A Novel Approach of CTL Model Checking Based on Probe Machine

Dong Wang[†], Jing Liu[*†], Jin Xu[*‡], Haiying Sun[†], Jiexiang Kang[§]
[†] *Shanghai Key Laboratory of Trustworthy Computing, East China Normal University, Shanghai, China*
[‡] *Key Laboratory of High Confidence Software Technologies, Peking University, Beijing, China*
[§] *China Aeronautical Radio Electronics Research Institute, Shanghai, China*

***Abstract*—Model checking has established as an effective method for automatic system analysis and verification. It is making its way into many domains and methodologies. However, the state space may be extremely large for many practical systems, and this is a major limitation for state-space search algorithms in model checking. We have proposed a novel computing model called probe machine in 2016, which is a fully parallel computing model. In comparison to the Turing machine, it can solve the graph search problems efficiently, which can overcome the existing model checking limitations. In this paper, we propose a novel approach to perform Computation Tree Logic (CTL) model checking based on the mathematical model of probe machine, which can verify all CTL properties. It can greatly reduce the verification time for systems with large state space. We develop a model checker called CTL2PROBE based on our approach and the experimental results show that our approach is better than NuSMV.**

***Index Terms*—Model Checking, Probe Machine, Graph Search, Computation Tree Logic**

I. Introduction

Model Checking is a technology that can verify whether a system satisfies the given property automatically [1]. The way of verification is exploring all possible system states in a brute-force manner [2]. In this way, it is a real challenge to examine the largest possible state spaces that can be treated with current means, i.e. processors and memories. Therefore, the chief limitation is the state explosion problem where the size of the global state graph grows exponentially with the size of the program itself [3].

In order to deal with the state explosion problem, many approaches have been proposed. Ordered Binary Decision Diagram (OBDD) is proposed by McMillan to represent state space, improving the scale of verified system [4], [5]. Another successful technique for dealing with state explosion is based on the partial order reduction [6], [7]. It exploits the independence of concurrently executed events. Although symbolic representations and partial order reduction have greatly increased the size of the systems that can be verified, many realistic systems are still too large to be handled. Thus, some researchers have turned their attention to other novel computation architectures like molecules and DNA to break through the limitations of Turing machines.

[*] Corresponding author: Jing Liu, jliu@sei.ecnu.edu.cn, Jin Xu, jxu@pku.edu.cn

DOI reference number: 10.18293/SEKE2021-139

In 2006, Turing Award winner Allen Emerson use DNA molecules to conduct CTL model checking for the first time [8]. In this work, he proposes a DNA-computing-based method and designs a checking algorithm for CTL formula $\mathbf{EF}p$ where Kripke structures are used to model the system and the states and transitions of the model are encoded into DNA strands. It permits a very compact representation of extremely large state graphs. For example, a graph of size 10^{16} states can be represented within 0.01 liter of DNA. Therefore, the parallelism of DNA computing and the vast storage of DNA molecules provide the opportunity to break the state space limitation on traditional electronic computers.

Probe machine is a mathematical model proposed by Xu in 2016 [9]. It can be implemented by using nano-DNA probe technologies. It is a fully parallel computing model in the sense that it can simultaneously process multiple pairs of data, rather than sequentially process every pair of linearly adjacent data. Many NP-complete problems, i.e., the graph coloring, Hamilton cycle problems, traveling salesman problem [10], and postman problem [11] have been solved based on the probe machine. Probe machine can enumerate all solutions to these problems by only one probe operation.

Similarly, the way of CTL model checking is exploring all possible system states in a brute-force manner and finding a path satisfying the given property. Therefore, we propose a novel method to perform CTL model checking based on the probe machine. Compared to traditional model checkers, our approach can relieve the state explosion problem and reduce the verification time.

In summary, this paper makes the following contributions:

- We design a mapping algorithm to transform the model of Kripke structure into the data library and probe library that can run directly on the probe machine.
- We develop a model checker called CTL2PROBE to simulate the probe machine, which takes the model as input and obtains all feasible paths or counterexamples.
- We conduct several experiments based on different numbers of states. Compared to NuSMV, the experiment results prove the feasibility and efficiency of our approach.

The rest of this paper is organized as follows. Section II briefly introduces CTL model checking and the concept of probe machine. Section III presents our approach and provides complexity analysis. Section IV introduces the model checker

CTL2PROBE based on our approach, and the experimental results show that our approach is better than NuSMV. Section V concludes our work.

II. Preliminaries

A. CTL Model Checking

A Kripke structure is a variation of the transition system, originally proposed by Saul Kripke [12], used in model checking [13] to represent the behavior of a system. It consists of a graph whose nodes represent the reachable states of the system and whose edges represent state transitions, together with a labeling function which maps each node to a set of properties that hold in the corresponding state.

A Kripke structure is defined as a four-tuple

$$\mathcal{M} = \{S, I, R, L\}$$

- a finite set of states S.
- a set of initial states $I \subseteq S$.
- a transition relation $R \subseteq S \times S$.
- a labeling function $L : S \to 2^{AP}$.

Computation Temporal Logics properties are traditionally interpreted in terms of Kripke structures. Clarke has proved that any CTL formula can be expressed in terms of $\neg$, $\vee$, **EX**, **EU** and **EG** [13]. Thus, it is sufficient to be able to handle six cases, depending on whether g is atomic or has one of the following forms: $\neg f_1$, $f_1 \vee f_2$, $\mathbf{EX}f_1$, $\mathbf{E}[f_1 \mathbf{U} f_2]$ or $\mathbf{EG}f_1$. These CTL formulas describe the following properites.

$\mathcal{M}, s \models \neg f_1 \quad \Leftrightarrow \quad \mathcal{M}, s \not\models f_1$

$\mathcal{M}, s \models f_1 \vee f_2 \quad \Leftrightarrow \quad \mathcal{M}, s \models f_1$ or $\mathcal{M}, s \models f_2$

$\mathcal{M}, s \models EXf_1 \quad \Leftrightarrow \quad$ there exists a state t such that $R(s,t)$ and $\mathcal{M}, t \models f_1$

$\mathcal{M}, s \models E[f_1 U f_2] \quad \Leftrightarrow \quad$ there exists an infinite path π starting at s and there exists a $k \geq 0$ such that $\mathcal{M}, s_k \models f_2$ and for all $0 \geq j < k, \mathcal{M}, s_j \models f_1$

$\mathcal{M}, s \models EGf_1 \quad \Leftrightarrow \quad$ there exists an infinite path π starting at s such that for all $i \geq 0, \mathcal{M}, s_i \models f_1$

B. Probe Machine

Probe Machine (PM) is defined as a nine-tuple

$$PM = (X, Y, \sigma_1, \sigma_2, \tau, \lambda, \eta, Q, C)$$

where the nine components denote the data library (X), probe library (Y), data controller (σ_1), probe controller (σ_2), probe operation (τ), computing platform (λ), detector (η), true solution storage (Q), and residue collector (C). The following will introduce four main components: data library, probe library, probe operation, and computing platform.

- The data library X is viewed as a set of n elements, denoted by $X = \{x_1, x_2, ..., x_n\}$. Each data x_i contains a body and p types of data fibers. Data x_i is defined as
$$x_i = \{x_i^1, x_i^2, ..., x_i^p\}$$
- The probe in the probe machine is defined as a tool to find two data and implement some operations (e.g., connective and transitive operations) between them. Formally, let x_i^j and x_t^m be two types of data fibers. The probe between x_i^j and x_t^m denoted as $\tau^{x_i^j x_t^m}$. A connective probe, denoted as $\overline{x_i^j, x_t^m}$, refers to a probe $\tau^{x_i^j x_t^m}$ connecting two target data fibers x_i^j and x_t^m, forming a high-order aggregation.
- A probe operation τ is a process of executing many probe operations simultaneously.
- The computing platform, denoted by λ, is an environment to conduct probe operations τ. It helps probes rapidly find the target data fibers and then conduct probe operations. High cohesiveness, threshold property, and uniqueness are the fundamental functions of the computing platform.
  - High cohesiveness is the rule that high-order aggregation data are given higher priority than low-order aggregation to be executed probe operation.
  - Threshold property limit that the size of two aggregations for probe operation must not exceed the size of the graph itself.
  - Uniqueness is the rule that there is at most one data for each type that an aggregation contains.

III. Model Checking based on Probe Machine

This section is concerned with CTL model checking on the probe machine. We first introduce the procedures of model checking on the probe machine. Construction of the data library and probe library are two main steps. Subsequently, we propose the methods to construct data library and design methods for probe library of **EG** and **EU** respectively. In the end, we analyze the time complexity for the designed approach.

A. Procedures of Model Checking on Probe Machine

Xu solve the graph coloring problems with connective probes that connect two adjacent vertexes of different colors [9]. Inspired by the previous work, we propose a graph search mechanism for model checking. Here, we use a probe to connect two adjacent states in the graph which satisfying the CTL property. Each subpath is considered as data to be probed. In this way, each subpath continuously grows after each probe operation until the initial state is included in the path.

The procedures of model checking on the probe machine are divided into four steps as follows

- Construction of the data library X.
- Establishment of the probe library Y according to the CTL property.
- Implement the probe operation $\tau(X, Y)$ in the computing platform λ, producing a large number of solutions.
- Find the true paths T of all solutions.

Specially, Clarke has proved that any CTL formula can be expressed in terms of $\neg$, $\vee$, **EX**, **EU** and **EG** [13]. Thus, it is sufficient to be able to handle six cases. We usually check $L(s)$ to verify $\mathcal{M}, s \models \neg f_1$, $\mathcal{M}, s \models f_1 \vee f_2$ and traverse the successors of the state s to check $\mathcal{M}, s \models \mathbf{EX}f_1$. These

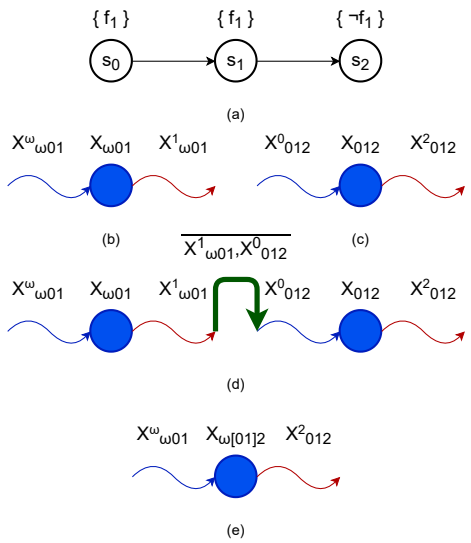

Fig. 1. (a) A simple Kripke Structure. (b) Data $x_{\omega 01}$ with its two data fibers $\underline{x^{\omega}_{\omega 01}, x^{1}_{\omega 01}}$. (c) Data x_{012} with its two data fibers x^0_{012}, x^2_{012}. (d) Probe $\overline{x^1_{\omega 01}, x^0_{012}}$ (e) Two-aggregation data $x_{\omega[01]2}$.

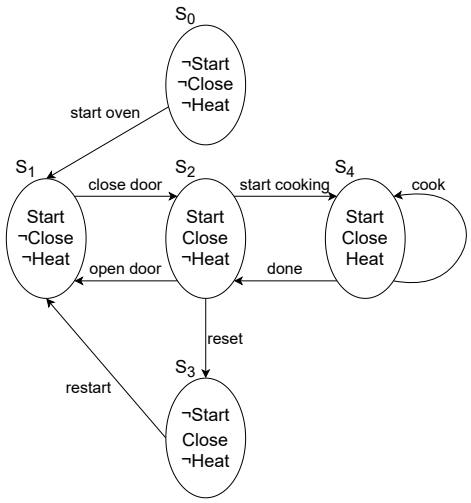

Fig. 2. Microwave oven example.

three verifications can be solved within a short time, so we don't need some particular methods on the probe machine to check them. Therefore, only two algorithms are needed for **EG** and **EU**. And the following will respectively introduce how to build the data library and probe library for $\mathcal{M}, s \models \mathbf{EG}f_1$ and $\mathcal{M}, s \models \mathbf{E}[f_1 \mathbf{U} f_2]$ respectively.

B. Construction of Data Library

Let a graph G mean a Kripke structure, $\mathcal{M} = \{S, I, R, L\}$. We denote $V(G)$ and $E(G)$ as the sets of states and transitions of G respectively. G has a set of nodes $V(G) = \{v_1, \ldots, v_n\}$ and a set of edges $E = \{e_1, \ldots, e_p\}$, as well as $L(v_i) \in \{f_1, \neg f_1\}$ holds for each v_i.

In addition, we denote $Pre(v_i)$ and $Suc(v_i)$ as the sets of predecessor nodes and successor nodes of the vertex v_i. $E(v_i)$ is defined as the set of edges out of the vertex v_i. Let $E^2(v_i)$ be the set of all directed two-paths with internal vertex v_i. Formally

$$E^2(v_i) = \{v_l v_i v_r \triangleq x_{lij} | v_l \in Pre(v_i), v_r \in Suc(v_i)\} \quad (1)$$

Data x_{lij} are defined according to (1). Let's take an example[see Fig. 1(a)], it is a simple Kripke structure with 3 states and 2 transitions. Every state is considered as data with its transiton edge, such as $E^2(v_1) = \{x_{012}\}$ [see Fig. 1(c)].

Based on $E^2(v_i)$, we construct the data library X of the connective probe machine as follows:

$$\begin{aligned} X &= \cup_{i=1}^{n} E^2(v_i) = \\ &= \cup_{i=1}^{n} \{x_{lir} \triangleq v_l v_i v_r | v_l \in Pre(v_i), v_r \in Suc(v_i)\} \end{aligned} \quad (2)$$

where x_{lir} has exactly two types of data fibers, the left one is x^l_{lir} and the right one is x^r_{lir}. Typically, initial states have no predecessor and final states have no successor, so we define ω as empty. The data $x_{\omega 01}$ of the initial state s_0 is shown in Fig. 1(b).

In fact, each data represent a path and a n-aggregation data represents a path of length n. For example, data x_{012} represents a one-length path, $\pi = v_1$ [see Fig. 1(c)], and 0 and 2 respectively represent the possibe node connecting this path. Similarly, a two-aggregation data $x_{\omega[01]2}$ represents a two-length path, $\pi = v_0, v_1$ [see Fig. 1(e)].

C. Probe Library for $\mathbf{EG}f_1$

The logical operator **EG** means that there is a path so that all future states on it are satisfied. Based on this property, we set a rule for the probe library Y of $\mathbf{EG}f_1$ as follows.

Rule 1: Let $\underline{x_{utv}}$ and x_{lir} be two data in X. Then there exists a probe $\underline{x^v_{utv}, x^l_{lir}}$ between them if and only if $l = t$, $v = i$ and $f_1 \in L(v_t)$, $f_1 \in L(v_i)$.

As shown in Fig. 1(d), $f_1 \in L(v_0)$, $f_1 \in L(v_1)$ and the data $\underline{x_{\omega 01}}$ is adjacent to another data x_{012}, so there exists a probe $\underline{x^1_{\omega 01}, x^0_{012}}$ to connect the two data fibers, which forming a two-aggregation data $x_{\omega[01]2}$ and remaining two data fiber $x^{\omega}_{\omega 01}$ and x^2_{012} [see Fig. 1(e)].

By taking the graph in Fig. 2 as an example, the following steps describe the process of checking $\mathcal{M}, s_0 \models \mathbf{EG}\neg Heat$ by our approach.

Step 1: Data library X can be constructed as follows:

$$X = E^2(v_0) \cup E^2(v_1) \cup E^2(v_2) \cup E^2(v_3) \cup E^2(v_4).$$

where, $E^2(v_0) = \{x_{\omega 01}\}$, $E^2(v_1) = \{x_{012}, x_{212}, x_{312}\}$, $E^2(v_2) = \{x_{121}, x_{123}, x_{124}, x_{421}, x_{423}, x_{424}\}$, $E^2(v_3) = \{x_{231}\}$, and $E^2(v_4) = \{x_{242}, x_{244}, x_{442}, x_{444}\}$. Let $\zeta(x)$ represent data fibers of data x. There are total 15 types of data in X, and 30 types of data fibers has been generated as follows:

$\zeta(x_{\omega 01}) = \{x^{\omega}_{\omega 01}, x^{1}_{\omega 01}\}$, $\zeta(x_{012}) = \{x^{0}_{012}, x^{2}_{012}\}$
$\zeta(x_{212}) = \{x^{2}_{212}, x^{2}_{212}\}$, $\zeta(x_{312}) = \{x^{3}_{312}, x^{2}_{312}\}$
$\zeta(x_{121}) = \{x^{1}_{121}, x^{1}_{121}\}$, $\zeta(x_{123}) = \{x^{1}_{123}, x^{3}_{123}\}$
$\zeta(x_{124}) = \{x^{1}_{124}, x^{4}_{124}\}$, $\zeta(x_{421}) = \{x^{4}_{421}, x^{1}_{421}\}$
$\zeta(x_{423}) = \{x^{4}_{423}, x^{3}_{423}\}$, $\zeta(x_{424}) = \{x^{4}_{424}, x^{4}_{424}\}$
$\zeta(x_{231}) = \{x^{2}_{231}, x^{1}_{231}\}$, $\zeta(x_{242}) = \{x^{2}_{242}, x^{2}_{242}\}$
$\zeta(x_{244}) = \{x^{2}_{244}, x^{4}_{244}\}$, $\zeta(x_{442}) = \{x^{4}_{442}, x^{2}_{442}\}$
$\zeta(x_{444}) = \{x^{4}_{444}, x^{4}_{444}\}$,

Step 2: Probe library construction

Based on the data library X, we construct the corresponding probe library Y according to rule 1.

$$
\begin{aligned}
Y_{01} &= \{\overline{x^{1}_{\omega 01},\, x^{0}_{012}}\} \\
Y_{12} &= \{\overline{x^{2}_{012},\, x^{1}_{121}},\, \overline{x^{2}_{012},\, x^{1}_{123}},\, \overline{x^{2}_{012},\, x^{1}_{124}}, \\
&\quad \overline{x^{2}_{212},\, x^{1}_{121}},\, \overline{x^{2}_{212},\, x^{1}_{123}},\, \overline{x^{2}_{212},\, x^{1}_{124}}, \\
&\quad \overline{x^{2}_{312},\, x^{1}_{121}},\, \overline{x^{2}_{312},\, x^{1}_{123}},\, \overline{x^{2}_{312},\, x^{1}_{124}}\} \\
Y_{21} &= \{\overline{x^{1}_{121},\, x^{2}_{212}},\, \overline{x^{1}_{421},\, x^{2}_{212}}\} \\
Y_{23} &= \{\overline{x^{3}_{123},\, x^{2}_{231}},\, \overline{x^{3}_{423},\, x^{2}_{231}}\} \\
Y_{31} &= \{\overline{x^{1}_{231},\, x^{3}_{312}}\} \\
Y_{24} &= Y_{42} = \emptyset
\end{aligned}
$$

Step 3: Probe operations

In each iteration, probes connect the probeable data to form larger data, namely, aggregations. After several rounds, each data containing the initial state is a feasible path satisfying the property, otherwise, the largest data is a counterexample.

Probe operation rules are based on three functions of the computing platform: high cohesiveness, threshold, and uniqueness. They ensure that aggregations grow quickly according to the rules. By taking the oven example, the process of probe operation is as follows.

Data library X and probe library Y have been constructed, and the following Table I showing how to obtain the feasible paths in the computing platform λ. For model checking, the number of iterations is at most equal to $\lceil log_2 n \rceil + 1$, n is the number of vertexes in the graph G. In this example, it takes three steps for probe operations.

TABLE I
SOLUTIONS PROCEDURES

	Computing Platform
0	$x_{\omega 01}$, x_{012}, x_{212}, x_{312} ...
1	$x_{\omega[01]2}$, $x_{0[12]3}$, $x_{1[21]2}$, $x_{1[23]1}$, $x_{2[31]2}$ ...
2	$x_{\omega[0123]1}$, $x_{0[123]1}$, $x_{\omega[012]3}$, $x_{4[231]1}$ ...
3	$x_{\omega[0123]1}$

In the first iteration, the probe $\overline{x^{1}_{\omega 01},\, x^{0}_{012}}$ find the data fiber $x^{1}_{\omega 01}$ of the data $x_{\omega 01}$ and x^{0}_{012} of x_{012}, connecting them and forming a two-aggregation data $\overline{x_{\omega[01]2}}$. And the data $x_{1[24]4}$ isn't formed for lack of probe $\overline{x^{4}_{124}, x^{2}_{244}}$. Many two-aggregations are generated in this round, but some are not shown in the table I.

In the second iteration, some four-aggregations and three-aggregations are generated. For example, the probe $\overline{x^{2}_{012},\, x^{1}_{123}}$ find the data fiber x^{2}_{012} of the data $x_{\omega[01]2}$ and x^{1}_{123} of $x_{1[23]1}$, connecting them and forming a four-aggregation data $x_{\omega[0123]1}$.

In the third iteration, no larger aggregation is produced, so the iteration stops.

It is clear that $x_{\omega[0123]1}$ contians the initial node s_0 and a cycle $\pi = s_1 s_2 s_3 s_1 s_2 s_3 \ldots$, so it can be concluded that $\mathcal{M}, s_0 \models \mathbf{EG}\neg Heat$ and an available path is $\pi = s_0 s_1 s_2 s_3 s_1 s_2 s_3 \ldots$.

For the problem of $\mathbf{EG}f_1$, all true solutions are the aggregations including the initial state s_0 and a cycle, and feasible paths are recorded by aggregations themselves. Otherwise, it means $\mathcal{M}, s_0 \not\models f_1$. Furthermore, the largest aggregation can be the counterexample, since it represents the path closest to true solutions.

D. Probe Library for $\boldsymbol{E}[f_1 \boldsymbol{U} f_2]$

The logic operator **EU** means that there is a path keeping a state before another certain state appears. Based on this property, two rules are set as follows to establish probe library Y of $\mathbf{E}[f_1 \mathbf{U} f_2]$.

Rule 1: Let x_{utv} and x_{lir} be two data in X. Then there exists a probe $\overline{x^{v}_{utv}, x^{l}_{lir}}$ between them if and only if $l = t, v = i$ and $\{\neg f_2, f_1\} \in L(v_t)$, $f_2 \in L(v_i)$.

Rule 2: Let x_{utv} and x_{lir} be two data in X. Then there exists a probe $\overline{x^{v}_{utv}, x^{l}_{lir}}$ between them if and only if $l = t, v = i$ and $\{\neg f_2, f_1\} \in L(v_t)$, $f_1 \in L(v_i)$.

Rule 1 and rule 2 ensure that only one state on the path satisfies f_2 and lies at the endpoint. By taking the graph in Fig. 2 as an example, the following steps describe the process of checking $\mathcal{M}, s_0 \models \mathbf{E}[\neg Heat\ \mathbf{U}\ Close]$ on probe machine.

Based on the two rules, we construct the corresponding probe library Y as follows.

$$
\begin{aligned}
Y_{01} &= \{\overline{x^{1}_{\omega 01},\, x^{0}_{012}}\} \\
Y_{12} &= \{\overline{x^{2}_{012},\, x^{1}_{121}},\, \overline{x^{2}_{012},\, x^{1}_{123}},\, \overline{x^{2}_{012},\, x^{1}_{124}}, \\
&\quad \overline{x^{2}_{212},\, x^{1}_{121}},\, \overline{x^{2}_{212},\, x^{1}_{123}},\, \overline{x^{2}_{212},\, x^{1}_{124}}, \\
&\quad \overline{x^{2}_{312},\, x^{1}_{121}},\, \overline{x^{2}_{312},\, x^{1}_{123}},\, \overline{x^{2}_{312},\, x^{1}_{124}}\} \\
Y_{21} &= Y_{23} = Y_{24} = Y_{31} = Y_{42} = \emptyset
\end{aligned}
$$

And the following Table II showing how probe operation τ is executed to obtain solutions in the computing platform λ. After three iterations, the aggregation $x_{\omega[012]3}$ and $x_{\omega[012]4}$ contian the initial node s_0, it can be concluded that $\mathcal{M}, s_0 \models \mathbf{E}[\neg Heat\ \mathbf{U}\ Close]$ and an available path is $\pi = s_0 s_1 s_2$

E. Complexity Analysis

We analyze the time complexity in terms of three steps proposed in Section III-A.

TABLE II
SOLUTIONS PROCEDURES

	Computing Platform
0	$x_{\omega 01}$, x_{012}, x_{212}, x_{312} ...
1	$x_{\omega[01]2}$, $x_{0[12]3}$, $x_{0[12]4}$ ...
2	$x_{\omega[012]3}$, $x_{\omega[012]4}$, ...
3	$x_{\omega[012]3}$, $x_{\omega[012]4}$

Theorem 1: The time complexity of data library X is at most $O(V \times E^2)$.

For each vertex, outgoing and incoming edges need to be visited. Thus, the time complexity of constructing the data library X is $O(V \times E^2)$.

Theorem 2: The time complexity of probe library Y is at most $O(V \times E)$.

We construct the probe library Y by traversing every vertex with its predecessor nodes and its time complexity is $O(e_1)$, and e_1 is the number of its predecessor. The total time complexity of this process is at most $O(V \times E)$.

For model checking, the number of iterations is at most equal to $\lceil log_2 n \rceil + 1$, n is the number of vertexes in the graph G. Thanks to the underlying parallelism of the probe machine, the processing ability of one probe operation τ is 2^q, q is the number of all possible edges to probe [9]. Therefore, our approach of CTL model checking based on the probe machine can greatly reduce the verification time for systems with large state space.

IV. Prototype Tool

This section is concerned with the framework of our tool CTL2PROBE and simulation of CTL model checking on probe machine. First, we will introduce a model checker called CTL2PROBE based on our approach. And then, some simulations are conducted, which prove the feasibility and efficiency of our approach.

A. CTL2PROBE

To simulate automatic verification of the CTL model checking on the probe machine, we develop a model checker called CTL2PROBE. Fig. 3 shows the framework of CTL2PROBE. It consists of three functional modules: Parsing, Modeling, and Computing. The specific design is as follows:

Parsing: This module mainly parses a JSON file into a model and CTL formula. In this process, the transitions between states are recorded in a HashMap structure and the CTL formula is transformed into a parse tree. It takes a particular format JSON file as input. We define this special format to effectively simplify the parsing process. It is a structured markup language similar to XML that users can quickly obtain or change the contents of elements. As shown in Fig. 4, it is a JSON file that describing the state s_0 of oven example at Section III. The file includes the label, predecessor,

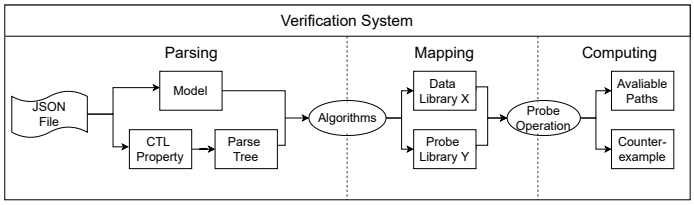

Fig. 3. The Framework Design of Verification System

```
"s0": {
    "lab": ["!Start", "!Close", "!Heat"],
    "pre": [],
    "suc": [1]
},
```

Fig. 4. The JSON file of the oven example

and successor of each state and provides the CTL formula for verification. In the module, CTL formula will be transformed into the formula in terms of $\neg$, $\vee$, **EX**, **EU** and **EG**, and stored in the data structure based on parse tree [see Fig. 5]. It is a bottom-up solving process and the given formula is satisfied when the top node contains the initial node.

Mapping: This module is responsible for mapping elements into the data library and probe library. The rule of mapping is the described in Section III. Data library and probe library are provided for the computing module to verify CTL property.

Computing: This module is used to implement probe operations according to the data library and probe library. The rule of computing is based on three functions of the computing platform: high cohesiveness, threshold, and uniqueness. We realize them according to the following rules.

- High cohesiveness: High-order aggregation data are given higher priority than low-order aggregation to be executed probe operation.
- Threshold: The size of two data for probe operation must not exceed the size of the graph itself.

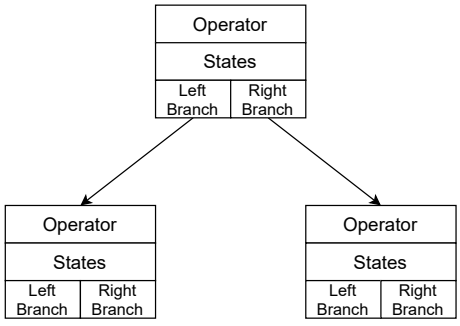

Fig. 5. The data structure of parse tree nodes

```
******* Data Library X *********

w01  012  212  312  121  123

124  421  423  424  231  242

244  442  444  01234  41234
******* Probe Library Y *********

<w01,012>  <121,212>  <421,212>  <231,312>  <012,121>  <212,121>

<312,121>  <012,123>  <212,123>  <312,123>  <012,124>  <212,124>

<312,124>  <123,231>  <423,231>
******* Process of Verification *********
P 1 :  ['w012', 'w01234', '1212', '4212', '2312', '0121', '2121', '31
21', '0123', '2123', '3123', '0124', '2124', '3124', '1231', '4231']
P 2 :  ['w0121', 'w0123', 'w0124', '23121', '23123', '23124', '12312'
, '42312', '01231', 'w01231', '21231', '31231']
P 3 :  []

******* Result *********
Satisfying, and Path is: s0-s1-s2-s3, s0-s1-s2-s1
Modeling Time:  0.0010128021240234375
Verification Time:  0.00031495094299316406
```

Fig. 6. The process of verification

- Uniqueness: There is at most one data for each type that an aggregation contains.

B. Simulation

To verify the effectiveness and efficiency of the proposed algorithm, we have carried out several experiments on CTL2PROBE.

As shown in the Fig. 6, it is the result of verification of $\mathbf{EG}\neg Heat$ in the oven example. It takes a JSON file as the input of the program and the output consists of four parts: data library, probe library, the process of verification, and result. The result part consists of all available paths, modeling time, and verification time. The total time of this verification is $1240 \times 10^{-6}s$, but NuSMV takes $15313 \times 10^{-6}s$ to verify the oven example. It is powerful proof of the efficiency of our tool.

To further prove the efficiency of verifying the model with more states, we have carried out several experiments. We randomly generate multiple graphs with 5, 10, 50 states. Each of the instances runs independently 10 consecutive times to measure the average runtime. The experiment is programmed by Python and executed on a computer system with specifications of Intel Core i-7 at 2.2 GHz CPU and 16 GB RAM under macOS operating system.

The experimental results are shown in Table III. The results show that CTL2PROBE is better than NuSMV in some cases. However, the verification time by CTL2PROBE will increase significantly as the number of nodes increases. Because CTL2PROBE is a tool to simulate probe machine for probe operation and the properties of huge capacity and total parallel are difficult to realize on CTL2PROBE. It is definite that model checking on the probe machine is much faster than traditional model checkers. Our experiment proves the feasibility and efficiency of model checking on the probe machine.

V. Conclusion

In this paper, a novel CTL model checking approach based on the probe machine is proposed, which can solve the limitation for state-space explosion and reduce the verification time for systems with large state space. We design a mapping algorithm to transform the model of Kripke structure into the data library and probe library that can run directly on the probe machine. We develop a model checker called CTL2PROBE to simulate the probe machine, which takes the model as input and obtains all feasible paths or counterexamples. Compared to NuSMV, our approach is more efficient by several comparison experiments.

TABLE III
Comparison of CTL2PROBE with NuSMV on some cases

Nodes	CTL2PROBE(10^{-6})			NuSMV(10^{-6})
	Modeling	Computing	Total Time	
5	426	301	727	13291
10	1127	957	2084	21532
50	17402	1574	18976	30804

VI. Acknowledgement

This work was supported in part by the National Key Research and Development under Project 2019YFA0706404, in part by the NSFC under Project 61972150, and in part by the Shanghai Knowledge Service Platform under Project ZF1213.

References

[1] E. M. Clarke, E. A. Emerson, and A. P. Sistla, "Automatic verification of finite-state concurrent systems using temporal logic specifications," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 8, no. 2, pp. 244–263, 1986.

[2] C. Baier and J.-P. Katoen, *Principles of model checking*. MIT press, 2008.

[3] Y. Kwon and E. Kim, "A design of gpu-based quantitative model checking," in *International Conference on Verification, Model Checking, and Abstract Interpretation*, pp. 441–463, Springer, 2021.

[4] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L.-J. Hwang, "Symbolic model checking: 1020 states and beyond," *Information and computation*, vol. 98, no. 2, pp. 142–170, 1992.

[5] E. M. Clarke, O. Grumberg, K. L. McMillan, and X. Zhao, "Efficient generation of counterexamples and witnesses in symbolic model checking," in *Proceedings of the 32nd annual ACM/IEEE Design Automation Conference*, pp. 427–432, 1995.

[6] P. Godefroid, "Using partial orders to improve automatic verification methods," in *International Conference on Computer Aided Verification*, pp. 176–185, Springer, 1990.

[7] C. Flanagan and P. Godefroid, "Dynamic partial-order reduction for model checking software," *ACM Sigplan Notices*, vol. 40, no. 1, pp. 110–121, 2005.

[8] E. A. Emerson, K. D. Hager, and J. H. Konieczka, "Molecular model checking," *International Journal of Foundations of Computer Science*, vol. 17, no. 04, pp. 733–741, 2006.

[9] J. Xu, "Probe machine," *IEEE transactions on neural networks and learning systems*, vol. 27, no. 7, pp. 1405–1416, 2016.

[10] M. A. Rahman and J. Ma, "Solving symmetric and asymmetric traveling salesman problems through probe machine with local search," in *International Conference on Intelligent Computing*, pp. 1–13, Springer, 2019.

[11] J. Yang, Z. Yin, J. Cui, Q. Zhang, and Z. Tang, "The chinese postman problem based on the probe machine model," in *International Conference on Bio-Inspired Computing: Theories and Applications*, pp. 55–62, Springer, 2018.

[12] S. A. Kripke, "Semantical consideration on modal logic.," *Acta Philosophica Fennica*, vol. 16, 1963.

[13] E. M. Clarke Jr, O. Grumberg, D. Kroening, D. Peled, and H. Veith, *Model checking*. MIT press, 2018.