Guess and then Check: Controller Synthesis for Safe and Secure Cyber-Physical Systems

Rong Gu$^{1[0000-0003-0570-6005]}$, Zahra Moezkarimi$^{1[0000-0001-5495-9098]}$, and Marjan Sirjani$^{1[0000-0001-5478-0987]}$

Mälardalen University, Västerås, Sweden
`(first.last)@mdu.se`

Abstract. In this paper, we report our ongoing work on safe and secure controller synthesis for cyber-physical systems (CPS). Our approach separates the synthesis process into three phases, in which we alternatively perform exhaustive and selective exploration of the system's state space. In this way, we combine the strengths of exhaustive search and learning to mitigate the state-space-explosion problem in controller synthesis while preserving the guarantee of safety and security. We implement the synthesis algorithms in the Rebeca (Reactive Objects Language) platform, which provides modelling, verification, and state-space visualization. We evaluate the new approach in an experiment, demonstrating the reduced number of explored states, which shows the potential of our approach for synthesizing safe and secure controllers for complex CPS.

1 Introduction

Correctness by construction was introduced by Church [7], who first brought up the famous synthesis problem. Since then, a great amount of effort has been made to address this problem [4, 11, 16]. Controller synthesis for *cyber-physical systems* (CPS) is different from that of pure software or hardware systems due to the close interaction of the cyber components and the physical components. Hence, the correctness of CPS depends on not only what and when actions are performed but also the reactions of the environment, which can be nondeterministic or stochastic. Besides, safety and security are also crucial for CPS. Safety means the system must not cause damage to itself and the environment, whereas security is concerned with external intrusion into the system [5]. In this paper, we aim to synthesize CPS controllers that are functionally correct, safe, and secure.

Exhaustive-search-based Synthesis. As synthesis is about finding a combination of the desired behaviour of CPS, a natural method is to exhaustively explore the state space of the system while collecting the desired execution traces, i.e., sequences of state-action pairs [1, 6]. The exhaustive-search-based synthesis has a correctness guarantee by the nature of exhaustive search. However, the state space for searching can easily grow to a scale that is unsolvable by the exhaustive-search-based methods [16]. Although many heuristics have been proposed to improve the performance of such methods in practical problems, bounded scalability is still the dominant factor limiting the application of the exhaustive-search-based synthesis.

Learning-based Synthesis. Learning bears the hope of overcoming the challenge of scalability in synthesis, as it has the potential to exploit the experience of other systems [11] or its own experience in the previous episodes of learning [10]. The key advantage of learning is that exhaustive exploration is not needed anymore. Instead, learning uses traces sampled from random simulations, and thus the state-space-explosion problem is alleviated. However, the sacrifice of using random simulation is the correctness guarantee. As a safety-critical system, an error in CPS may cause casualties, whereas accidents and attacks on CPS are becoming pervasive in our society, such as crashes involving Tesla's driver-assistance system [17], Jeep hacking on a highway [19], and a fatal crash caused by a self-driving car of Uber [3]. Therefore, learning needs complementary methods for safety and security guarantees.

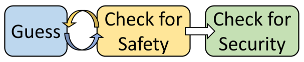


Fig. 1. Synthesis process

Our Aim. In this paper, we aim to combine the strengths of exhaustive exploration and learning in controller synthesis of CPS to tackle the state-space-explosion problem and preserve the guarantee of safety and security for the synthesized controllers. We model the CPS and its external environment as a Markov Decision Process where the actions of the CPS (resp., environment) are modelled as controllable (resp., uncontrollable) actions. Now controller synthesis is about finding the combination of controllable actions that satisfy the requirements regardless of how the uncontrollable actions take place. Our method is called *Guess and Check* as the synthesis starts with guessing a controller that *may* be correct and then thoroughly checks the *guessed* controller in the following phases (see Figure 1). Briefly, we alternatively adopt exhaustive search and learning (or random search) in different phases, which enable us to deal with large state spaces that are not solvable by pure exhaustive methods and still guarantee the safety and security of the synthesized controllers.

The remainder of the paper is organized as follows. Section 2 defines the problem and illustrates it in an example. Section 3 describes the algorithms, the platform, and a preliminary evaluation. In Section 4, we compare our method to other studies and envision future work.

2 Problem Description

2.1 Problem Definition

Functional correctness, safety, and security may refer to different meanings in different areas. To avoid confusion, we adapt the definitions of system models in the literature [14] and define CPS and its requirements as follows.

Definition 1 (CPS). *A CPS denoted by $\mathcal{C}$ is a quadruple $\mathcal{C}=(X, X_0, A, T)$, where X is a (possibly infinite) set of states, $X_0 \subseteq X$ is a (possibly infinite) set of initial states, A is a (possibly infinite) set of actions, and $T \subseteq X \times A \times X$ is a transition relation.*

If a CPS has finite sets of states and actions, it is a finite or symbolic system. A transition $t \in T$ is denoted as $t = (x, a, x')$, or $t(x, a) = x'$, where x' is the successor of state x when the system's action is a. If a CPS is deterministic, given any state $x \in X$ and any action $a \in A$, $t(x, a)$ returns either an empty set or a set with only one state; otherwise, the CPS is either non-deterministic (i.e., $t(x, a)$ can return multiple states) or stochastic (i.e., $t(x, a)$ returns a probabilistic distribution over X). Given a CPS and an action sequence $a_1a_2... \in A^*$, one can induce a state sequence $x_1x_2... \in X^*$, where $x_n = t(x_{n-1}, a_{n-1})$. We also call a state sequence a trace and denote it by π. One can obtain a finite trace by cutting a trace at any of its states. A finite part of a trace is denoted by π_f. A CPS's states can be partially observable, so we call $O(\pi)$ the observable part of trace π and $H(\pi)$ the hidden part.

Definition 2 (Controller). *Given a CPS $\mathcal{C}$, a controller of $\mathcal{C}$ is a partial function $\sigma : \pi_f \to A$. If $\mathcal{C}$ is Markovian, the controller can be memoryless, that is, $\sigma : last(\pi_f) \to A$, where $last(\pi_f)$ is the last state of trace π_f.*

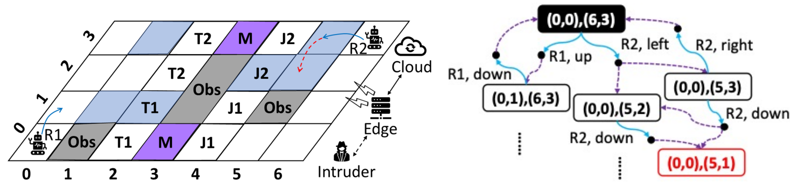
Given a CPS $\mathcal{C}$ controlled by a controller σ (denoted by $\mathcal{C}|\sigma$), one can induce a set of traces Π_σ s.t. $\forall \pi \in \Pi_\sigma$, $\exists a \in \sigma(\pi_f)$. Let $\mathcal{G} \subseteq X$ be the goal states that CPS aims to reach and $\mathcal{U} \subseteq X$ be the unsafe states that CPS must avoid, this paper is about synthesizing $\mathcal{C}|\sigma$ for a nondeterministic and Markovian CPS such that its Π_σ satisfies the following three properties.

- **Functional correctness**: $\forall \pi \in \Pi_\sigma$, $\exists x \in \pi$ s.t. $x \in \mathcal{G}$.
- **Safety**: $\forall \pi \in \Pi_\sigma$, $\forall x \in \pi$ s.t. $x \cap \mathcal{U} = \emptyset$.
- **Security**: $\forall \pi \in \Pi_\sigma$, $\exists \pi' \in \Pi_\sigma$ s.t. $H(\pi) \neq H(\pi')$ and $O(\pi) = O(\pi')$.

Assuming a CPS is non-deterministic due to the uncertain reaction from the environment, intuitively, functional correctness means the CPS must always eventually reach the goal state regardless of the environment's reaction. Similarly, safety means the system must always avoid unsafe states. We can express the safety and functional correctness as invariance and reachability properties of temporal logic [2], respectively. Assuming confidential information is contained in the hidden part of a CPS's states, security here means this information must not be revealed to unauthorized ones, like intruders. Therefore, the definition of security above means that intruders must not be able to deduce confidential information from the observable states. Formally, security is also formulated as hyperproperties [8] as it involves multiple traces. We elaborate CPS and the three types of requirements in the following example.

2.2 Illustrative Example

In this section, we illustrate the problem in an example abstracted from an industrial use case, where robots are employed in a factory for goods delivery (Figure 2). For simplicity, we discretize the environment as a 7×4 grid, in which robot R1 (resp., R2) must finish tasks T1 and T2 (resp., J1 and J2) at the right cells, and then they meet at an M cell as the destination of the mission. The robots can go through the cells without performing the tasks. The grey (resp., blue) cells are obstacles (resp., wet floors), where the robots must not enter (resp., may slip). When robots slip on a wet floor, they may end in the wrong position, i.e., one cell below the target. The robots' trajectories are planned at the edge-computing server (ECS) whereas the high-level tasks are scheduled at the cloud-computing server (CCS). An intruder is trying to attack the ECS and change the task order but he cannot access the CCS.



(a) The grid of the environment (b) MDP of the CPS

Fig. 2. An example of CPS: two robots collaborating in a confined environment.

When modelling this example, we employ a *Markov Decision Process* (MDP) for a *2-player* game [12], where actions of the CPS and the environment are assigned to different transitions of the model (see Figure 2(b)), that is, controllable ones (blue arrows) and uncontrollable ones (dotted violet arrows). At each state, a robot gets to choose a controllable action, e.g., moving up/down, after which the environment's actions take place, which decides the robot's ending positions. A correct and safe controller must guarantee the robot reaches the specific cells for task execution and never enters the unsafe cells despite the environment's actions. For security, we mean confidential information, e.g., task order, must not be revealed to the intruder. Although the intruder does not have access to the CCS, he can deduce the task order by using the robots' trajectories. For instance, $\texttt{trace}_1$ is insecure because the only chance for R1 to execute T1 is at cell `(2,1)`, and it is before the T2 cells being visited (e.g., `(2,2)`). In contrast, $\texttt{trace}_2$ is secure as R1 visits the T1 and T2 cells alternatively more than once, so the intruder cannot deduce R1's task execution order from its moving trajectory.

```
trace₁: (0,0)->(0,1)->(1,1)->(2,1)->(2,2)->(2,3)->(3,3)
trace₂: (0,0)->(0,1)->(1,1)->(2,1)->(2,2)->(2,1)->(2,2)->(2,3)->(3,3)
```

Functional correctness means the robots must finish all tasks within a time frame. To synthesize a functional correct, safe and secure controller is not trivial, especially when considering the uncertain actions of the environment. Next, we introduce our solution.

3 Solution and Preliminary Evaluation

In this section, we introduce the algorithms of our method *Guess and Check* and explain the thoughts behind the algorithms.

Phase 1: guess for a controller. Our goal of the first phase is to guess a controller that *may* satisfy the requirements. Generally, we explore the state space of the model by following two rules: i) exhaustive exploration of controllable actions, and ii) selective exploration of uncontrollable actions. Algorithm 1 presents how guessing is conducted. For simplicity, we assume that the state-space exploration is depth-first. However, using other orders of exploration does not affect the correctness of the algorithm. As a recursive function, the termination conditions of Algorithm 1 are defined in lines 3 - 8. Specifically, when the state-space exploration reaches a state where the reachability property is satisfied (line 3), the trace is added to the *controller*, which is a global variable used in all algorithms. Alternatively, when the exploration reaches an *unsafe* state, or a loop, or a deadlock where no action is available (line 6), the trace is pruned from the controller. In both cases, the trace is fed into a learning algorithm (line 4 and line 7), e.g., Q-learning [18], for calculating a policy that enables the environment to *win* the game faster (i.e., the *BEST* function in line 10), that is, leading the exploration to an unsafe state. Lines 9 - 14 implement the exploration rules i) and ii). After phase 1, we find an *optimistic* controller that may be correct as we only explore the environment's actions partially.

Algorithm 1: Guess for an optimistic controller

```
1  Set<Pair<State, Action>> $\sigma$                                   // controller
2  Function GUESS(State x, Trace $\pi$, Set<State> $\mathcal{G}$, Set<State> $\mathcal{U}$)
3      if $x \in \mathcal{G}$ then
4          LEARN&ADD($\sigma$, $\pi$)
5          return
6      if $x \in \mathcal{U} \vee (\exists pair \in \pi \wedge x \in pair) \vee x.actions = \emptyset$ then
7          LEARN&PRUNE($\sigma$, $\pi$)
8          return
9      if $\exists a \in x.actions \wedge a.type = ENVIRONMENT$ then
10         Action best := BEST(x.actions)
11         GUESS(best.target, $\pi$.push(x, best), $\mathcal{G}$, $\mathcal{U}$)
12     if $\forall a \in x.actions \wedge a.type = SYSTEM$ then
13         while (Action next := NEXT(x.actions) $\neq$ LAST(x.actions)) do
14             GUESS(next.target, $\pi$.push(x, next), $\mathcal{G}$, $\mathcal{U}$)
15     return
```

Algorithm 2: Check for the safety requirement

```
1  Function C4SA(State x, Trace $\pi$, Set<State> $\mathcal{G}$, Set<State> $\mathcal{U}$)
2      Boolean pass := false
3      if $x \in \mathcal{G}$ then
4          return (true)
5      if $x \in \mathcal{U} \vee (\exists pair \in \pi \wedge x \in pair) \vee x.actions = \emptyset$ then
6          LEARN&PRUNE($\sigma$, $\pi$)
7          return (false)
8      if $\exists a \in x.actions \wedge a.type = ENVIRONMENT$ then
9          pass := true
10         for $a \in x.actions \wedge a.type = ENVIRONMENT \wedge pass$ do
11             pass := pass $\wedge$ C4SA(a.target, $\pi$.push(x, a), $\mathcal{G}$, $\mathcal{U}$)
12     if $\forall a \in x.actions \wedge a.type = SYSTEM$ then
13         if $x \notin \sigma.getAllStates()$ then
14             for $a \in x.actions$ do
15                 GUESS(a.target, $\pi$.push(x, a), $\mathcal{G}$, $\mathcal{U}$)
16         for $a \in x.actions \wedge Pair(x, a) \in \sigma$ do
17             pass := pass $\vee$ C4SA(a.target, $\pi$.push(x, a), $\mathcal{G}$, $\mathcal{U}$)
18     return (pass)
```

Phase 2: check for safety. In Algorithm 2, we start to check the optimistic controller for the safety property. Generally, we replace the rules of state-space exploration with two new rules: i) selectively exploring controllable actions that are contained in the optimistic controller, and ii) exhaustive exploration of uncontrollable actions. Similarly to Algorithm 1, lines 3 - 7 in Algorithm 2 show the termination conditions of the recursive algorithm, that is, when the check passes or fails. Lines 8 - 11 depict the exhaustive exploration of the environment's actions until one trace fails to pass the check. If we see a state that only has the system's actions to choose but is not contained in the controller (line 13), we go back to guess a new controller starting from that state (line 15). We use $\vee$ in line 17 because one of the system's actions passing the check is enough for the state to be included in the controller. However, we use $\wedge$ in line 11 because all the environment's actions need to pass the check. After phase 2, we have obtained a safe and functionally correct controller. Next, we check for security.

Phase 3: check for security. By following the generated controller, we explore the state space again to check for the security properties (Algorithm 3). We formulate the confidential and public information as properties P_s and P_c, respectively. As lines 3 - 4 show, if a trace π_1 satisfies P_s, there must be a trace π_2 that satisfies P_c, and π_1 and π_2 are similar enough to prevent the intruders from distinguishing them (i.e., $D(\pi_1, \pi_2) \leq \tau$), where D is a function for computing the distance of two traces. The function can be replaced by an equation $O(\pi_1) = O(\pi_2)$ for discrete state spaces, where O returns the observable part of a trace. The definition of the distance function is not the focus of this paper. Interested readers are referred to the literature [14].

Algorithm 3: Check for the security requirement

```
Function C4CE(State x, Property $P_s$, Property $P_c$)
    Set<Trace> $\Pi_\sigma$ := EXPLORE(x, $\sigma$)
    for $\pi_1 \in \Pi \wedge \pi_1 \models P_s$ do
        if $\neg((\exists \pi_2 \in \Pi_\sigma \wedge \pi_2 \models P_c) \wedge D(\pi_1,\pi_2) \leq \tau)$ then
            LEARN&PRUNE($\sigma$, $\pi_1$)
    return
```

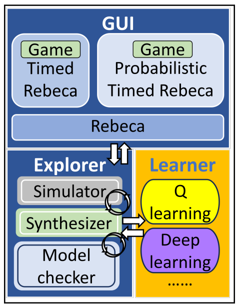


Fig. 3. Platform

Platform and evaluation. We aim to realize the algorithms in a platform including a GUI for modelling CPS in various branches of Rebeca, a back-end state-space explorer, and external libraries for learning. Figure 3 shows the architecture of our platform, where the GUI supports modelling the CPS in two languages, which are all based on Rebeca and extended to support games [12], that is, the modelling of controllable actions of CPS and uncontrollable actions of the environments. The back-end explorer includes a simulator that can randomly explore the state space via Monte-Carlo Simulation, a set of model checkers that are suitable for different kinds of models, and a synthesizer that calls the simulator and model checkers for controller synthesis. The learning module is an external library such that the platform is extendable and adaptive.

Now, we present our experiment for a preliminary evaluation of the platform[1]. We show how the *Guess and Check* method reduces the number of states explored in phase 1 and phase 2[2]. We build several models based on the example in Fig. 2. In these models, we set the maximum number of steps that robots can move and adjust their goals accordingly. Then we generate the state space of the models and synthesize a controller for each of the models using our algorithms. Note that we use random simulation instead of learning in phase 1, and we repeat the experiment 10 times and use a box plot to show the numbers of explored states in the experiments (Fig. 4). The experimental results show that in both phases 1 and 2, the number of explored states is much less than the total number of states. The encouraging results demonstrate that our approach has the potential to solve problems that are too complex to be solved by the exhaustive-search-based methods and preserve the guarantee of safety and security that is impossible for pure reinforcement learning.

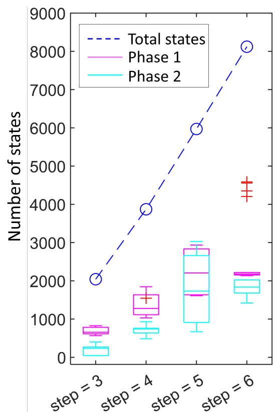


Fig. 4. Result

4 Discussion and Future Work

Related work. Due to the page limit, we only compare our method with the latest studies in this section. The most recent work in learning-based synthesis [11] introduces a method for winning strategy synthesis in parity games derived from LTL synthesis. Parker et al. [15, 13] propose binary decision diagrams and synthesis algorithms based on probabilistic model checking. Our primary distinction from prior methods is that we integrate exhaustive search and learning, leveraging the unique advantages of each in a synergistic manner.

Future work. We will finish implementing our platform in an open-source toolset of *Rebeca* [9], in which we integrate the features of Rebeca, Timed Rebeca, and Probabilistic Timed Rebeca to facilitate the synthesis algorithms. As Rebeca has been applied in many CPS applications, we will experiment with our new algorithms on these real-world problems. We implement the learning module as an external library, which enables us to explore various learning models, such as neural networks (NN), in our synthesis algorithms. In this line of research, we will try to see how model checking and machine learning can benefit each other in controller synthesis. As safety-critical CPS are often working alongside humans, we would like to consider the human factors in the controller synthesis of CPS, such as investigating best-effort strategies when the environmental constraints are too restrictive for the CPS to achieve all of their goals. We aim to make our approach adaptive for multiple objectives aligned with human preferences.

[1] Code of the experiment is published: https://github.com/rgu01/RebecaLearning

[2] The number of states in phase 3 depends on the controller size, not on the approach.

References

1. Asarin, E., Maler, O., Pnueli, A., Sifakis, J.: Controller synthesis for timed automata. IFAC Proceedings Volumes **31**(18), 447–452 (1998)
2. Baier, C., Katoen, J.P.: Principles of model checking. MIT press (2008)
3. BBC: Uber's self-driving operator charged over fatal crash. https://www.bbc.com/news/technology-54175359 (September 16th, 2020)
4. Bloem, R., Jobstmann, B., Piterman, N., Pnueli, A., Saár, Y.: Synthesis of reactive (1) designs. Journal of Computer and System Sciences **78**(3), 911–938 (2012)
5. Burns, A., McDermid, J., Dobson, J.: On the meaning of safety and security. The computer journal **35**(1), 3–15 (1992)
6. Cassez, F., David, A., Fleury, E., Larsen, K.G., Lime, D.: Efficient on-the-fly algorithms for the analysis of timed games. In: CONCUR 2005. Springer (2005)
7. Church, A.: Application of recursive arithmetic to the problem of circuit synthesis. Journal of Symbolic Logic **28**(4) (1963)
8. Clarkson, M.R., Schneider, F.B.: Hyperproperties. Journal of Computer Security **18**(6), 1157–1210 (2010)
9. Group, R.R.: Rebeca. https://rebeca-lang.org/ (2017)
10. Gu, R., Jensen, P.G., Seceleanu, C., Enoiu, E., Lundqvist, K.: Correctness-guaranteed strategy synthesis and compression for multi-agent autonomous systems. Science of Computer Programming **224**, 102894 (2022)
11. Křetínskỳ, J., Meggendorfer, T., Prokop, M., Rieder, S.: Guessing winning policies in ltl synthesis by semantic learning. In: International Conference on Computer Aided Verification. pp. 390–414. Springer (2023)
12. Kumar, P.R., Shiau, T.H.: Existence of value and randomized strategies in zero-sum discrete-time stochastic dynamic games. SIAM Journal on Control and Optimization **19**(5), 617–634 (1981)
13. Kwiatkowska, M., Norman, G., Parker, D., Santos, G.: Symbolic verification and strategy synthesis for turn-based stochastic games. In: Raskin, J., Chatterjee, K., Doyen, L., Majumdar, R. (eds.) Principles of Systems Design - Essays Dedicated to Thomas A. Henzinger on the Occasion of His 60th Birthday. Lecture Notes in Computer Science, vol. 13660, pp. 388–406. Springer (2022)
14. Liu, S., Trivedi, A., Yin, X., Zamani, M.: Secure-by-construction synthesis of cyber-physical systems. Annual Reviews in Control **53**, 30–50 (2022)
15. Parker, D.: Multi-agent verification and control with probabilistic model checking. In: International Conference on Quantitative Evaluation of Systems. pp. 1–9. Springer (2023)
16. Pnueli, A., Rosner, R.: On the synthesis of an asynchronous reactive module. In: International Colloquium on Automata, Languages, and Programming. pp. 652–671. Springer (1989)
17. Post, T.W.: 17 fatalities, 736 crashes: The shocking toll of tesla's autopilot. https://www.washingtonpost.com/technology/2023/06/10/tesla-autopilot-crashes-elon-musk/ (June 10th, 2023)
18. Sutton, R.S., Barto, A.G.: Reinforcement learning: An introduction. MIT press (2018)
19. Wired: Hackers remotely kill a jeep on the highway—with me in it. https://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/ (2015)