CTL Model Checking Based on Binary Classification of Machine Learning

Weijun Zhu
School of Computer and Artificial Intelligence
Zhengzhou University, China
zhuweijun@zzu.edu.cn

Huanmei Wu
College of Public Health,
Temple University, USA
huanmei.wu@temple.edu

Abstract: *In this study, we establish and pioneer an approximate Computational Tree Logic (CTL) Model Checking (MC) technique, in order to avoid the famous State Explosion (SE) problem in the Computational Tree Logic Model Checking (CTLMC). To this end, some Machine Learning (ML) algorithms are introduced and employed. On this basis, CTL model checking is induced to binary classification of machine learning, by mapping all the two different results of CTL model checking into all the two different results of binary classification of machine learning, respectively. The experimental results indicate that the newly proposed approach has a maximal accuracy of 100% on our randomly generated data set, compared with the latest algorithm in the classical CTL model checking. Furthermore, the average speed of the new approach is at most 120 thousand times higher than that of the latest algorithm, which appears in the current version of a popular model checker called NuXMV, in the classical CTL model checking. These observations prompt that the new method can get CTL model checking results quickly and accurately, since the SE problem is avoided completely.*

Keywords: *Model checking, computational tree logic, machine learning, binary classification.*

Received October 10, 2020; accepted October 14, 2021
https://doi.org/10.34028/iajit/19/2/12

1. Introduction

Temporal logic model checking (model checking for short) was developed Clarke and Emerson [16] in the 1980s, who were recognized by the 2007 Turing Award from the Association for Computing Machinery (ACM). This formal technique has been used widely for more than three decades.

There are some notable real-world examples about Model Checking (MC) applications, such as Stanford University's MC verification of a network protocol [31], Microsoft's MC-based software verifier for analyzing the source code of Windows device drivers [5, 27], the Turing laureate Prof. Clarke's assertion about model checking for detecting the Pentium floating-point dividing bug, i.e., the most famous of the Intel microprocessor bugs [14, 17, 35], National Aeronautics and Space Administration (NASA)'s applications about model checking for analyzing avionics software [20]. Even outside of computer science, model checking also play a vital role in analyzing an active structural control system to make buildings more resistant to earthquakes [15].

In general, the procedure of model checking (refers to qualitative model checking in this paper) is as follows. A temporal logic formula specifies a property which should be satisfied by a computational system, as well as an automaton or Kripke structure models this system. The MC result will be "yes", if a model checking algorithm automatically finds that the Kripke structure satisfies the formula. Otherwise, the MC result will be "no".

There are many temporal logics. The two most frequently used in the MC practices of the information technology industry are Linear Temporal Logic (LTL) [33] and Computational Tree Logic (CTL) [8, 22]. LTL was introduced to computer science by Pnueli [33] who is also a Turing Award winner. CTL was presented by Prof. Clarke, another Turing Award winner [16]. The different temporal logics express the different properties and, thus, have varied definitions, MC algorithms, MC tools, and applications. For example, SPIN is an LTL model checker, while its-ctl is a CTL model checker. Furthermore, NuSMX and NuXMV are model checkers for both LTL formulas and CTL formulas.

The SE problem has become a major bottleneck, both in LTL model checking and CTL model checking, so that Clarke listed the State Explosion (SE) problem as the first main disadvantage of the concurrent model checking [15]. Much progress has been made on this problem [15]. The many existing approaches (the state reduction technique, see [2, 4, 18, 19, 21, 25, 28, 34, 38] for details on many works combating the famous SE problem) can reduce the huge state space effectively. However, the SE problem remains. "unavoidable in worst case, but steady progress over the past 28 years using clever algorithms, data structures, and engineering", Clarke wrote, in one of his PowerPoint documents entitled "Model Checking: My 30 Year Quest to Conquer the State Explosion Problem" [14].

Since the SE problem cannot be avoided in the

framework of accurate computing, how about approximate computing?

Aiming to combat SE in CTL model checking, we try to propose an approximate CTL model checking approach in this study. To this end, we employ the more machine learning algorithms (the sixteen Machine Learning (ML) algorithms) to predict CTL model checking results, respectively. Our experimental results show that each of these machine learning algorithms can conduct predictions with a high optimal accuracy (from 90% to 100%), supposing an accurate/classical CTL model checking approach has an accuracy of 1. With the newly proposed method, the efficiency is improved by thousands of times, even hundreds of thousands of times, compared with state of the art of the algorithm in the classical CTL model checking, which is employed by the nuXmv model checker [9]. As shown in section 4, our experiments will demonstrate this point.

These observations indicate that it is possible to avoid the SE in CTL model checking using approximate computing, although the cost is also possible. As a result, the approximate CTL model checking technique is formed. This is the contribution of this paper.

It should be noted that, we do not regard the machine learning as a panacea in advance. And we just wonder whether this popular technique can address our problem or not, when this problem has not been solved by the existing approaches. Thus, we explore it with an objective experimental way. In order to improve the readability, we use some intuition meaning and figures instead of formal definitions to express model checking, machine learning and our idea in this paper.

The remainder of this paper is organized as follows. Section 2 briefs some preliminaries. The newly proposed approach will be illustrated in section 3. In section 4, we will conduct the comprehensive experiments. And the newly proposed approach and some related ones will be compared in section 5. In the last section, we will draw our conclusion.

2. Background

2.1. NuXMV

As a symbolic CTL model checker, NuXMV was designed by Carnegie Mellon University [9]. And it employs a state of the art of CTLMC algorithms [9], as claimed by [9].

NuXMV can be employed to model checking finite-state systematic models and infinite-state systematic models. And both CTL and LTL are supported by this tool.

In order to perform CTL model checking, a user's standard process can be depicted as follows:

1. The user should write a document to describe his/her systematic model, i.e., an automaton or a Kripke structure, in NuXMV language. Yes, NuXMV is not only a model checking tool, but also a program language.
2. The user's CTL formulas should be written in the same document in the same language. And the location is behind the description of systematic model.
3. The above document should be closed, and a given NuXMV command will be manually executed in command-line environment by the user, to get the returned model checking results.

See [9] for more details on NuXMV.

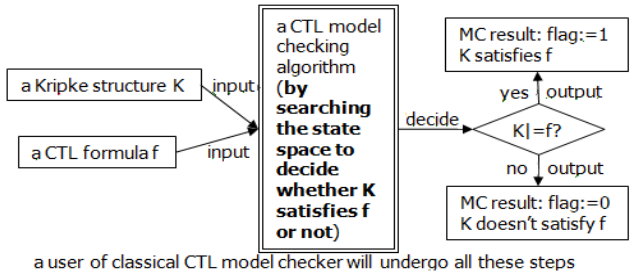
2.2. ML Algorithms and the Tools

Machine learning has been applied to many fields, such as natural language process [1], image process. Data classification is one of the major missions of machine learning. And a lot of ML algorithms can do it. In this paper, we employ the following algorithms: Boosted Tree (BT), Random Forest (RF), Decision tree (DT), Logistic Regression (LR), Extra Trees (ET), K-Nearest Neighbors (KNN), Nearest Centroid (NC), Ridge (RIDGE), Passive Aggressive (PA), Stochastic Gradient Descent (SGD), Linear Support Vector Classification (L_SVC), Nu Support Vector Classification (N_SVC), C-Support Vector Classification (C_SVC), Gaussian Process (GPC), Naive Bayes classifier for multivariate Bernoulli models (NB_BER) and Gaussian Naive Bayes (NB_GAU). Furthermore, Turi Create [3] and Scikit-learn [24] integrate some popular ML algorithms mentioned above, as well as implement them. See [3, 24] for more details on these algorithms and tools.

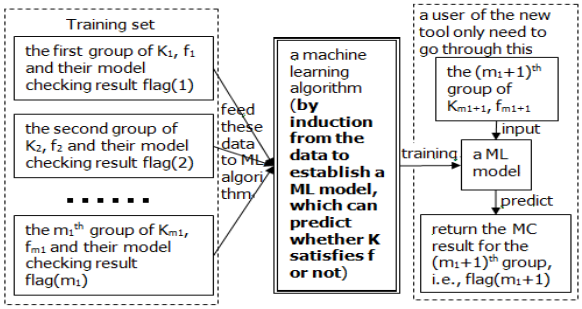
3. The Key Principle of the Newly Proposed Approach

The approximate Computational Tree Logic Model Checking (CTLMC) can be defined as follows: giving a pair of systematic model K and a CTL formula f, how to decide whether K satisfies f, with a machine learning algorithm.

The key principle of the newly proposed approach is illustrated in Figure 1. And this figure shows some information about the core steps of the new method, as follows. At first, similar to the approximate Linear Temporal Logic Model Checking (LTLMC) [41], some records about Kripke structures, temporal logic formulas and their model checking results will be inputted to take part in the process of training. And then, the obtained ML model can predict the CTLMC result for another pair of formula and Kripke structure. Differing from the approximate LTLMC [41], this time, CTL formula f and Kripke structures K make up the two ML features and the only one label is CTL model checking result r, as well as CTL model checking results instead of LTL ones can be predicted.



a) For a given pair of Kripke structure K and a CTL formula f, one can determine whether K satisfies f or not, using the classical CTL model checking.



b) A ML model M which can predict the model checking result for another pair of K_{m1+1} and f_{m1+1}, since M is obtained by training m_1 groups of K, f and their model checking results.

Figure 1. Given one pair of systematic model and formula, the new method predicts whether this model satisfies this formula or not.

How to vectorize the ML features consisting of automata and CTL formulas before a ML training is started? An automaton can be transformed into a digital string, and this string can be input into the training set and the testing set, as shown in example 1 in section 4.3. As for a CTL formula, its original form can be input directly into the training set and the testing set, and many ML tools provide a function which can regularize the values of the ML features, guaranteeing the quality of the following ML training. As a result, a ML algorithm can directly access this training set to start the training process.

4. Experiments

4.1. Experimental Target

The efficiency and the power of the newly proposed approach will be explored. And the approximate CTL model checking (based on machine learning/learning from data) and the classical CTL model checking (based on accurate computing/state exploration) will be compared in terms of the efficiency and the power, as well as the approximate CTL model checking based on the different machine learning algorithms will be also compared in terms of the efficiency and the power.

4.2. Platform

1. CPU: Intel(R) Core (TM) i7-4770 @3.40GHz.
2. RAM: 16.0 G.
3. Operating System (OS): Windows 7 64 bit.
4. NuXMV: CTL model checker.
5. Graphlab: for implementing the following four machine learning algorithms: RF, BT, DT and LR.
6. Scikit-learn: for implementing the following sixteen ML algorithms: RF, BT, ET, DT, KNN, NC, LR, RIDGE, PA, SGD, L_SVC, N_SVC, C_SVC, GPC, NB_BER and NB_GAU.

4.3. Experimental Procedures

This time, we produced randomly 50 Kripke structures K and 200 CTL formulas f, where the length of each of formula (L) is 500 (each formula has 500 symbols).

To this end, we conducted ten thousand (200*50=10000) groups of sub-experiments with NuXMV, in order to find out whether these fifty Kripke structures satisfy two hundred CTL formulas, respectively. The obtained data set A consists of these 10000 records, and each record contains the following three fields: K, f and r. The experimental steps in this study is similar with the ones in [41]. No more detail is given here due to the simplicity.

It should be noted that there are three data sets related with A, i.e., A1, A2 and A3. These data sets are formed in the following way:

1. A1 is a sub set of A, and it contains seven thousand records (L=500) which are selected randomly from A.
2. A2 is a sub set of A, and it contains one thousand records (L=500) which are selected randomly from A.
3. The intersection of A1 and A2 is empty.
4. A3 is an external data set of A, i.e., the intersection of A3 and A is empty, and it contains one thousand records (L=500). These different data sets will play different roles for the different objectives in the following experiments. Please see the section 4.7 for more details.

We provide here an example to illustrate how to obtain a record in the data set A, according to a given CTL formula and a given Kripke structure.

Example 1, Let us suppose the Kripke structure K is illustrated in Figure 2, and we can encode this Kripke structure as K= "000010010010111011012212430324 3". See [41] for more details on the way of encoding.

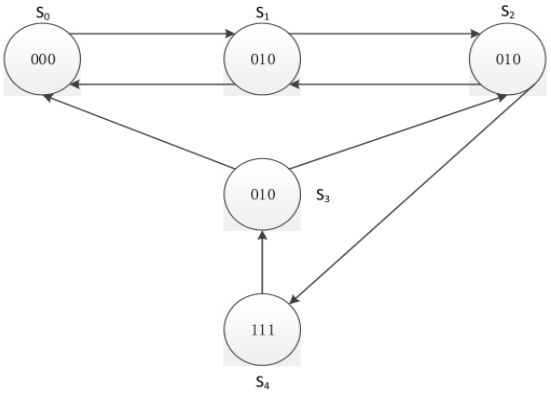


Figure 2. a Kripke structure K.

Supposing the CTL formula ff_1 is shown in Figure 3-a), we can use NuXMV to conduct CTL model checking. The result is illustrated by Figure 4-a), indicating K satisfies ff1. Thus, a record is added to A, and this record contains the three fields: "000010010010111011010122124303243", "A[(!(p=0&q=0|r=0))U……" and "true", where the first two fields are the features and the last one is the label when ML training comes.

A[(!(p=0&q=0|r=0))U(A[(!EX((AX(A[(E[(EG(p=0&q=1|r=1))U(AF(p=1|q=0|r=1)))]U(AF(p=1&q=1&r=0))))]&(EG(E[(A[(EG(p=1&q=1|r=0))U(EF!(p=1|q=1&r=1)))]U((AF!(p=0&q=1&r=1)))))))))U(A[(A[(E[(A[(AG(p=1|q=0&r=0))U(AX(p=0|q=1&r=0)))]U(EX(p=1|q=0&r=1)))]U(p=0|q=1&r=0)))]U(E[(E[(AX((EG(p=0|q=1&r=0))&(!E[(A[EG(p=0|q=1&r=0)U(!AF(p=1&q=1|r=1)))]U(EX(p=0|q=1&r=1)))))]U(!E!EX(A[(A[(A[(AX((!AG(p=0|q=1&r=0))&(E[(A[(E[AF(p=0&q=1|r=0)U(p=0|q=1|r=0))]U(AX(p=0&q=0|r=0)))]U(!EG(p=0&q=1|r=0)))))]U(E[(A[(EG(A[(E[(AF(p=0&q=1|r=1))U(EG!(p=0&r=1)))]U(E[(AF!(p=1|r=1))U(AF!(p=0&q=1)))])))]U(AG(q=0&r=1)))]U(E[(A[(E[(p=1&q=1&r=0)U(AF(p=0|q=1|r=0)))]U(!EX(p=0&q=1&r=0))]U(AG((EF(p=0|q=1&r=0))&(AX(q=0|r=1))))))))]U(AF((AG(p=1&q=1|r=0))|(E[(A[(EG(p=0|q=1|r=1))U(EG(p=1&q=0&r=1)))]U(E[(AF(p=0|q=1&r=1))U(AX(p=1|q=0|r=1))))))))]U(A[(AF(E[(A[(AG!(p=0|q=1|r=0))U(AF!(p=1&q=0)))]U(EX!(p=0&q=1|r=1))))]U(AF(p=1&q=0|r=1)))))))]U(A[(A[(p=1&q=1|r=0)U(!EG(A[(E[(p=1&q=1&r=0)U(p=0|q=1|r=0))]U(EF(p=0&q=0&r=0))))))]U(E[(AX(E[(E[(AG(p=0|q=1&r=0))U(!EG(p=0&q=1&r=1)))]U(!EX(p=1&q=1|r=0))))]U(!EX(p=1&q=1|r=0))))]U(!AG(p=0&q=1|r=0))))))))))]

a) ff1.

EF(!AF(E[(AF!(p=0&q=0|r=0))U(EX(E[(AG(E[(AF(E[(!AX(!E[(A[EX(p=0&q=1|r=1)U(EF(p=1|r=1)))])U(E[AX(p=1|q=1)U(!AG(p=1&q=1&r=0))])]))U(A[(EF(A[(E[(AF(p=1&q=0&r=1))U(AX!(p=0|q=1|r=0))])]U(AG!(p=1&q=0&r=0))))]U(AF(A[(p=0&q=0|r=0)U(!EX(A[(AG(E[(AG(A[(!EX((E[(E[AG(p=0&q=1&r=1)U(p=1|q=1|r=0)])U(EX(p=0&q=1|r=1))])&(EX(E[(E[(E[(EX!(p=1&q=0&r=1))]U(AX(p=1|q=1|r=1)))]U((EX!(p=1|q=0&r=1)))]U(EX(A[(E[(!E[(EX(p=1&q=0&r=1))]U(EX!(p=0|q=1|r=0)))]U(EX(p=0&q=1|r=0))))]U(E[(E[EX(p=1&q=1&r=0)U(p=0|q=1&r=0)])U(p=1&q=0&r=0)])))))))]U(A[(E[(p=0&q=0|r=1)U(EG(p=0|q=1&r=1))]U(A[(AF(p=1&q=0)U(p=1&r=0))])))]U(E[(A[(AG(p=0&q=1&r=1))U(EX(p=1&q=0&r=1))])]U(E[(AG(p=1|q=1&r=1))U(!EX(p=1|q=0|r=1))])))]U(E[(AX(E[(E[(AG(p=0&q=1|r=1))U(EG!(p=0&r=1))])]U(A[(AF(p=1&q=0)U(p=1&r=0))])))]U(E[(AX(E[(E[(AG(p=0&q=1|r=1))U(EG(p=0|q=1&r=1))]U(E[(AX(p=1|q=1)U(!AG(p=1&q=1&r=0))])]U(EX!(p=1&q=1|r=1))]U(!EX(E[(A[(AX!(p=1&q=1&r=0))U(AF!(p=1|q=0|r=0))])]U(EX!(p=0|q=1|r=1))])))))))))

b) ff2.

Figure 3. The two CTL formulas.

a) MC result for ff1.

b) MC result for ff2.

Figure 4. The model checking results on the K and the two CTL formulas.

Now, the CTL formula ff_2 is shown in Figure 3-b). We also use NuXMV to conduct CTL model checking. The result is illustrated by Figure 4-b), indicating K does not satisfy ff2. Thus, another record is added to A, and this record contains the three fields: "000010010010111011010122124303243", "EF(!AF(E[(AF!(p=0&q=0|r=0))U……" and "false", where the first two fields are the features and the last one is the label when ML training comes.

In this way, we can produce more records to build A. Example 1 is over.

In our scikit-learn and graph lab experiments, the above three data sets provide the raw data for machine learning. As for the division of training sets and test sets, we use the same way to do it with [41], due to the same reason with [41]. In fact, the hypermeter "fraction" means the ratio of number of records in a training set to number of records in a data set.

4.4. Experimental Results

In this subsection, we will conduct our experiments on A1 with graph lab.

First, Tables 1 and 3 compares the four algorithms on A1 in terms of the optimal accuracies and average time for one record, and Table 2 shows the values of the hyper-parameters while the results of Table 1 occur. In this tables, the used metrics are AUC, TNR, TPR, precision, accuracy and running speed. The first five metrics are related to the power of the new method, and the last one indicates the efficiency of the new method.

Table 1. Graph Lab experiments where length of each formula is 500 (A1 is used).

Algorithms	RF	BT	DT	LR
AUC	0.74	0.96	0.59	1
Specificity (TNR)	0.482	0.615	0.172	1
Sensitivity (TPR)	1	1	1	1
Precision	0.916	0.939	0.889	1
Predictive Accuracy	0.922	0.944	0.891	1
Running time per record (in second)	0.000029	0.000028	0.000023	0.000022

Table 2. What are the values of the parameters and the hyper-parameters if the illustrations of Table 1 occur.

Algorithms	RF	BT	DT	LR
Training record #	6268	6285	6291	6271
Testing record #	732	715	709	729
Seed	456	2242	2289	2233
Fraction	0.89	0.9	0.9	0.9

Table 3. Graph Lab experiments where length of each formula is 500 (A1 is used).

Length of formulas, i.e., L	Average running time (t_1) of NuXMV for one pair of Kripke structure and formula (s)	ML algorithms	Average predictive time (t_2) of the new method based on ML for one record (s)	t_1 / t_2
L=500	0.0762	RF	0.000029	2628
		BT	0.000028	2721
		DT	0.000023	3313
		LR	0.000022	3464

Let NuXMV's accuracy be 1, the approach based on machine learning has a highest accuracy of 100% (L=500), when LR is used, as listed in Table 1. In other words, the newly proposed approach has approached the latest algorithm in the classical MC technique, in terms of predictive accuracy. The reason is that CTLMC is a strongly learnable problem, causing a ML-based approach has a good learning ability.

See Table 3, compared to state of the art of approach in the classical CTLMC technique, the machine-learning-based approach is several thousand times faster, due to strongly learnable problem again.

4.5. Discussions

First, A1 has seven thousand records, including 5790 positive samples and 1210 negative ones, ensuring the generalization ability and preventing the data from imbalance.

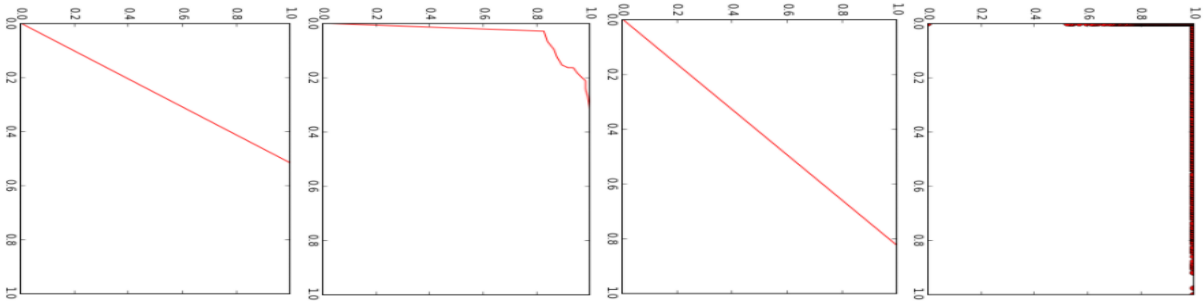
Second, as depicted in Table 1, the different algorithms have the different optimum predictive accuracies. RF, BT and DT have the low accuracies, and they are wholly unsuited to approximate CTLMC. In contrast, LR is the preferred algorithm.

4.6. More Comparisons Among the Different Machine Learning Algorithms

In addition to accuracy, Area Under Curve (AUC), Receiver Operating Characteristic (ROC) curve, specificity, i.e., True Negative Rate (TNR), sensitivity, i.e., True Positive Rate (TPR) and precision are often used as the metrics for evaluating binary classifiers. Table 1 depicts TPR, TNR and precision for the four optimum classifiers originated from the four machine learning algorithms (L=500). And Figure 5 shows their ROC curves, and their AUC values are given in Table 1. It is widely known that, AUC is the entire area beneath a ROC curve and AUC measures how well a model is able to distinguish between classes. Generally speaking, the classifier which has a bigger AUC value is the better one. As illustrated in Figure 5 and Table 1, LR shows a better performance.

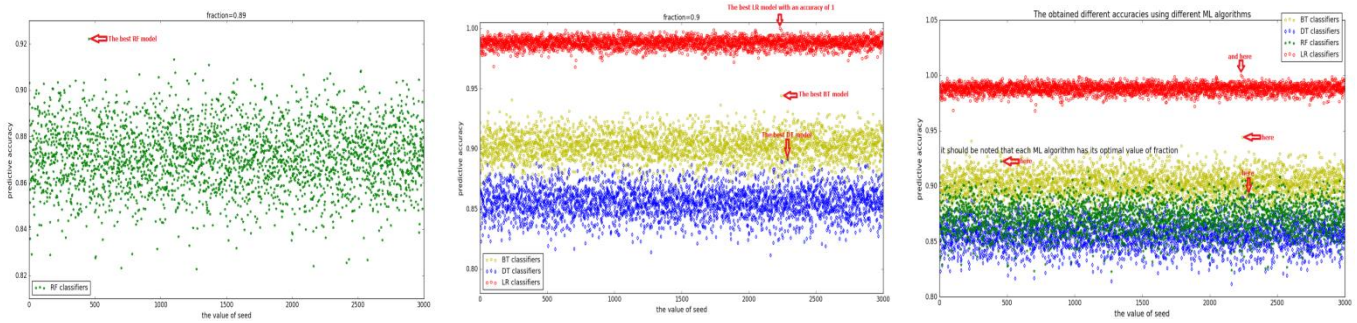
In the above discussion, our Graph-lab program automatically tunes the two hyper-parameters' values and searches the optimum accuracy for every algorithm. And the different algorithm has its different optimal accuracy although the hyper-parameters are set to the different values.

Now, the obvious question becomes: what happen if all the algorithms have the same values of the hyper-parameters? Figure 6 shows the distribution of accuracies varying in space of a great number of values of the several hyperparameters, for each algorithm. For example, RF get its optimal accuracy when fraction=0.89, and Figure 6-a) shows all the three thousand values of accuracy when fraction=0.89. Obviously, the optimal value of accuracy occurs, when fraction=0.89 and seed=456. And the location pointed by a red arrow indicates this situation. Similarly, Figure 6-b) shows us what will happen if other three ML algorithms are used, respectively. And the third sub-figure makes a summary. In this case, Figure 6 indicates that LR is better than RF, BT and DT in terms of the overall accuracy (L=500).



a) RF. b) BT. c) DT. d) LR.

Figure 5. Roc curve of the optimal classifiers, (L=500 and A1 is used).



a) Obtained various classifiers when fraction=0.89. b) Obtained various classifiers when fraction=0.9. c) Obtained various classifiers (a combination of the above two sub-figures).

Figure 6. Comparison of performance of ML algorithms with the same values of hyper-parameters (seed and fraction) (L=500 and A1 is used).

4.7. More Comparisons on the Different Data

Until now, data set is fixed and test set varies with the change of the hyper-parameters' values, in order to explore the new method based on the different ML algorithms in terms of ability and the efficiency on a given data set. Next, we will use a given data set A2 to test the above four optimum classifiers, in order to explore the above optimal ML models originating from the different ML algorithms in terms of generalization ability and the efficiency on a given testing set.

As shown in Tables 4 and 5, LR provides the fast and accurate classifier, where the LR optimal model is best again, although the other three classifiers are also fast, as shown in Table 5. It should be noted that the first column in Table 5 shows the average running time for one record using NuXMV, and the third column shows the average running time for one record using different ML algorithm. How many times has the efficiency been improved? One can get the answers by dividing the value of the first column by a value of the third column.

Table 4. Graph Lab experiments where length of each formula is 500 (A2 is used).

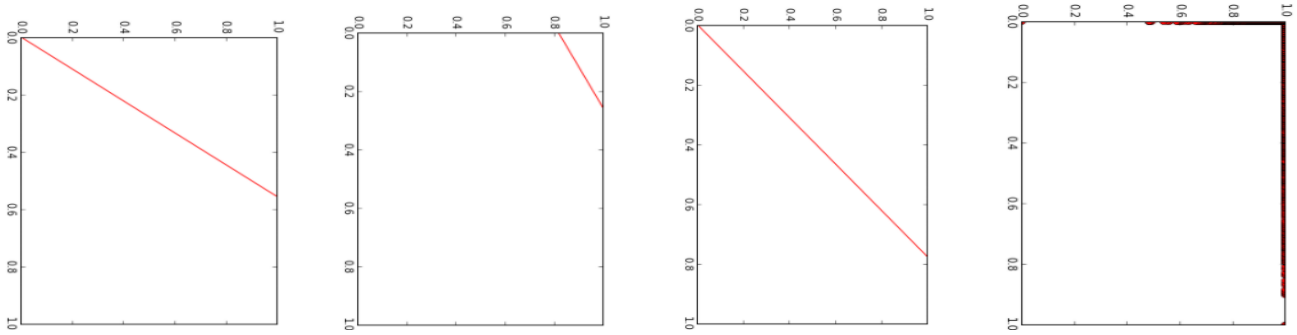
Algorithms	RF	BT	DT	LR
Prediction Accuracy	0.852	0.904	0.852	1
Running time per record (in second)	0.000031	0.000032	0.00003	0.00003
AUC	0.721	0.976	0.61	1
Specificity (TNR)	0.221	0.495	0.221	1
Sensitivity (TPR)	1	1	1	1
Precision	0.846	0.894	0.846	1

Table 5. Compared with NuXMV, the new method enhances the efficiency of CTL model Checking (L=500 and A2 is used).

Average running time (t_1) of NuXMV for one pair of Kripke structure and formula (s)	ML algorithms	Average predictive time (t_2) of the new method based on ML for one record (s)	t_1/t_2
0.073	RF	0.000031	2355
	BT	0.000032	2281
	DT	0.00003	2433
	LR	0.00003	2433

Furthermore, Figure 7 provides their ROC curves. As shown in this figure, LR does the best again.

Now, all the four classifiers run on A3. Table 6 shows the results, and the performance is unacceptable at all!



a) RF. b) BT. c) DT. d) LR.

Figure 7. Roc curve of the optimal classifiers, (L=500 and A2 is used).

Similar to the approximate LTLMC [41], the combination of Tables 4 and 6 demonstrates the one thing again: the new method would not perform well, if an appropriate data set could not be constructed beforehand. See [41] for more details.

Table 6. Graph Lab experiments where length of each formula is 500 (A3 is used).

Algorithms	RF	BT	DT	LR
Prediction Accuracy	0.41	0.41	0.41	0.41

It should be noted that, no benchmark set is used in this work, because up to now there is no such thing as a benchmark platform or original data available for the approximate CTL model checking based on machine learning. Why not use an existing benchmark set in term of the classical CTL model checking? The reason is that the different formulas have different lengths in the existing related benchmark sets, so that they cannot be employed directly to perform training, testing and predictions. After all, not enough formulas are available in them, if only the formulas which have the same length are selected for ML training and testing. Furthermore, an ML experiment on a very small data set will be unconvincing and infeasible.

4.8. More Comparisons on the Different Platforms

In the prior context, the NuXMV model checker employs the later CTLMC algorithm [9], whereas graph lab does not employ latest machine learning algorithm. The thing is the new method employing a common machine learning algorithm performs better than the latest CTLMC algorithm, demonstrated by the above experiments. Thus, the conclusion about the advantage of the new method is convincing.

Now, Let us relax our constraints, and see what happens if some latest ML algorithms are used. In this subsection, our machine learning experiments will be conducted on scikit-learn. And the sixteen ML algorithms are employed, respectively.

Table 7. The optimal accuracies and the evaluation indexes on Scikit-learn.

Algorithms	optimal accuracies	running time	AUC	sensitivity	specificity	precision	Algorithms	optimal accuracies	running time	AUC	sensitivity	specificity	precision
RF	1	1.14E-5	1	1	1	1	PA	1	1.26E-6	1	1	1	1
BT	1	4.06E-6	1	1	1	1	SGD	1	1.07E-6	1	1	1	1
ET	1	1.28E-5	1	1	1	1	L-SVC	1	7.96E-7	1	1	1	1
DT	1	1.49E-6	1	1	1	1	N-SVC	1	1.82E-5	1	1	1	1
KNN	1	5.79E-5	1	1	1	1	C-SVC	0.977	1.96E-5	0.875	1	0.75	0.976
NC	1	2.01E-6	1	1	1	1	GPC	1	1.78E-5	1	1	1	1
LR	1	8.68E-7	1	1	1	1	NB_BER	1	1.57E-6	1	1	1	1
RIDGE	1	6.25E-7	1	1	1	1	NB_GAU	0.925	2.82E-6	0.94	0.88	1	1

Table 8. The values of the hyper-parameters when the results in table 7 occurs.

Algorithms	fraction	seed	seed_clf
RF	0.82	3	29
BT	0.88	69	0
ET	0.83	69	10
DT	0.88	69	0
KNN	0.9	2218	——
NC	0.81	2112	——
LR	0.86	743	——
RIDGE	0.81	431	——
PA	0.88	4	77
SGD	0.89	12	34
L-SVC	0.82	431	——
N-SVC	0.83	431	——
C-SVC	0.89	2519	——
GPC	0.87	270	——
NB_BER	0.81	431	——
NB_GAU	0.9	1062	——

Table 9. Compared with NuXMV, the new method based on scikit-learn enhances the efficiency of CTL model Checking.

algorithms	For a pair of Kripke structures and CTL formulas, the average running time t_1 with NuXMV (seconds)	Average prediction time t_2 (seconds) for one record with ML algorithm on scikit-learn	t_1/t_2
RF	0.0762	1.14025E-05	6927
BT	0.0762	4.05844E-06	19050
ET	0.0762	1.27909E-05	5862
DT	0.0762	1.49E-06	50800
KNN	0.0762	5.78504E-05	1314
NC	0.0762	2.01208E-06	38100
LR	0.0762	8.68075E-07	87586
RIDGE	0.0762	6.25226E-07	120952
PA	0.0762	1.25949E-06	58615
SGD	0.0762	1.07059E-06	69273
L-SVC	0.0762	7.96381E-07	95250
N-SVC	0.0762	1.81793E-05	4233
C-SVC	0.0762	1.95727E-05	3810
GPC	0.0762	1.77928E-05	4281
NB_BER	0.0762	1.57289E-06	47625
NB_GAU	0.0762	2.8167E-06	27214

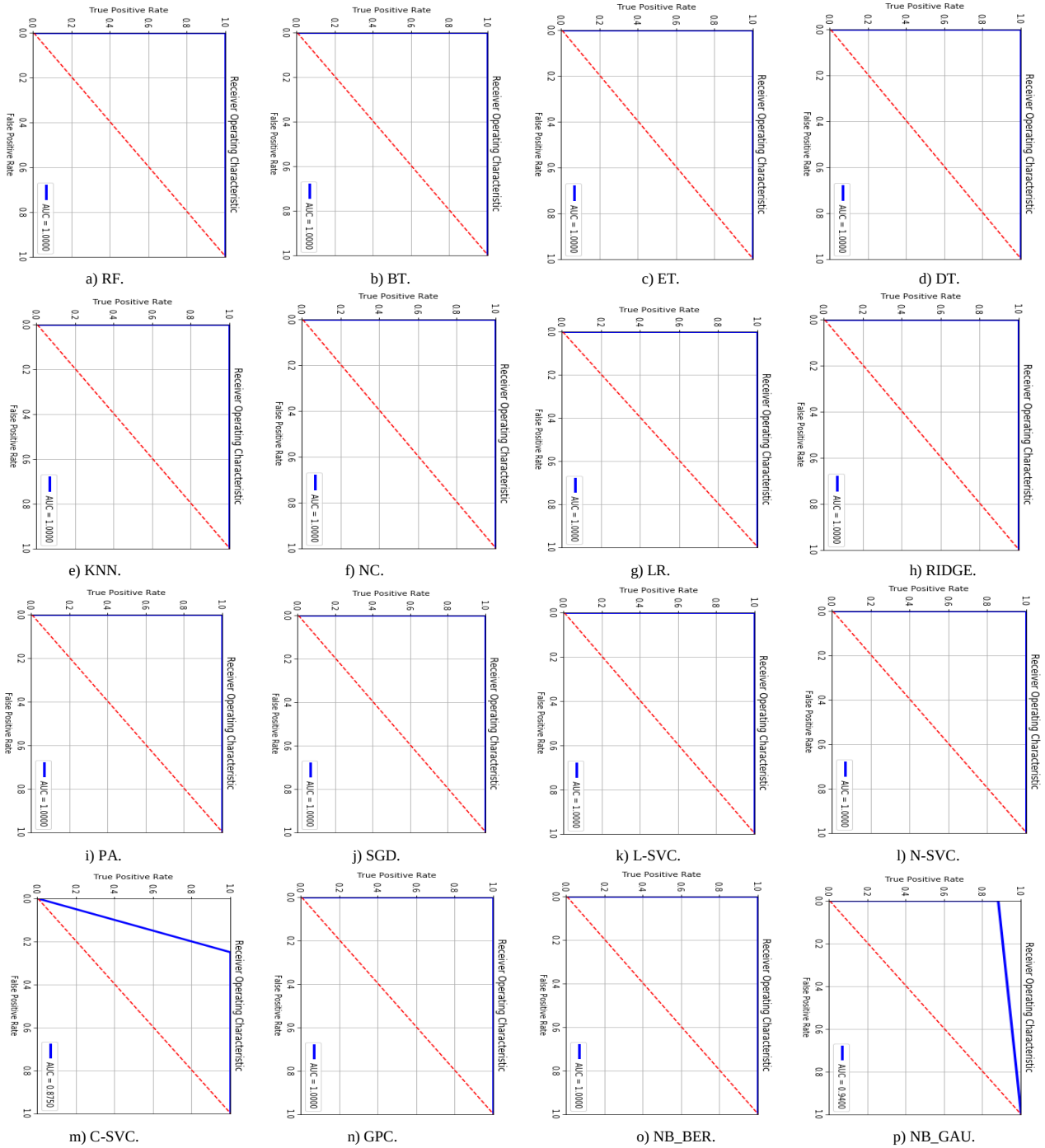
Table 7 depicts the experimental results, and Table 8 describes the corresponding values of the hyper-parameters. Furthermore, Figure 8 illustrates the sixteen ROC curves. In addition, Table 9 compares the new method based on scikit-learn and state of the art of the CTLMC in terms of the efficiency. As shown in Tables 7 and 9, some algorithms naming "regression" perform well, such as LR and RIDGE. Especially for RIDGE, it has an optimal accuracy of 1, as well as RIDGE is over 100000 times faster than the latest CTL model checking algorithm. The reason is that RIDGE regression inherits the advantage of linear regression, i.e., high speed, and it reduces over-fitting by adding regularization term L2. Obviously, our experiments demonstrate these two "regression"-based optimal classifiers are very suitable for the approximate CTL model checking.

5. Comparisons between the New Method and the Relevant Approaches

5.1. The Studies Related with both Machine Learning and Model Checking

In an existing study [41], the authors investigate some researches related with both MC and ML [6, 7, 10, 11, 12, 13, 26, 29, 30, 32, 36, 37].

However, there exist fundamental differences between the new approach and these related ones [41]. These studies do not directly use machine learning to perform model checking. However, the new method can do this, as shown in Figure 1.



a) RF. b) BT. c) ET. d) DT.
e) KNN. f) NC. g) LR. h) RIDGE.
i) PA. j) SGD. k) L-SVC. l) N-SVC.
m) C-SVC. n) GPC. o) NB_BER. p) NB_GAU.

Figure 8. Roc curve of the sixteen optimal classifiers on scikit-learn.

5.2. Comparison to the Existing CTLMC

In fact, the newly proposed approximate CTLMC technique has some disadvantages.

Similar to the approximate LTLMC [41], the newly proposed approximate CTL one is not recommended for users of safety-critical systems, at present. Furthermore, no counterexample is generated, as well as a massive data set which has been conducted beforehand, is needed.

However, the most important advantage of the new method is that the SE dissolves into nothingness.

All existing CTL model checking approaches needs to explore exhaustively state space, whereas the newly proposed approximate CTLMC technique based on machine learning never explores any state space, as well as it only perform predictions based on data. It is this reason that causes the well performance in our experiments.

It should be noted that, generally speaking, the more serious the state space explosion problem is, the more memory the computation consumes. This phenomenon indirectly leads to a sharp decline in efficiency. In our experiments, the length of each CTL formula is 500. Even so, the new method has shown tens of thousands of improvements over the traditional methods, in terms of efficiency. Image what the advantage of the new method should be, if the CTL formulas would be longer and longer.

In summarize, it is safe to say that the newly proposed approximate CTL model checking technique and the classical CTL model checking technique complement each other. They have a complementary relationship rather than the alternative one.

5.3. Comparison to the Approximate LTLMC

In an existing study [41], the four ML algorithms were employed and an approximate LTL model checking method was presented. Lucky, the SE in LTL model checking is totally avoided with this method in our experimental conditions, although it seems that the cost is inevitable, and the usage is limited from the point of view of the current technical level. However, it is the first time that hopefully model checking will not be bothered by the SE problem, although it is just an approximate solution rather than the accurate one.

Just as LTL model checking and CTL model checking take part in making up of temporal logic model checking, one sub-problem named SE in LTL model checking and another sub-problem named SE in CTL model checking forms the major aspects of SE problem in temporal logic model checking. Obviously, it seems that something is missing for us, in terms of the approximate LTL model checking for dealing with the SE problem. Yes, it is another sub-problem: approximate model checking for CTL.

Compared with the ML-based approach for predict LTL model checking results [41], the newly proposed method can predict CTL model checking results, using machine learning algorithms. In this way, these two approaches make up a complete approximate model checking technique.

Table 10 summarizes the comparisons between these two methods. Obviously, these two works aim to combat the different sub-problem of the SE problem in model checking, as well as employ the different machine learning algorithms as the core engines. And the different experimental results are obtained.

Table 10. Comparison among the approximate LTL Model checking method and the new one.

	The approximate LTL model checking algorithm in [41]	**The new method**
State explosion problem	Has been avoided	Has been avoided
Time complexity	polynomial	polynomial
For LTL model checking or CTL one?	LTL model checking	CTL model checking
How many ML algorithms are compared?	Four machine learning algorithms	Sixteen machine learning algorithms
Which ML algorithm is the most suitable?	LR	RIDGE
The max predictive accuracy	1	1
How many times faster than classical MC?	At most 6.3 million times, due to SE and time complexity.	At most 120 thousand times, due to SE

In fact, CTL model checking needs an approximate solution more than LTL model checking does, while a temporal logic formula is very long. The reason is as follows.

As shown in this table, the combination of the two factors, i.e., avoiding SE and the reduction of time complexity, causes the efficiency is improved by million times at most in terms of LTL. In comparison, only the one factor, i.e., avoiding SE, causes the efficiency is improved by hundreds of thousand times at most in terms of CTL. Note that SE problem usually has a greater effect on memory than on time, generally speaking. Obviously, the SE problem is so serious that a serious lack of memory occurs, and it is the serious shortage of memory rather than time complexity itself leads to the decrease of time efficiency, while a CTL formula has a length of 500. Thus, a long CTL formula needs a way to avoid SE more than a long LTL formula does, while model checking is performed.

In a word, the topics, the used algorithms and the results are all different between the approximate LTL model checking and the approximate CTL one. And it is necessary to study the two temporal logics separately.

It should be noted that there exists some works also naming “approximate model checking”. However, they are the different things at all, and they just have the same title with our works, as analyzed in [41].

In summary, this paper proposes an approximate CTL model checking technique while [41] put forwarded an approximate LTL model checking one. As a result, the approximate model checking technique is formed.

5.4. Comparison with Some Works Based on DNA Computing

As far as model checking is concerned, the SE restricts the scale of MC applications, which can be alleviated rather than avoided in classical computing. To this end, some studies explore a different way.

Deoxyribo Nucleic Acid (DNA) molecules can be employed to perform model checking. This idea was first proposed by Prof. Emerson, the Turing Award winner [23]. In one existing research, the authors proposed several DNA-computing-based approaches to conduct model checking, in which DNA molecules exhibite tremendous power in terms of parallel computing [39, 40, 42]. As a result, DNA model checking is forming.

In terms of computing environments of model checking, the platforms used in the above methods are DNA computing devices rather than electronic computers. This is the essential difference between the approximate model checking and DNA model checking. However, how to avoid the SE problem? It is a common goal and mission for both the approximate model checking and DNA model checking. Considering the former method run on electronic computers rather than

DNA computing devices, it is safe to say that the former method has a wider prospect, compared to the latter one.

6. Conclusions

This study pioneers an approximate CTL model checking technique, avoiding SE in CTLMC completely. Up to now, state of the art of CTL model checking is inadequate to verify very large-scale OSs against the complex branch temporal properties in practice, due to the SE problem. This background will help us understand the benefit of using the new method.

Acknowledgment

This study has been supported by National Natural Science Foundation of China under grant U1204608.

References

[1] Abdelrazaq D., Abu-Soud S., and Awajan A., "A Machine Learning System for Distinguishing Nominal and Verbal Arabic Sentences," *The International Arab Journal of Information Technology*, vol. 15, no. 3A, pp. 576-584, 2018.

[2] Abe T., Ugawa T., and Maeda T., "Reordering Control Approaches to State Explosion in Model Checking with Memory Consistency Models," *in Proceedings of Working Conference on Verified Software: Theories, Tools, and Experiments*, Heidelberg, pp. 170-190, 2017.

[3] Apple Incorporation, "apple/turicreate: Turi Create simplifies the development of custom machine learning models," Retrieved from: https://github.com/apple/turicreate/, Last Visited, 2020.

[4] Attie A., "Synthesis of Large Dynamic Concurrent Programs from Dynamic Specifications," *Formal Methods in System Design*, vol. 48, no. 1-2, pp. 94-147, 2016.

[5] Ball T., Cook B., Levin V., and Rajamani S., "SLAM and Static Driver Verifier: Technology Transfer of Formal Methods inside Microsof," *in Proceedings of International Conference on Integrated Formal Methods*, Canterbury, pp 1-20, 2004.

[6] Behjati R., Sirjani M., and Ahmadabadi M., "Bounded Rational Search for on-the-Fly Model Checking of LTL Properties," *in Proceedings of International Conference on Fundamentals of Software Engineering*, Kish Island, pp. 292-307, 2009.

[7] Belzner L. and Gabor T., "Bayesian Verification under Model Uncertainty," *in Proceedings of EEE/ACM 3rd International Workshop on Software Engineering for Smart Cyber-Physical Systems*, Buenos Aires, pp. 10-13, 2017.

[8] Benari-Ari M., Pnueli A., and Manna Z., "The Temporal Logic of Branching Time," *Acta Informatica*, vol. 20, no. 3, pp. 207-226, 1983.

[9] Benjamin B., Marco B., Roberto C., Alessandro C., Michele D., Marco G., Alberto G., Ahmed I., Cristian M., Andrea M., Sergio M., Marco R., Mirko S., Stefano T., Gianni Z.,"The Nuxmv Model Checker," FBK, https://es-static.fbk.eu/tools/nuxmv/, Last Visited, 2020.

[10] Bortolussi L., Milios D., and Sanguinetti G., "Machine Learning Methods in Statistical Model Checking and System Design-Tutorial," *in Proceedings of 6th International Conference Runtime Verification*, Vienna, pp. 323-341, 2015.

[11] Bortolussi L. and Sanguinetti G., "Learning And Designing Stochastic Processes from Logical Constraints," *in Proceedings of International Conference on Quantitative Evaluation of Systems*, Buenos Aires, pp. 89-105, 2013.

[12] Bortolussi L. and Silvetti S., "Bayesian Statistical Parameter Synthesis for Linear Temporal Properties of Stochastic Models," *in Proceedings of International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, Thessaloniki, pp. 396-413, 2018.

[13] Brázdil T., Chatterjee K., Chmelík M., Forejt V., Křetínský J., Kwiatkowska M., Parker D., and Ujma M., "Verification of Markov Decision Processes using Learning Algorithms," *in Proceedings of International Symposium on Automated Technology for Verification and Analysis*, Sydney, pp. 98-114, 2014.

[14] Clarke E., "Model Checking: My 30 Year Quest to Conquer the State Explosion Problem," Retrieved from: http://www.cs.cmu.edu/~emc/15414-f11/lecture/lec27_MC.pdf, Last Visited, 2020.

[15] Clarke E., "Model Checking Overview," Retrieved from: http://www.cs.cmu.edu/~emc/15-398/lectures/overview.pdf, Last Visited, 2020.

[16] Clarke E. and Emerson E., "Design and Synthesis of Synchronization Skeletons Using Branching Time Temporal Logic," *in Proceedings of Workshop on Logic of Programs*, Yorktown Heights, pp. 52-71, 1981.

[17] Clarke E., Khaira M., and Zhao X., "Word Level Model Checking-Avoiding the Pentium FDIV Error," *in Proceedings of 33rd Design Automation Conference Proceedings*, Las Vegas, pp. 645-648, 1996.

[18] Clarke E., Klieber W., Nováček M., and Zuliani P., *Tools for Practical Software Verification*, Springer Link, 2012.

[19] Dobrikov I. and Leuschel M., "Optimising The Prob Model Checker for B Using Partial Order Reduction," *Formal Aspects of Computing*, vol. 28, no. 2, pp. 295-323, 2016.

[20] Dutra A., "Software Model Checking:

High-Assurance Software Design," NASA, Retrieved from: https://ti.arc.nasa.gov/tech/rse/vandv/software-model-checking/, Last Visited, 2020.

[21] Elkader K., Grumberg O., Pasˇareanuˇ C., and Shoham S., "Automated Circular Assume-Guarantee Reasoning," *Formal Aspects of Computing*, no. 30, pp. 571-595, 2018.

[22] Emerson E. and Clarke E., "Using Branching Time Temporal Logic to Synthesize Synchronization Skeletons," *Science of Computer Programming*, vol. 2, no. 3, pp. 241-266, 1982.

[23] Emerson E., Hager K., and Konieczka J., "Molecular Model Checking," *International Journal of Foundations of Computer Science*, vol. 17, no. 04, pp. 733-742, 2006.

[24] Fabian P., Gaël V., Alexandre G., Vincent M., Bertrand T., Olivier G., Mathieu B., Peter P., Ron W., Vincent D., Jake V., Alexandre P., David C., Matthieu B., Matthieu P., Édouard D., "Scikit-Learn: Machine Learning in Python," retrieved from: https://scikit-learn.org/stable/, Last Visited, 2021.

[25] Groefsema H., Van-Beest N., and Aiello M., "A Formal Model for Compliance Verification of Service Compositions," *IEEE Transactions on Services Computing*, vol. 11, no. 3, pp. 466-479, 2016.

[26] Haim S. and Walsh T., "Restart Strategy Selection Using Machine Learning Techniques," *in Proceedings of the International Conference on Theory and Applications of Satisfiability Testing*, Swansea, pp. 312-325, 2009.

[27] Halleux P., Rajamani S., Ball T., and Hoare T., Microsoft Research, "SLAM," Retrieved from: https://www.microsoft.com/en-us/research/project/slam/, Last Visited, 2020.

[28] Kojima H., Nagashima Y., and Tsuchiya T., "Model Checking Techniques for State Space Reduction in Manet Protocol Verification," *in Proceedings of IEEE International Parallel and Distributed Processing Symposium Workshops*, Chicago, pp. 509-516, 2016.

[29] Liang J., Ganesh V., Poupart P., and Czarnecki K., "Learning Rate Based Branching Heuristic for SAT Solvers," *in Proceedings of International Conference on Theory and Applications of Satisfiability Testing*, Bordeaux, pp. 123-140, 2016.

[30] Liang J., Oh C., Mathew M., Thomas C., Li C., and Ganesh V., "Machine Learning-Based Restart Policy for CDCL SAT Solvers," *in Proceedings of International Conference on Theory and Applications of Satisfiability Testing*, Oxford, pp. 94-110, 2018.

[31] Musuvathi M., Park D., Chou A., Engler D., and Dill D., "CMC: A Pragmatic Approach to Model Checking Real Code," *ACM SIGOPS Operating Systems Review*, vol. 36, pp. 75-88, 2002.

[32] Pedro A., Crocker P., and Simão M., "Learning Stochastic Timed Automata from Sample Executions," *in Proceedings of International Conference on Leveraging Applications of Formal Methods*, Heraklion, pp. 508-523, 2012.

[33] Pnueli A., "The Temporal Logic of Programs," *in Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, Providence, pp. 46-57, 1977.

[34] Ročkai P., Barnat J., and Brim L., "Improved State Space Reductions for LTL Model Checking of C and C++ Programs," *in Proceedings of NASA Formal Methods Symposium*, Moffett Field, pp. 1-15, 2013.

[35] Sack H., SciHi BlogSciHi Blog, "The Pentium FDIV Bug," Retrieved from: http://scihi.org/the-pentium-fdiv-bug/, Last Visited, 2015.

[36] Sanguinetti G ., "Machine Learning Methods for Model Checking in Continuous Time Markov Chains," https://www.cs.ox.ac.uk/seminars/1195.html, Last Visited, 2020.

[37] Tulsian V., Kanade A., Kumar R., Lal A., and Nori A., "MUX: Algorithm Selection for Software Model Checkers," *in Proceedings of the 11th Working Conference on Mining Software Repositories*, New York, pp. 132-141, 2014.

[38] Zheng H., Zhang Z., Myers C., Rodriguez E., and Zhang Y., "Compositional Model Checking of Concurrent Systems," *IEEE Transactions on Computers*, vol. 64, no. 6, pp. 1607-1621, 2015.

[39] Zhu W., Feng C., and Wu H., "Model Checking Temporal Logic Formulas Using Sticker Automata," *BioMed Research International*, 2017.

[40] Zhu W., Han Y., and Zhou Q., "Performing Ctl Model Checking Via Dna Computing," *Soft Computing*, vol. 23, no.12, pp. 3945-3963, 2019.

[41] Zhu W., Wu H., and Deng M., "LTL Model Checking Based on Binary Classification of Machine Learning," *IEEE Access*, vol. 7, pp. 135703-135719, 2019.

[42] Zhu W., Zhou Q., and Zhang Q., "A LTL Model Checking Approach Based on DNA Computing," *Chinese Journal of Computers*, vol. 39, no. 12, pp. 2578-2597, 2016.

Weijun Zhu received a Ph.D. degree in Computer Science from Xi-Dian University in 2011. Afterwards, he finished postdoctoral researches twice. Subsequently, Dr. Zhu conducted a two-year study at Peking University and Tsinghua University, as a visiting scholar. Currently, he is working as an associate professor at Zhengzhou University. Until now, Dr. Zhu has authored and co-authored more than eighty papers in some journals and conferences. His research interests include machine learning applications, formal methods, bioinformatics and DNA computing, and information security.

Huanmei Wu received a Ph.D. in Computer Science from Northeastern University (Boston, MA) in 2005. Currently, she is the professor and Department Chair of Health Services Administration and Policy, as well as the Assistant Dean for Global Engagement at Temple University College of Public Health. Dr. Wuis an interdisciplinary researcher and educator in computer science, informatics, biomedical science, and public health, partnering with academia, industries, and local communities. She has served in multiple academic leadership positions and directed various educational programs and research projects.