knowledge, as it is carefully crafted to fit the visual and reactive effects of Atari games, such as image resolution and sampling rate. Nevertheless, that domain knowledge is not reflected in behavioral rules during simulation.

1) https://www.ptvgroup.com/en/solutionsproducts/ptv-vissim/.

### 4.3 Using ES for agent training and representation

Zhang et al. [46] developed a knowledge-guided policy network called KoGuN that combines human prior suboptimal knowledge with reinforcement learning. Human (suboptimal) knowledge is represented by a fuzzy rule controller that is fine-tuned using a refine module. The combination of the knowledge controller and the refine module provide a "warm start" and accelerate learning.

Cao et al. [47] proposed a generalizable logic synthesis framework (GALOIS) to synthesize hierarchical and strict cause-effect logic programs. It uses a white-box program as the policy to interact with the environment and collect data for policy optimization. Specifically, the state is encoded as a set of ground atoms that are given to the agent, which represents a synthesized program. The agent outputs a predicate that is decoded back to an action. GALOIS adds high-order intelligence (e.g., logic deduction and reuse) to DRL, providing it with an advantage over mainstream baselines in terms of asymptotic performance, generalizability, and knowledge reusability across different environments.

### 4.4 Applying domain knowledge to state representation

In a DRL setting, an agent's learned behavior is a mapping from each state to an action or a set of possible actions, each with its own probability. To take relevant actions, the agent must first consolidate all its sensory inputs with the knowledge of its own internal state, in order to determine the current state. Since sensory inputs from the real world, like images of an urban or a highway scene, are never identical; state abstraction, or clustering, must take place. This process often relies heavily on domain knowledge.

For example, in [48], domain knowledge is used to group states that are similar to each other by focusing on higher-level properties like road configurations, e.g., the number of lanes, and object relationships, as defined, say, by the distances and directions of cars, or by the identity of the lane that a given car happens to be in.

### 4.5 Applying domain knowledge to safe learning

One of the problems in ML solutions, and in reinforcement learning in particular, is safety during the learning/training process, when the process is carried out in the real world and not on a simulator. As the agent explores the applicability of certain actions in certain states, it may make unsafe choices.

The extensive survey in [49], which discusses general safety issues in ML and DRL, concentrates in part on the issue of using safety constraints to guide RL exploration. The survey focuses on the mathematical techniques of incorporating models or actual data about safety and risks, but does not expand on the methods and languages used to specify or obtain those models or data.

### 4.6 Languages for DRL knowledge specification and reward shaping

A specific relevant research area involves the language used to specify the domain knowledge for reward shaping; i.e., the enriching and refining of reward values and setting the points in time when rewards are provided to the learner in order to accelerate the convergence of the learning process.

For example, the authors of [50] proposed a specification language designed especially for the compositional specification of complex control tasks.

An additional language for reward shaping is presented in Subsection 5.4.

## 5 Behavioral-programming specification with ML

In this section, we review our own contributions to hybrid ES and ML designs. Subsection headings are adapted from the titles of the main cited articles.

All of these efforts are based on a specific ES paradigm called behavioral programming (BP) – a modeling approach focused on allowing users to specify the system behavior in a natural and intuitive manner that is aligned with how humans perceive the system requirements [8–10]. A behavioral program consists of a set of scenarios (that state what to do) and anti-scenarios (that state what is not allowed to happen), which are interwoven at runtime, yielding cohesive behavior.

Integrating BP and ML has several merits. The key difference between BP and other ES languages is its alignment with the requirements, which allows domain-expert to express their knowledge in an intuitive and agile manner that is aligned with the desired behavior as they perceive it. In addition, BP's

semantics allows for applying formal methods for verifying properties of the model. Another advantage is the ability to traverse the state-space of the model and generate sequences of executions for, e.g., testing the correctness of the ML model.

### 5.1 Labor division with movable walls: composing ES with ML

In [6], the authors propose an approach for integrating AI and ES techniques in developing complex, intelligent systems, in a way that can simplify agile/spiral development and the maintenance processes. More specifically:

(1) Provide a set of explicit goals for the system, with the ability to check whether or not they are achieved in a given system run.

(2) When such a check fails, succinctly describe the gap as a new requirement. For illustration, consider a sports coach or a driving instructor working with an advanced student. Following a student's action that seems to the instructor to be a mistake, the latter adds ever more refined goals and rules to improve the student's future performance.

(3) As goals are refined and requirements added, as well as when bugs are discovered, add new components that precisely address the gap between the revised goals and what the existing system accomplishes.

(4) Aim at an architecture in which independent ES-only components can be readily composed with independent AI-only components, where each AI or ES component addresses a particular (sub-)goal without directly interfacing with others.

(5) Repeatedly identify opportunities for completely, or at least partially, replacing AI-based components with ES ones. This kind of replacement can be triggered by many things, such as:

• The emergence of a better understanding by engineers and stakeholders of the elements of a particular goal.

• A demand by engineers or regulators for better visibility of the logic or rationale of specific system behavior.

• A performance issue that must be corrected.

### 5.2 Using BP with SMT constraint resolution, context, and DRL for a simplified RoboCup-type game

In [51], four BP-based controllers are specified for a player in a simplified RoboCup-type game. In one of the implementations, the BP-based specifications were improved using DRL in a simulated environment that mimics the real physical game environment. Specifically, the four implementations are:

(1) A simple BP-based controller of the robotic player, with scenarios that guide it as to what actions to take under various conditions.

(2) An extension of the first implementation that uses the Z3 SMT solver to provide a composition of behavioral modules based on constraint resolution. For example, in a given state, different scenarios can impose different constraints on the angle at which the ball should be kicked, and the solver will find (or compute) the scenario-driven action that complies with all these constraints. This allows for reaching cohesive composite behavior by incremental development with independent or loosely coupled behavioral components.

(3) An extension of the first implementation with a set of idioms that subject the executable behavioral specifications to context. For example, the behavioral requirements and constraints are different in the following two contexts: (i) the ball is free in the field, and (ii) the ball is held by the opponent. The idioms enable the use of context information to parameterize multiple general behavior scenarios (like "go to a target") dynamically and naturally.

(4) An extension of the first implementation, but with scenarios being endowed with adaptive capabilities based on reinforcement learning. For example, when a scenario specifies what action a robo-player should take when it is too far from the ball (say, "increase speed"), the parameter defining "too far" is learned. While requiring a training session, this allows the ES to be more general and more intuitive.

A similar approach to [51] has been done in [52], where reinforcement learning has been applied to refine the execution mechanism and action selection of BP so that it will achieve certain goals more efficiently.

### 5.3 Augmenting deep neural networks with scenario-based guard rules

In [53], a DNN model is "surrounded" by BP specifications and uses the BP rules to override decisions made by the DNN model when certain criteria are met, as described in Subsection 3.2 above. The architecture is as follows:

(1) Override the DNN decisions using a BP policy is implemented by having anti-scenarios that represent undesired actions emanating from the DNN.

(2) The BP runtime algorithm is modified so that each request behavior is associated with the confidence level of the DNN, and that among all possible behaviors, the one with the highest confidence score is selected.

(3) The DNN presents itself to the rest of the scenario-based model through a wrapper scenario object ODNN. This scenario repeatedly waits for environment events that are mapped to DNN inputs and which can trigger DNN activation. When such an event arrives, ODNN waits for the DNN to complete its computation and then requests all events that are part of the DNN output and have a nonzero score or probability.

(4) Other behavioral scenarios can then request additional events and/or block events requested by ODNN.

(5) During execution, when no additional scenarios exist, the system selects the event with the highest score. If the DNN assigns the highest score to an event that is currently blocked by the BP specification, at runtime, the selected event will be the one representing the output with the next-highest score, but which is not blocked.

This setup enables dealing with both safety and liveness requirements; that is, not doing "bad" things and eventually doing "good" things. It is also amenable to formal verification.

### 5.4 Scenario-assisted reward-shaping for DRL

In [54], the authors propose an integration of BP and DRL using a reward-shaping approach that penalizes the agent when rules are violated, with an unconstrained optimization method. The authors use ES to express behavioral requirements and safety constraints and apply these to a DRL training process. More specifically:

(1) The behavioral and safety requirements are specified as BP scenarios; anti-scenarios are used to indicate undesired behavior.

(2) When an action chosen by the DRL agent is blocked by any scenario, a fixed penalty is directly reduced from the accrued reward.

The results show that the agent learns to reduce violations of the BP-specified rules substantially.

### 5.5 Constrained reinforcement learning for robotics via scenario-based programming

In [55], the authors use ES to express the behavioral and safety requirements and apply those to a DRL training process, similarly to [54]. However, here it is done while keeping the ES-imposed costs separate from the DRL reward function.

More specifically:

(1) The behavioral and safety requirements are specified as BP scenarios, where BP event blocking is used to indicate undesired behavior, similarly to [54].

(2) The cost calculation uses a reflection mechanism that checks which events are blocked by which BP scenarios at any given point in time, and when the DRL agent takes an action that is blocked by some BP scenario, a related, separate scenario-specific cost component is incremented.

(3) Each scenario is also assigned an upper bound for its costs (namely, the number of allowed violations).

(4) The separate cost components are used by the learning process as computational constraints, where the goal is now (i) to maximize the reward, and (ii) to satisfy all the cost constraints, keeping the number of violations within the allowed bounds, and when this is not possible, to minimize each of the scenario-specific costs. For this purpose, the authors introduce an optimization of the Lagrangian/PPO DRL training method [56].

This separation between reward value and different costs saves the designers the difficult, if not impossible, task of manually converting all aspects of desired and undesired behavior into a single numerical scale, as is typically done in reward shaping.

Thus, the key advantages of the approach introduced here over that of [54] are:

• It adopts a constraint-driven DRL framework that differentiates between optimizing the main reward and minimizing costs. This is in contrast to [54], where a single reward penalty is determined ad-hoc by an unspecified method, and is given when the desired property is violated.

• It defines constraint thresholds independently for each rule/property and handles multiple such constraints in a uniform way. The method in [54] only allows a global minimization of the total cost.

The results in [55] show a promising method of incorporating BP-based domain-expert knowledge, which defines constraints and requirements, directly into the DRL training loop, thus effectively modifying the learned agent policy so that it complies to a high degree with the specified constraints and requirements.

### 5.6 From requirements to source code: evolution of behavioral programs

The following is related to the integration of ES with artificial intelligence in general – not specifically ML. We describe an approach for generating behavioral specifications by means of genetic programming (GP). GP is a method for iterative optimization of the functionality of a computer program. While the methods presented here do not directly rely on DL, GP is considered to be a form of artificial intelligence [57], and the natural selection process of evolution in the biosphere which inspires GP is considered a form of learning [57–59].

In [60], GP has been used for generating complete behavioral programs from scratch. The authors defined grammar for context-aware behavioral programs [61] that allowed them to represent behavioral programs as trees. They also designed viable and effective genetic operators (i.e., mutation and crossover) that utilize fault-diagnosis techniques [62]. To demonstrate the approach, they evolved complete programs from scratch of a highly competent O player for the game of tic-tac-toe. While it may seem like a simple task, automatically generating executable code has a long history of arguably modest success, mostly limited to the generation of small programs of up to 200 lines of code and genetic improvement of existing code. Here, the evolved programs were longer, well structured, consisting of multiple, explainable modules that specify the different behavioral aspects of the program and are similar to handcrafted programs. To validate the evolved programs' correctness, they utilized BP's mathematical characteristics to perform formal verification and verify the program behavior under all possible execution paths. The analysis proved that it plays as expected more than 99% of the time.

## 6 Using the framework in system design and maintenance

The taxonomy described above can assist developers in designing systems and planning upgrades to existing systems. There are many questions or properties associated with the system and the development project that affect such ES-and-ML integration decisions. For example:

(1) Is the system at hand fully developed, partially developed, or being assembled from significant reusable components?

(2) Availability of domain knowledge: access to human experts or existing expert systems, and documentation, that can provide much domain expertise. Can the existing knowledge be readily encapsulated and formulated in a programmable form? Is obtaining the necessary domain knowledge a major project in its own right?

(3) Is there sufficient data available for ML training? Is there an environment suitable for reinforcement learning?

(4) What human skills, computing resources, and calendar time are available for ML development and ES development, including iterative ML training and agile development?

(5) How critical are the different criteria for the project: precision/accuracy, performance, explainability, development costs, schedules, and maintainability?

Table 2 offers examples of applying such questions to the available domain knowledge and the system under consideration. Clearly, these factors and design choices are not mutually exclusive, developers have to weigh and compose these recommendations.

**Table 2** Use-cases for applying the framework

| Development use case | Applicable design choice |
|---|---|
| A domain-specific rule is to be added to an already developed ML system, like handling a new text change in a road sign in a driver assistance system, or a well-defined behavioral bug, or a succinct description of a use case that a competitor system failed to handle. | Runtime ES override rules. |
| A new ML-based system is being developed. However, a single domain-specific rule constitutes critical safety-related requirements; both the conditions and actions are readily programmable with ES; furthermore, stakeholders want to be able to readily demonstrate that the issue was handled during development. | Implement the rule ES specifications, either in training or at runtime of the ML system. |
| The inputs to succinct domain-specific rules are associated with rich inputs, like interpreting an image or a command in spoken or typed natural language. | Interpretation of triggers of action rules will likely be encapsulated in AI/ML modules. |
| Multiple domain-specific ES rules are available, but it is not clear how they should be prioritized or composed relative to each other and relative to other ML-based recommendations. | The ES rules will likely be incorporated in the training of the ML system. |
| A new easily programmed domain-specific rule applies to a narrow, easily trainable ML component of the application. | Use the rule only to automate the creation or selection of new training data for the ML component. |

## 7 Conclusion

Hybrid solutions, combining machine learning, and model-based executable specifications are a promising avenue to explore in the design of complex intelligent systems [4]. In this paper, we provided a framework for presenting and studying the various approaches to developing hybrid systems. In addition to guiding the structure and flow of this paper and assisting in decision-making at development time, our framework can apparently be used to study new emerging approaches not covered here and perhaps even help trigger new designs and integration techniques. Furthermore, We believe that with recent advancements in ML, there are even more opportunities for such hybrid ML-and-ES designs to offer significant advantages over end-to-end ML.

We surveyed and categorized approaches for such an integration, focusing on the integration of ML with scenario-based programming, also termed BP – a modeling paradigm designed to allow users to specify the system's behavior in a natural and intuitive manner, which is aligned with how humans often describe requirements. We elaborated on the advantages of BP as a specification approach in this context.

While the various approaches to ES and ML integration offer significant advantages, challenges and areas for future research remain.

First, we believe that the use of BP in hybrid systems may prove to be handy in providing situation-aware explanations. Dumas et al. [63] explained the challenges and opportunities in situation-aware explainability. They use 'explaining' as a general term, which could unfold into a variety of articulated queries, such as: "What are the reasons for performing action T? Why was decision X taken? When was it concluded that the sub-goal had been attained?" As an inherent step, they note that explainability should be realized so that its output is presented to the agent, in order for the agent to be able to understand and accordingly act upon it and the evolving situation inducing it. ES might provide the missing link for generating situation-aware explanations, since the specifications provide abstractions for the current state/context and can provide the causality and the consequences of a given fact. We believe that utilizing the specifications for providing situation-aware explanations is a promising direction. One possible way to do this is to map domain-independent explanations given by, e.g., SHAP, to the domain-specific specification.

Second, the hybridization of ES and ML may be streamlined with the future development of standard design patterns and respective interfaces for facilitating the various categories of integration.

Third, when such integration relies on an incremental addition of domain knowledge or intelligent capabilities, it is desired to have at our disposal appropriate metrics and methods for measuring the contribution of such incremental enhancement. Furthermore, developing the ability to estimate such contribution early on, as part of planning an enhancement, is a nontrivial challenge in its own right.

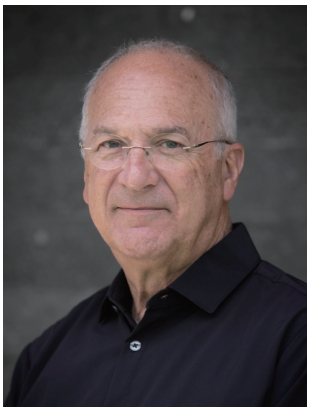
**Acknowledgements** This work was supported by National Natural Science Foundation of China (NSFC) and Israel Science Foundation (ISF) (Grant No. 3698/21). Additional support was provided by a research grant from the Estate of Harry Levine, the Estate of Avraham Rothstein, Brenda Gruss, and Daniel Hirsch, the One8 Foundation, Rina Mayer, Maurice Levy, and the Estate of Bernice Bernath. We thank the reviewers of earlier versions for their valuable comments and suggestions.

**References**


1 Zhu Z, Zhao H. A survey of deep RL and IL for autonomous driving policy learning. IEEE Trans Intell Transp Syst, 2022, 23: 14043–14065
2 Kuutti S, Bowden R, Jin Y, et al. A survey of deep learning applications to autonomous vehicle control. IEEE Trans Intell Transp Syst, 2020, 22: 712–733
3 Harel D, Marron A, Sifakis J. Autonomics: in search of a foundation for next-generation autonomous systems. Proc Natl Acad Sci USA, 2020, 117: 17491–17498
4 Sifakis J, Harel D. Trustworthy autonomous system development. ACM Trans Embed Comput Syst, 2023, 22: 1–24
5 Dulac-Arnold G, Levine N, Mankowitz D J, et al. An empirical investigation of the challenges of real-world reinforcement learning. 2020. ArXiv:2003.11881
6 Harel D, Marron A, Rosenfeld A, et al. Labor division with movable walls: composing executable specifications with machine learning and search (blue sky idea). In: Proceedings of the AAAI Conference on Artificial Intelligence, 2019. 9770–9774
7 Harel D. Statecharts: a visual formalism for complex systems. Sci Comput Program, 1987, 8: 231–274
8 Damm W, Harel D. LSCs: breathing life into message sequence charts. Formal Methods Syst Des, 2001, 19: 45–80
9 Harel D, Marelly R. Come, Let's Play: Scenario-based Programming Using LSCs and the Play-engine. Berlin: Springer, 2003
10 Harel D, Marron A, Weiss G. Behavioral programming. Commun ACM, 2012, 55: 90–100
11 Pnueli A. The temporal logic of programs. In: Proceedings of the 18th Annual Symposium on Foundations of Computer Science (SFCS 1977), 1977. 46–57
12 Julian K D, Lopez J, Brush J S, et al. Policy compression for aircraft collision avoidance systems. In: Proceedings of the 35th Digital Avionics Systems Conference (DASC), 2016. 1–10
13 Silver D, Huang A, Maddison C J, et al. Mastering the game of Go with deep neural networks and tree search. Nature, 2016, 529: 484–489
14 LeCun Y, Bengio Y, Hinton G. Deep learning. Nature, 2015, 521: 436–444
15 Martínez-Fernández S, Bogner J, Franch X, et al. Software engineering for AI-based systems: a survey. ACM Trans Softw Eng Methodol, 2022, 31: 1–59
16 Zhang T J, Xu H Z, Wang X L, et al. Noveld: a simple yet effective exploration criterion. In: Proceedings of Advances in Neural Information Processing Systems, 2021, 34: 25217–25230
17 Badia A P, Sprechmann P, Vitvitskyi A, et al. Never give up: learning directed exploration strategies. 2020. ArXiv:2002.06038
18 Le Mero L, Yi D, Dianati M, et al. A survey on imitation learning techniques for end-to-end autonomous vehicles. IEEE Trans Intell Transp Syst, 2022, 23: 14128–14147
19 Zheng B, Verma S, Zhou J, et al. Imitation learning: progress, taxonomies and challenges. IEEE Trans Neural Netw Learn Syst, 2022. doi: 10.1109/TNNLS.2022.3213246
20 Shin J, Nam J. A survey of automatic code generation from natural language. J Inform Process Syst, 2021, 17: 537–555
21 Dehaerne E, Dey B, Halder S, et al. Code generation using machine learning: a systematic review. IEEE Access, 2022, 10: 82434–82455
22 Wong M F, Guo S, Hang C N, et al. Natural language generation and understanding of big code for AI-assisted programming: a review. Entropy, 2023, 25: 888
23 Milan A, Rezatofighi S H, Dick A, et al. Online multi-target tracking using recurrent neural networks. In: Proceedings of the 31st AAAI Conference on Artificial Intelligence, 2017
24 Schwalbe G, Schels M. Concept enforcement and modularization as methods for the ISO 26262 safety argumentation of neural networks. In: Proceedings of the 10th European Congress on Embedded Real Time Software and Systems, 2020. 1–10
25 Hwang S, Lee K, Jeon H, et al. Autonomous vehicle cut-in algorithm for lane-merging scenarios via policy-based reinforcement learning nested within finite-state machine. IEEE Trans Intell Transp Syst, 2022, 23: 17594–17606
26 Ronan T, Anastasio S, Qi Z, et al. Openensembles: a python resource for ensemble clustering. J Mach Learn Res, 2018, 19: 956–961
27 Shalev-Shwartz S, Shammah S, Shashua A. On a formal model of safe and scalable self-driving cars. 2018. ArXiv:1708.06374
28 Chen S, Sun Y, Li D, et al. Runtime safety assurance for learning-enabled control of autonomous driving vehicles. In: Proceedings of International Conference on Robotics and Automation (ICRA), 2022. 8978–8984
29 Ghosh S, Lincoln P, Tiwari A, et al. Trusted machine learning: model repair and data repair for probabilistic models. In: Proceedings of AAAI Workshops, 2017
30 Damm W, Galbas R. Exploiting learning and scenario-based specification languages for the verification and validation of highly automated driving. In: Proceedings of the 1st International Workshop on Software Engineering for AI in Autonomous Systems (SEFAIAS), 2018. 39–46
31 Fremont D J, Kim E, Pant Y V, et al. Formal scenario-based testing of autonomous vehicles: from simulation to the real world. In: Proceedings of the 23rd International Conference on Intelligent Transportation Systems (ITSC), 2020. 1–8
32 Wang H. ReNN: rule-embedded neural networks. In: Proceedings of the 24th International Conference on Pattern Recognition (ICPR), 2018. 824–829
33 Ray P, Chakrabarti A. A mixed approach of deep learning method and rule-based method to improve aspect level sentiment analysis. Appl Comput Inform, 2022, 18: 163–178
34 Kang C, Kim G, Yoo S I. Detection and recognition of text embedded in online images via neural context models. In: Proceedings of the 31st AAAI Conference on Artificial Intelligence, 2017
35 Qiao Z, Zhou Y, Yang D, et al. Seed: semantics enhanced encoder-decoder framework for scene text recognition. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020. 13528–13537
36 Dreossi T, Ghosh S, Yue X, et al. Counterexample-guided data augmentation. 2018. ArXiv:1805.06962
37 Hu Z, Ma X, Liu Z, et al. Harnessing deep neural networks with logic rules. 2016. ArXiv:1603.06318
38 Hu Z, Ma X, Liu Z, et al. Harnessing deep neural networks with logic rules. In: Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, Berlin, 2016. 2410–2420
39 D'Angelo M, Gerasimou S, Ghahremani S, et al. On learning in collective self-adaptive systems: state of practice and a 3D framework. In: Proceedings of the 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), 2019. 13–24
40 Quin F, Weyns D, Bamelis T, et al. Efficient analysis of large adaptation spaces in self-adaptive systems using machine learning. In: Proceedings of the 14th International Symposium on Software Engineering for Adaptive and Self-Managing

Systems (SEAMS), 2019. 1–12
41 Braun M, Wrede S. Incorporation of expert knowledge for learning robotic assembly tasks. In: Proceedings of the 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), 2020. 1594–1601
42 Li G, Yang Y, Li S, et al. Decision making of autonomous vehicles in lane change scenarios: deep reinforcement learning approaches with risk awareness. Transp Res Part C-Emerg Technol, 2022, 134: 103452
43 Ye Y, Zhang X, Sun J. Automated vehicle's behavior decision making using deep reinforcement learning and high-fidelity simulation environment. Transp Res Part C-Emerg Technol, 2019, 107: 155–170
44 Wei L, Li Z, Gong J, et al. Autonomous driving strategies at intersections: scenarios, state-of-the-art, and future outlooks. In: Proceedings of IEEE International Intelligent Transportation Systems Conference (ITSC), 2021. 44–51
45 Kaiser L, Babaeizadeh M, Milos P, et al. Model-based reinforcement learning for Atari. 2019. ArXiv:1903.00374
46 Zhang P, Hao J, Wang W, et al. Kogun: accelerating deep reinforcement learning via integrating human suboptimal knowledge. In: Proceedings of the 29th International Conference on International Joint Conferences on Artificial Intelligence, 2021. 2291–2297
47 Cao Y, Li Z, Yang T, et al. Galois: boosting deep reinforcement learning via generalizable logic synthesis. 2022. ArXiv:2205.13728
48 Wolf P, Kurzer K, Wingert T, et al. Adaptive behavior generation for autonomous driving using deep reinforcement learning with compact semantic states. In: Proceedings of IEEE Intelligent Vehicles Symposium (IV), 2018. 993–1000
49 Brunke L, Greeff M, Hall A W, et al. Safe learning in robotics: from learning-based control to safe reinforcement learning. Annu Rev Control Robot Auton Syst, 2022, 5: 411–444
50 Jothimurugan K, Alur R, Bastani O. A composable specification language for reinforcement learning tasks. In: Proceedings of Advances in Neural Information Processing Systems, 2019
51 Elyasaf A, Sadon A, Weiss G, et al. Using behavioural programming with solver, context, and deep reinforcement learning for playing a simplified robocup-type game. In: Proceedings of the 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C), 2019. 243–251
52 Eitan N, Harel D. Adaptive behavioral programming. In: Proceedings of the 23rd International Conference on Tools with Artificial Intelligence, 2011. 685–692
53 Katz G. Augmenting deep neural networks with scenario-based guard rules. In: Proceedings of International Conference on Model-Driven Engineering and Software Development, 2020. 147–172
54 Yerushalmi R, Amir G, Elyasaf A, et al. Scenario-assisted deep reinforcement learning. In: Proceedings of the 10th International Conference on Model-Driven Engineering and Software Development (MODELSWARD), 2022. 310–319
55 Corsi D, Yerushalmi R, Amir G, et al. Constrained reinforcement learning for robotics via scenario-based programming. 2022. ArXiv:2206.09603
56 Ray A, Achiam J, Amodei D. Benchmarking Safe Exploration in Deep Reinforcement Learning. Technical Report. 2019. https://cdn.openai.com/safexp-short.pdf
57 Iba H, Hasegawa Y, Paul T K. Applied Genetic Programming and Machine Learning. Boca Raton: CRC Press, 2009
58 Vanchurin V, Wolf Y I, Katsnelson M I, et al. Toward a theory of evolution as multilevel learning. Proc Natl Acad Sci USA, 2022, 119: 2120037119
59 Cohen I R, Marron A. Evolution is driven by natural autoencoding: reframing species, interaction codes, cooperation and sexual reproduction. Proc R Soc B, 2023, 290:
60 Poliansky R, Sipper M, Elyasaf A. From requirements to source code: evolution of behavioral programs. Appl Sci, 2022, 12: 1587
61 Elyasaf A. Context-oriented behavioral programming. Inf Softw Tech, 2021, 133: 106504
62 Simani S, Fantuzzi C, Patton R J. Model-based fault diagnosis techniques. In: Proceedings of Model-based Fault Diagnosis in Dynamic Systems Using Identification Techniques, 2003
63 Dumas M, Fournier F, Limonad L, et al. AI-augmented business process management systems: a research manifesto. ACM Trans Manage Inf Syst, 2023, 14: 1–19

# Profile of David HAREL



Prof. David Harel has been President of the Israel Academy of Sciences and Humanities since September 2021, and has been on the faculty of the Weizmann Institute of Science since 1980, serving as Department Head from 1989 to 1995, and Dean of the Faculty of Mathematics and Computer Science between 1998 and 2004. He was a co-founder of I-Logix, Inc., which later became part of IBM. He received his Ph.D. from MIT in 1978 in record time of 20 months, and spent two years at IBM's Yorktown Heights Research Center, and sabbatical years at Carnegie-Mellon University, Cornell University, and the University of Edinburgh.

He has worked in several areas of theoretical computer science, producing fundamental results in computability theory, logics of programs, database theory, and automata theory. Over the years, he became involved in software and systems engineering, visual languages, layout of diagrams, modeling and analysis of biological systems, the synthesis and communication of smell, and most recently in the analysis of prosody. He is the inventor of the language of Statecharts and co-inventor of live sequence charts (LSCs) and scenario-based programming, and was part of the teams that designed the tools Statemate, Rhapsody, the Play-Engine and PlayGo, and the game Plethora.

Harel devotes part of his time to education and expository work, has delivered a lecture series on Israeli radio and has hosted a series of programs on Israeli television. He was also involved in planning a new computer science curriculum for high school. Some of his expository writings include the award winning book Algorithmics: The Spirit of Computing (1987, 1992, 2003, 2012), which was the Spring 1988 Main Selection of the Macmillan Library of Science, and Computers Ltd.: What They Really Can't Do (2000, 2012). The first of these was translated into Dutch, Hebrew, Polish, Chinese, German and Italian; and the second into German, Italian, Chinese and Hebrew.

He has received many awards, including the ACM Karlstrom Outstanding Educator Award (1992), the Israel Prize (2004), the ACM Software System Award (2007), the Emet Prize (2010), the IEEE Harlan Mills Award (2023), several best paper awards and six honorary degrees in France, Italy, the Netherlands and Israel. He is a Fellow of the ACM (1994), the IEEE (1995) the AAAS (2007) and the EATCS (2016), and a member of the Academia Europaea (2006) and the Israel Academy of Sciences (2010). He is an international member of the US National Academy of Engineering (2014), the American Academy of Arts and Sciences (2014), the US National Academy of Sciences (2019), and the Chinese Academy of Sciences (2021), and is a Fellow of the Royal Society (FRS) (2020).

Harel is also an ardent peace and human rights activist, and his main hobbies are photography and choir singing.

## Selected publications


• Harel D. Algorithmics: The Spirit of Computing. Reading: Addison-Wesley, 1987. 2nd ed., 1992; 3rd ed., 2004 (with Y. Feldman) (Dutch, 1989; Hebrew (Open University Press), 1991; Polish, 1992; Polish, 2001; Chinese 2006; German, 2006; Italian 2008)

• Harel D, Politi M. Modeling Reactive Systems with Statecharts: The STATEMATE Approach. New York: McGraw-Hill, 1998 (Early version, The Languages of STATEMATE. Technical Report, I-Logix, Inc., Andover, 1991)

• Harel D. Computers Ltd.: What They Really Can't Do. Oxford: Oxford University Press, 2000. Revised paperback edition, 2003 (German, 2002; Italian, 2002; Chinese, 2003; Hebrew, 2004)

• Harel D, Kozen D, Tiuryn J. Dynamic Logic. Cambridge: MIT Press, 2000

• Harel D, Marelly R Come, Let's Play: Scenario-Based Programming Using LSC's and the Play-Engine. Berlin: Springer-Verlag, 2003

• Harel D, Meyer A R, Pratt V R. Computability and completeness in logics of programs. In: Proceedings of the 9th Annual ACM Symposium on Theory of Computing, Boulder, 1977. 261–268

• Chandra A K, Harel D. Computable queries for relational data bases. J Comput Syst Sci, 1980, 21: 156–178 (Also, In: Proceedings of the 11th Annual ACM Symposium on Theory of Computing, Atlanta, 1979. 309–318)

• Harel D. On folk theorems. Comm Assoc Comput Mach, 1980, 23: 379–389

• Chandra A K, Harel D. Structure and complexity of relational queries. J Comput System Sci, 1982, 25: 99–128 (Also, In: Proceedings of the 21st Annual Symposium on Foundations of Computer Science, Syracuse, 1980)

• Harel D, Pnueli A, Stavi J. Propositional dynamic logic of nonregular programs. J Comput Syst Sci, 1983, 26: 222–243

• Harel D, Pnueli A. On the development of reactive systems. In: Logics and Models of Concurrent Systems. New York: Springer-Verlag, 1985. 477–498

• Harel D. Recurring dominoes: making the highly undecidable highly understandable. Ann Disc Math, 1985, 24: 51–72

• Harel D. Effective transformations on infinite trees, with applications to high undecidability, dominoes and fairness. J Assoc Comput Mach, 1986, 33: 224–248

• Harel D, Unger R, Sussman J. Beauty is in the genes of the beholder. Trends Biochem Sci, 1986, 11: 155–156 (cover feature) (Reprinted, Beauty is in the genes of the beholder: a golden tribute to a golden anniversary. In: DNA 50 — The Secret of Life: Celebrating the 50th Anniversary of the Double Helix Discovery. London: Faircount Press, 2003. 98–103)

• Harel D. Statecharts: a visual formalism for complex systems. Sci Comput Program, 1987, 8: 231–274 (Preliminary version, Technical Report CS84-05, The Weizmann Institute of Science, Rehovot, 1984)

• Harel D. On visual formalisms. Comm Assoc Comput Mach, 1988, 31: 514–530 (Reprinted, In: Diagrammatic Reasoning. Cambridge: AAAI Press and MIT Press, 1995. 235–271; In: High Integrity System Specification and Design. London: Springer-Verlag, 1999. 623–657)

• Harel D, Lachover H, Naamad A, et al. STATEMATE: a working environment for the development of complex reactive systems. IEEE Trans Softw Eng, 1990, 16: 403–414 (Early version, In: Proceedings of the 10th International Conference on Software Engineering, Singapore, 1988. 396–406; Reprinted, In: Software State-of-the-Art: Selected Papers. New York: Dorset House Publishing, 1990. 322–338)

• Harel D. Hamiltonian paths in infinite graphs. Israel J Math, 1991, 76: 317–336 (Also, In: Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, New Orleans, 1991. 220–229)

• Davidson R, Harel D. Drawing graphs nicely using simulated annealing. ACM Trans Graph, 1996, 15: 301–331

• Gal-Ezer J, Beeri C, Harel D, et al. A high-school program in computer science. Computer, 1995, 28: 73–80

• Harel D, Gery E. Executable object modeling with statecharts. Computer, 1997, 30: 31–42 (cover feature) (Also, In: Proceedings of the 18th International Conference on Software Engineering, Berlin, 1996. 246–257)

• Harel D, Naamad A. The STATEMATE semantics of statecharts. ACM Trans Softw Eng Method, 1996, 5: 293–333 (Preliminary version, Technical Report, I-Logix, Inc., 1989)

• Gal-Ezer J, Harel D. What (else) should CS educators know? Comm Assoc Comput Mach, 1998, 41: 77–84

• Damm W, Harel D. LSCs: breathing life into message sequence charts. Formal Method Syst Des, 2001, 19: 45–80 (Preliminary version, In: Proceedings of the 3rd IFIP International Conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS'99), 1999. 293–312)

• Gal-Ezer J, Harel D. Curriculum and course syllabi for a high-school program in computer science. Comput Sci Educ, 1999, 9: 114–147

• Harel D, Koren Y. A fast multi-scale method for drawing large graphs. J Graph Algorithm Appl, 2002, 6: 179–202

• Harel D, Marelly R. Specifying and executing behavioral requirements: the play in/play-out approach. Softw Syst Model, 2003, 2: 82–107

• Koren Y, Carmel L, Harel D. Drawing huge graphs by algebraic multigrid optimization. Multiscale Model Simul, 2003, 1: 645–673

• Harel D. A grand challenge for computing: full reactive modeling of a multi-cellular animal. Bull EATCS, 2003, 81: 226–235 (Early version, In: Proceedings of the UK Workshop on Grand Challenges in Computing Research, 2002; Reprinted, In: Current Trends in Theoretical Computer Science: The Challenge of the New Century, Algorithms and Complexity, Vol I. Singapore: World Scientific, 2004. 559–568)

• Efroni S, Harel D, Cohen I R. Towards rigorous comprehension of biological complexity: modeling, execution and visualization of thymic T-cell maturation. Genome Res, 2003, 13: 2485–2497 (cover feature)

• Harel D, Kugler H. The rhapsody semantics of statecharts (or, on the executable core of the UML). In: Integration of Software Specification Techniques for Applications in Engineering. Berlin: Springer-Verlag, 2004. 325–354

• Harel D, Rumpe B. Meaningful modeling: what's the semantics of 'semantics'? Computer, 2004, 37: 64–72 (cover feature)

• Fisher J, Piterman N, Hubbard E J A, et al. Computational insights into caenorhabditis elegans vulval development. Proc Natl Acad Sci, 2005, 102: 1951–1956

• Harel D. A turing-like test for biological modeling. Nat Biotechnol, 2005, 23: 495–496

• Haddad R, Carmel L, Sobel N, et al. Predicting the receptive range of olfactory receptors. PLoS Comput Biol, 2008, 4: e18

• Cohen I R, Harel D. Explaining a complex living system: dynamics, multi-scaling and emergence. J Royal Soc Interface, 2007, 4: 175–182

• Haddad R, Khan R, Takahashi Y K, et al. A metric for odorant comparison. Nat Method, 2008, 5: 425–429

• Fox Keller E, Harel D. Beyond the gene. PLoS ONE, 2007, 2: e1231

• Harel D. Can programming be liberated, period? Computer, 2008, 41: 28–37

• Setty Y, Cohen I R, Dor Y, et al. Four-dimensional realistic modeling of pancreatic organogenesis. Proc Natl Acad Sci, 2008, 105: 20374–20379

• Harel D. Statecharts in the making: a personal account. Comm Assoc Comput Mach, 2009, 52: 67–75

• Harel D, Marron A, Weiss G. Behavioral programming. Comm Assoc Comput Mach, 2012, 55: 90–100

• Haddad R, Medhanie A, Roth Y, et al. Predicting odor pleasantness with an electronic nose. PLoS Comput Biol, 2010, 6: e1000740

• Fisher J, Harel D, Henzinger T A. Biology as reactivity. Comm Assoc Comput Mach, 2011, 54: 72–82 (cover feature)

• Harel D, Maoz S, Szekely S, et al. PlayGo: towards a comprehensive tool for scenario based programming. In: Proceedings of the IEEE/ACM 25th International Conference on Automated Software Engineering (ASE2010), Antwerp, 2010. 359–360

• Harel D. Standing on the shoulders of a giant: one person's experience of Turing's impact. In: Proceedings of the 39th International Colloquium Conference on Automata, Languages, and Programming (ICALP), Warwick, 2012

• Harel D. Niépce-Bell or Turing: how to test odor reproduction? J Royal Soc Interface, 2016, 13: 125

• Harel D, Katz G, Marelly R, et al. Wise computing: towards endowing system development with proactive wisdom. Computer, 2018, 51: 14–26

• Sandak B, Cohen S, Gilboa A, et al. Computational elucidation of the effects induced by music making. PLoS ONE, 2019, 14: e0213247

• Sherman D, Harel D. Deciphering the underlying mechanisms of the pharyngeal motions in Caenorhabditis elegans. 2019. arXiv:1903.12009

• Harel D, Marron A, Sifakis J. Autonomics: in search of a foundation for next generation autonomous systems. Proc Natl Acad Sci, 2020, 117: 17491–17498

• Ravia A, Snitz K, Honigstein D, et al. A measure of smell enables creation of olfactory metamers. Nature, 2020, 588: 118–123

• Sifakis J, Harel D. Trustworthy autonomous system development. ACM Trans Embed Comput Syst, 2023, 22: 40

• Yerushalmi R, Amir G, Elyasaf A, et al. Enhancing deep reinforcement learning with scenario-based modeling. SN Comput Sci, 2023, 4: 156