TELKOMNIKA, Vol.15, No.1, March 2017, pp. 421~429
ISSN: 1693-6930, accredited **A** by DIKTI, Decree No: 58/DIKTI/Kep/2013
DOI: 10.12928/TELKOMNIKA.v15i1.3546

Local Model Checking Algorithm Based on Mu-calculus with Partial Orders

Hua Jiang*[1], Qianli Li[2], Rongde Lin[3]
[1,2]Key Lab of Granular Computing, Minnan Normal University, Zhangzhou 363000, Fujian, China
[3]School of Mathematical Science, Huaqiao University, QuanZhou 362021, Fujian, China
*Corresponding author, e-mail: sg_jh@126.com

Abstract

The propositionalμ-calculus can be divided into two categories, global model checking algorithm and local model checking algorithm. Both of them aim at reducing time complexity and space complexity effectively. This paper analyzes the computing process of alternating fixpoint nested in detail and designs an efficient local model checking algorithm based on the propositional μ-calculus by a group of partial ordered relation, and its time complexity is $O(d^2(dn)^{d/2+2})$ (d is the depth of fixpoint nesting, n is the maximum of number of nodes), space complexity is $O(d(dn)^{d/2})$. As far as we know, up till now, the best local model checking algorithm whose index of time complexity is d. In this paper, the index for time complexity of this algorithm is reduced from d to d/2. It is more efficient than algorithms of previous research.

Keywords: *model checking, propositional mu-calculus, computational complexity, fixpoint, partitioned dependency graph*

1. Introduction

Propositional μ-calculus [1-4] model checking technique is widely used in the design and verification of the finite-control concurrent system. Model checking algorithms can be segmented into two categories. One is global model checking that obtains all the sets of states which satisfy a given logic expression in a finite-control concurrent system. The other is local model checking, which is not always necessary to examine all the states. As we know, the state space explosion problem is the main problem that the propositional μ-calculus model checking algorithm faces with, so it is one of the hot topics to reduce time complexity and space complexity effectively.

For global model checking, according to Tarski Fixpoint theory [5] and the fixpoint operator of formula, it can be computed by iteration. A number of global algorithms have been devised, for global propositionalμ-calculus, Emerson and Lei [6] presented a global algorithm that time complexity of the global algorithm was $O(n^{d+1})$, then Andersen, Cleaveland and Steffen, et al., [7] improved the algorithm in [6], but the time complexity was still $O(n^{d+1})$. In 1994, Long, Browne and Clarke, et al., [10] got a group of partial ordered relation by Tarski fixpoint theory and designed a global algorithm, both time complexity and space complexity were $O(n^{d/2+1})$. In 2010, Hua Jiang [11] got two groups of partial ordered relation by Tarski fixpoint theory and designed a global algorithm, the time complexity of the global algorithm was $O((2n+1)^{d/2+1})$, and the space complexity is $O(dn)$, at present, this is the best study result of global model checking algorithm. Because the global algorithms can not solve some practical problems perfectly, the local model checking was necessary.

Some efficient local methods have been proposed. And the local algorithm of [12-16] were proposed by propositional μ-calculus, but the local algorithm in [17-20] were proposed in other ways. J. F. Jensen et al [17] described a local algorithm for evaluating minimal fixpoint on symbolic dependency graphs that was an extension of dependency graphs in pseudo code and proposed a local algorithm for this framework. However, they did not consider the evaluation of

Received October 16, 2016; Revised December 30, 2016; Accepted January 19, 2017

alternating fixpoints. Though reference [17] improved the complexity of the local algorithm, its efficiency did not achieve the desired results.

Related work can be found in [19] which presented global and local algorithms for computing fixpoint in linear time. In this way, the occurrence of the state exponential explosion problem is delayed, global algorithm is compared with local algorithm in [20], Jiang Hua [21] described an improved algorithm of global model checking for propositional μ-calculus. Modal μ-calculus are also important for studying probabilistic systems, Liu Wangwei, et al., [22] presented a natural and succinct probabilistic extension of μ-calculus, called PμTL, Castro Pablo, et al, [23] presented a probabilistic μ-calculus by using probabilistic quantification as an atomic operation and showed that PCTL and PCTL* can be captured in μ-calculus.

In this paper, we obtain a group of partial order relation by Tarski Fixpoint theory and the fixpoint operator of formula, then we present the bound algorithm which is based on the group of partial order relation. In this way, we can reduce the complexity and improve the computational efficiency. Our main result is a new efficient local algorithm that makes extensive use of monotonicity considerations to reduce the complexity of evaluation for evaluating partitioned dependency graphs [15] fixpoints. And the index for time complexity of this algorithm is reduced from d to $d/2$.

The structure of the rest of this paper is organized as follows. In section 2, the equivalence between semantics of propositionalμ-calculus and Partitioned Dependency Graphs(PDGs) is introduced, and the basic algorithm for evaluating PDG fixpoint is analyzed in detail. Section 3 gives the partial order relation in the evaluating PDG fixpoint firstly, and then presents a new algorithm based on partial orders, shows the time and space complexity of the algorithm is $O(d^2 \cdot (dn)^{d/2+2})$ and $O(d \cdot (dn)^{d/2})$, and finally gives some experimental results. This paper ends with a detailed discussion of some conclusions and directions for future research in section 4.

2. Partitioned Dependency Graphs and Fixpoint Evaluating Algorithm

The syntax of propositional μ-calculus formulas and the semantics under the transition system are refer to [24]. To guarantee the existence of the fixpoints, formulas with positive normal form (PNF) [1] are considered only, where each propositional variable is restricted to a fixpoint operator at most and the operator $\neg$ only acts on the atomic proposition.

2.1. Partitioned Dependency Graphs

Let transition system $M=(S,T,L)$, where S is a non-empty set of states, L is a mapping each atomic proposition to a subset of S , and T maps $\forall\, a \in \{a,b,a_1,a_2,...\}$ to a tuple of state, $T: a \to (S,S)$. For given a PNF fixpoint formula $\mu R.\phi$ or $\nu R.\phi$, the semantics denotes as $\mu R.\phi_{\ M}(\bar{S})$ or $\nu R.\phi_{\ M}(\bar{S})$ respectively, which is the least fixpoint or greatest fixpont of the predicate transformer $\tau: 2^S \to 2^S$ respectively. So the mapping between two subset of states defined by predicate transformer is a dependency, and thus the computation sequences of fixpoint evaluatings is equivalent to a partitioned dependency graphs [15].

Definition 1. A partitioned dependency graph (PDG) is a tuple $(V,E,V_1...V_n,\sigma)$, where V is a set of vertices, $E \subseteq V \times 2^V$ is a set of hyper-edges, $V_1...V_n$ is a finite sequence of subsets of V such that $\{V_1,...,V_n\}$ is a partition of V, and $\sigma:\{V_1,...,V_n\} \to \{\mu,\nu\}$ is a function that assigns μ or ν to each block of the partition [15]. Let $\theta \in \{\mu,\nu\}$. We shall subsequently write $\sigma(x)=\theta$ if $x \in V_i$ and $\sigma(V_i)=\theta$.

G is a PDG, $G=(V,E,V_1...V_n,\sigma)$. Xinxin Liu, et al., [15] regarded G as a nested boolean equation system [13], $\forall x \in V_i, x = \vee_{(x,S)\in E} \wedge_{y \in S} y$. And $\sigma(V_i)$ are nested in $V_1...V_n$, where V_1 and V_n are the outermost block and innermost block respectively.

Example 1. G is a PDG and $G=(V,E,V_1V_2V_3V_4,\sigma)$, where $V=\{x_1,x_2,x_3,x_4,x_5,x_6\}$, $V_1=\{x_1,x_2\}$, $V_2=\{x_3\}$, $V_3=\{x_4,x_5\}$, $V_4=\{x_6\}$, $E=\{(x_1,\{x_3,x_4\}),(x_2,\{x_6\}),(x_2,\{x_5\}),(x_3,\{x_1,x_5\}),(x_4,\{x_1\}),(x_5,\{x_3,x_6\}),(x_6,\{x_1\}),(x_6,\{x_4\})\}$, $\sigma(V_1)=\nu$, $\sigma(V_2)=\mu$, $\sigma(V_3)=\nu$, $\sigma(V_4)=\mu$. Thus, the corresponding nested boolean equation system consists of:

$$\nu:\begin{cases}x_1=x_3\wedge x_4\\ x_2=x_5\vee x_6\end{cases},\ \mu:\{x_3=x_1\wedge x_5,\ \nu:\begin{cases}x_4=x_1\\ x_5=x_3\wedge x_6\end{cases},\ \mu:\{x_6=x_1\vee x_4$$

2.2. Algorithm for PDG fixpoint Evaluatings

In reference [15] a local algorithm for evaluating PDG fixpoint, namely LAFP is proposed, where the search space is constructed as a subset of V which is divided into three blocks, and computes the fixpoints iteratively.

Given a PDG, let b denote the out-to-in sequence $b_1,b_2,...,b_d$, where d ($d \bmod 2=0$) is fixpoint nesting depth. There are n_i nodes in b_i, and the fixpoint types are $\sigma_{2k-1}(V_{2k-1})=\nu$, $\sigma_{2k}(V_{2k})=\mu$, $k=1,2,...$, respectively. So all the sequences of b are as follows:

$$\begin{cases}b_1:V_1=\{x_{10},x_{11},...,x_{1n_0}\}, & \sigma_1(V_1)=\nu\\ b_2:V_2=\{x_{20},x_{21},...,x_{2n_1}\}, & \sigma_2(V_2)=\mu\\ \vdots & \\ b_{d-1}:V_{d-1}=\{x_{(d-1)0},x_{(d-1)1},...,x_{(d-1)n_{(d-1)}}\}, & \sigma_{d-1}(V_{d-1})=\nu\\ b_d:V_d=\{x_{d0},x_{d1},...,x_{dn_d}\}, & \sigma_d(V_d)=\mu\end{cases} \tag{1}$$

Let's divides V_i into three blocks, denoting $V_i=V_i^{'}\cup V_i^{''}\cup V_i^{'''}$ $(1\le i\le d)$, where $V_i^{'}$ saves nodes waiting for computing, $V_i^{''}$ saves nodes which have been identified, $V_i^{'''}$ saves nodes which have not been identified. A assume that the initial value of state of each node of V_i are $True$ or $False$, then $V_1(val)=True$, $V_2(val)=False$, $V_3(val)=True$, $V_4(val)=False$, ..., $V_{d-1}(val)=True$, $V_d(val)=False$ respectively, $V_i(val)$ means the initial value of state of each node.

Let $g_1,g_2,...,g_{d-1},g_d$ be the computation function of the corresponding node of b in PDG, then the iteration formulas is as follows:

$$\begin{aligned}&V_1^{k_1+1}=g_1(V_1^{k_1},V_2^{k_1\omega},...,V_{d-1}^{k_1\omega...\omega},V_d^{k_1\omega...\omega\omega})\\ &V_2^{k_1(k_2+1)}=g_2(V_1^{k_1},V_2^{k_1k_2},...,V_{d-1}^{k_1k_2...\omega},V_d^{k_1k_2..\omega\omega})\\ &\vdots\\ &V_{d-1}^{k_1k_2...(k_{d-1}+1)}=g_{d-1}(V_1^{k_1},V_2^{k_1k_2},...,V_{d-1}^{k_1k_2...k_{d-1}},V_d^{k_1k_2...k_{d-1}\omega})\\ &V_d^{k_1k_2...k_{d-1}(k_d+1)}=g_d(V_1^{k_1},V_2^{k_1k_2},...,V_{d-1}^{k_1k_2...k_{d-1}},V_d^{k_1k_2...k_{d-1}k_d})\end{aligned} \tag{2}$$

The computing process of fixpoint nesting of LAFP is as follows. The computation sequence of nodes of V_1 is $V_1^0,V_1^1,V_1^2,...,V_1^{\omega-1},V_1^{\omega}$. If V_1 reaches the fixpoint with ω, then

$V_1.val = V_1^{\omega}$, $V_1.val$ means the iteration value of the nodes of V_1. When $V_1.val = V_1^{k_1}$, then the computation sequence of V_2 is $V_2^{k_1 0}, V_2^{k_1 1}, V_2^{k_1 2}, ..., V_2^{k_1(\omega-1)}, V_2^{k_1\omega}$. When $V_1.val = V_1^{k_1}$, $V_2.val = V_2^{k_1k_2}$, then the computation sequence of V_3 is $V_2^{k_1k_2 0}, V_2^{k_1k_2 1}, V_2^{k_1k_2 2}, ..., V_2^{k_1k_2(\omega-1)}, V_2^{k_1k_2\omega}$. When $V_1.val = V_1^{k_1}$, $V_2.val = V_2^{k_1k_2}$,…, $V_{d-1}.val = V_{d-1}^{k_1k_2...k_{d-1}}$, then the computation sequence of V_d is $V_d^{k_1k_2...k_{d-1}0}, V_d^{k_1k_2...k_{d-1}1}, V_d^{k_1k_2...k_{d-1}2}, ..., V_d^{k_1k_2...k_{d-1}(\omega-1)}, V_d^{k_1k_2...k_{d-1}\omega}$.

Therefore we can obtain $V_i^{k_1k_2...k_{i-1}k_i} = V_i^{'k_1k_2...k_{i-1}k_i} \cup V_i^{''k_1k_2...k_{i-1}k_i} \cup V_i^{'''k_1k_2...k_{i-1}k_i}$.

Thus, for given a PDG, the nesting computation sequence of Equation (1) descripted as:

$$
\begin{aligned}
& V_d^{00...00}, V_d^{00...01}, V_d^{00...02}, ..., V_d^{00...0\omega}, V_{d-1}^{00...1}, V_d^{00...10}, V_d^{00...11}, V_d^{00...12}, ..., V_d^{00...1\omega}, V_{d-1}^{00...2}, \\
& \vdots \\
& V_d^{00...(\omega-1)0}, V_d^{00...(\omega-1)1}, V_d^{00...(\omega-1)2}, ..., V_d^{00...(\omega-1)\omega}, V_{d-1}^{00...\omega}, ...V_2^{01}, \\
& V_d^{01...00}, V_d^{01...01}, V_d^{01...02}, ..., V_d^{01...0\omega}, V_{d-1}^{01...1}, V_d^{01...10}, V_d^{01...11}, V_d^{01...12}, ..., V_d^{01...1\omega}, V_{d-1}^{01...2}, \\
& \vdots \\
& V_d^{01...(\omega-1)0}, V_d^{01...(\omega-1)1}, V_d^{01...(\omega-1)2}, ..., V_d^{01...(\omega-1)\omega}, V_{d-1}^{01...\omega}, ...V_2^{02}, ..., V_2^{0\omega}, V_1^{1}, \\
& V_d^{10...00}, V_d^{10...01}, V_d^{10...02}, ...V_d^{10...0\omega}, V_{d-1}^{10...1}, V_d^{10...10}, V_d^{10...11}, V_d^{10...12}, ...V_d^{10...1\omega}, V_{d-1}^{10...2}, \\
& \vdots \\
& V_d^{10...(\omega-1)0}, V_d^{10...(\omega-1)1}, V_d^{10...(\omega-1)2}, ..., V_d^{10...(\omega-1)\omega}, V_{d-1}^{10...\omega}, ...V_2^{11}, V_d^{11...00}, V_d^{11...01}, ..., V_d^{11...0\omega}, \\
& V_{d-1}^{11...1}, V_d^{11...10}, V_d^{11...11}, V_d^{11...12}, ..., V_d^{11...1\omega}, V_{d-1}^{11...2}, ..., V_{d-1}^{11...\omega}, ...V_2^{12}, ..., V_2^{1\omega}, V_1^{2}, ..., V_1^{\omega}
\end{aligned}
\quad (3)
$$

3. Local Model Checking Algorithm based on Partial Orders

3.1. Partial Ordering Relation of Computing Node Set

Let $N_{(\sigma,i,r_i)}^{val}$ denotes data structure of computing nodes of V, $r_i(1 \le r_i \le n_i)$ is free variable, $val \in \{True, False\}$, $\sigma \in \{\mu, \nu\}$, i is nesting level. We will superscript relation names with vectors of iteration indices to show various approximations. We will let $k_i(0 \le k_i \le n)$ and $\bar{k}_i$ denote vectors of iteration indices. For example, $V_i^{\bar{k}_i 0}$ denotes $V_i^{k_1k_2...k_i 0}$, $\bar{k}_i 0 = k_1k_2...k_i 0$. If $\bar{k}_i 0 = 00...00$, then $V_i^{\bar{k}_i 0}$ means $V_i^{00...00}$. The notation $Cas(\bar{k}_t)$ means that $\bar{k}_t$ is the closest antecedent sequence. That is to say, if $\bar{k}_t \prec \bar{h}_t$, and $\exists h_i = k_i + 1$, where $1 \le i \le t$, then we have $Cas(\bar{h}_t) = \bar{k}_t$.

Let A and M be node sets which consist of $N_{(\sigma,i,r_i)}^{val}(1 \le i \le d)$, and satisfy both of the following criteria, (1) $|\mathsf{A}| = |\mathsf{M}|$. (2) if $N_{(\sigma,i,r_i)}^{False} \in \mathsf{A}$, then $N_{(\sigma,i,r_i)}^{True} \notin \mathsf{A}$. if $N_{(\sigma,i,r_i)}^{True} \in \mathsf{A}$, then $N_{(\sigma,i,r_i)}^{False} \notin \mathsf{A}$; M is similar. where $N_{(\sigma,i,r_i)}^{val}$ is the data structure of computing nodes and $r_i(1 \le r_i \le n_i)$ is free variable.

Definition 2. $\mathsf{F}(\mathsf{A}, \mathsf{M}) = \mathsf{A} \lhd \mathsf{M}$ is one-way, if A and M satisfy both of the following criteria

(1) $\forall\ N_{(\sigma,i,r_i)}^{False} \in \mathsf{A} \Rightarrow N_{(\sigma,i,r_i)}^{False} \in \mathsf{M} \ \vee\ N_{(\sigma,i,r_i)}^{True} \in \mathsf{M}$.

(2) $\forall\ N_{(\sigma,i,r_i)}^{True} \in \mathsf{A} \Rightarrow N_{(\sigma,i,r_i)}^{True} \in \mathsf{M}$.

Clearly, $\digamma$ satisfies reflexive, antisymmetrical and transitive, that is to say, $\digamma$ is a partial ordering relation of computing node set.

For the iteration formulas (2), when $V_1.val = V_1^{k_1}$, $V_2.val = V_2^{k_1k_2}$,…, $V_{d-1}.val = V_{d-1}^{k_1k_2...k_{d-1}}$, then the computation sequence of V_d is $V_d^{k_1k_2...k_{d-1}0}, V_d^{k_1k_2...k_{d-1}1}, V_d^{k_1k_2...k_{d-1}2}, ..., V_d^{k_1k_2...k_{d-1}(\omega-1)}, V_d^{k_1k_2...k_{d-1}\omega}$;

Because $\sigma_d(V_d) = \mu$, the val of each node of V_d is $False$ or $True$. If val is changed from $False$ to $True$, then storing the corresponding node of val in $V_d^{"}$. If $V_d^{k_1k_2...k_{d-1}0}, V_d^{k_1k_2...k_{d-1}1}, V_d^{k_1k_2...k_{d-1}2}, ..., V_d^{k_1k_2...k_{d-1}(\omega-1)}, V_d^{k_1k_2...k_{d-1}\omega}$ never change, by the Definition 4.2, the sequence satisfies the formulas $\digamma$, that is to say, the sequence is one-way, at the same time, $g_1, g_2, ..., g_{d-1}, g_d$ is monotonous, then:

$$\digamma_d : V_d^{k_1k_2...k_{d-1}0} \lhd V_d^{k_1k_2...k_{d-1}1} \lhd V_d^{k_1k_2...k_{d-1}2} \lhd ... \lhd V_d^{k_1k_2...k_{d-1}(\omega-1)} \lhd V_d^{k_1k_2...k_{d-1}\omega} \quad ;$$

For $\sigma_{d-1}(V_{d-1}) = \nu$, we have:

$$\digamma_{d-1} : V_{d-1}^{k_1k_2...k_{d-2}\omega} \lhd V_{d-1}^{k_1k_2...k_{d-2}(\omega-1)} \lhd V_{d-1}^{k_1k_2...k_{d-2}(\omega-2)} \lhd ... \lhd V_{d-1}^{k_1k_2...k_{d-2}1} \lhd V_{d-1}^{k_1k_2...k_{d-2}0} \quad ; \quad (4)$$

For $\sigma_2(V_2) = \mu$, we have $\digamma_2 : V_2^{k_10} \lhd V_2^{k_11} \lhd V_2^{k_12} \lhd ... \lhd V_2^{k_1(\omega-1)} \lhd V_2^{k_1\omega}$;

For $\sigma_1(V_1) = \nu$, we have $\digamma_1 : V_1^{\omega} \lhd V_1^{(\omega-1)} \lhd V_1^{(\omega-2)} \lhd ... \lhd V_1^1 \lhd V_1^0$.

Definition 3. $k_1k_2...k_t$ and $h_1h_2...h_t$ are non-negative integer sequence, and both of them have t integers. $k_1k_2...k_t$ is antecedent than $h_1h_2...h_t$, if they satisfy both of the following criteria:

(1) Exiting an odd (even) bit j of $k_1k_2...k_t$ and $h_1h_2...h_t$, s.t. $k_j < h_j$, where $1 < j \le t,\ j \bmod 2 = 0$.

(2) $k_m = h_m$, where $1 \le m \le t$, $m \ne j$;

$k_1k_2...k_t$ is antecedent than $h_1h_2...h_t$, denoted $k_1k_2...k_t \prec h_1h_2...h_t$; $k_1k_2...k_t \prec h_1h_2...h_t$ denotes that $V_i^{k_1k_2...k_t}$ has been computed when $V_i^{h_1h_2...h_t}$ is computed.

Definition 4. $k_1k_2...k_t$ is the antecedent sequence of $h_1h_2...h_t$, if they satisfy both of the following criteria:

(1) $k_1k_2...k_t \prec h_1h_2...h_t$. (2) $\exists h_i = k_i + 1$, where $1 \le i \le t$;

$k_1k_2...k_t$ is the closest antecedent sequence of $h_1h_2...h_t$, denoting $Cas(h_1h_2...h_t)$= $k_1k_2...k_t$.

Lemma 1 If $\sigma_i = \nu$, $Cas(h_1h_2...h_t)$=$k_1k_2...k_t$, then $V_i^{h_1h_2...h_i}$ and $V_i^{k_1k_2...k_i}$ satisfy $\digamma$, denoting $V_i^{h_1h_2...h_i} \lhd V_i^{k_1k_2...k_i}$.

Proof. (abbreviated)

Definition 5. $k_1k_2,...,k_t$ is an generalized antecedent sequence of $h_1h_2,...,h_t$, if they satisfy both of the following criteria:

(1) The even sequence of $k_1k_2,...,k_t$ is equal to the even sequence of $h_1h_2,...,h_t$, denoting $es(k_1k_2,...,k_t) = es(h_1h_2,...,h_t)$. (2) The odd sequence of $k_1k_2,...,k_t$ and the odd

sequence of $h_1h_2,...,h_t$ satisfy the lexicographic order, denoting $lo(os(k_1k_2,...,k_t)) \prec lo(os(h_1h_2,...,h_t))$.

$k_1k_2,...,k_t$ is an generalized antecedent sequence of $h_1h_2,...,h_t$, denoting $Gas(\bar{h}_t) = \bar{k}_t$, where $\bar{k}_t$ is $k_1k_2,...,k_t$ and $\bar{h}_t$ is $h_1h_2,...,h_t$.

Lemma 2. If $\sigma_i = \nu$, $Gas(\bar{h}_t) = \bar{k}_t$, then $V_i^{\bar{k}_i}$ and $V_i^{\bar{h}_i}$ satisfy F, denoted $V_i^{\bar{h}_i} \lhd V_i^{\bar{k}_i}$.

Proof. (abbreviated)

3.2. Local Model Checking Algorithm based on Partial Orders

As described above, LAFP presents an efficient local model checking algorithm, however, in the nested process, inner value of fixpoint is affected by outer value of fixpoint. If the value of outer iteration does not change, then the outer value of fixpoint starts computing with the value of inner iteration. When the value of outer iteration is changed, then all the inner value need to update, that is to say, a lot of computing processes is repeated.

For arbitrary sequence $\bar{k}_i$, i mod 2 = 1, by **Lemma** 1 and **Lemma** 2, we only need to start the computing from the antecedent sequence $Cas(\bar{k}_i)$ without affecting the correctness of result. Thus, let $V_i^{\bar{k}_i} = V_i^{Cas(\bar{k}_i)}$ instead of $V_i^{\bar{k}_i} = True$, then the iteration time can be reduced and the computing efficiency can be improved. The Local Model Checking Algorithm based on Partial Orders is as follows:

Algorithm 1 Local Model Checking Algorithm based on Partial Orders

for (*i = 1; i <= d; i ++*) **do**
  $V_i^{'} = V_i$, $V_i^{''} = \varnothing$, $V_i^{'''} = \varnothing$; // initialize
end for
i = d; // begin to compute from the innermost layer
while (*i > 0*) **do**
  if (*i == d*) **then**
    Do
    dequeue a node V_i^{*} from $V_i^{'}$;
    $V_i^{'} = V_i^{'} - V_i^{*}$; //remove V_i^{*}
    $V_i^{\bar{k}_{i-1}(k_i+1)} = g_i(V_1^{\bar{k}_1},...,V_{i-1}^{\bar{k}_{i-1}},V_i^{\bar{k}_{i-1}k_i},V_{i+1}^{\bar{k}_i n_{i+1}})$;
    if (*val of* $V_i^{\bar{k}_{i-1}(k_i+1)}$ *changed*) **then**
      $V_i^{''} = V_i^{''} + V_i^{*}$;
    else $V_i^{''} = V_i^{''} + V_i^{*}$;
    end if
    until $V_i^{'} = \varnothing$;
    i = i -1;
  end if
  if (*i != d*) **then**
    Do
    dequeue a node V_i^{*} from $V_i^{'}$;
    $V_i^{'} = V_i^{'} - V_i^{*}$;
    $V_i^{\bar{k}_{i-1}(k_i+1)} = g_i(V_1^{\bar{k}_1},...,V_{i-1}^{\bar{k}_{i-1}},V_i^{\bar{k}_{i-1}k_i},V_{i+1}^{\bar{k}_i n_{i+1}})$;
    if (*val of* $V_i^{\bar{k}_{i-1}(k_i+1)}$ *changed*) **then**
      $V_i^{''} = V_i^{''} + V_i^{*}$;

for (*t > i && t <= d*) **do** // update the value of all the inner layer
  if (*t % 2 == 0*) **then**
    $V_t^{\bar{k}_t 0} = False$; // the initial value is False
  else if ($os(\bar{k}_i 0) = \bar{0}$) **then**
    $V_t^{\bar{k}_t 0} = True$; // the initial value is True
    else $V_t^{\bar{k}_t 0} = V_t^{Cas(\bar{k}_t)\omega}$; //the initial value is $V_t^{Cas(\bar{k}_t)\omega}$
    end if
  end if
end for
else $V_i^{''} = V_i^{''} + V_i^{*}$;
end if
until $V_i^{'} = \varnothing$;
i = i – 1;
end if
end while

3.3. Time Complexity Analysis

When $i = 1$, according to 3.2, the computation sequence of the corresponding node is $V_1^0, V_1^1, V_1^2, ..., V_1^{\omega}$ in block b_1, n_1 is the total number of computing node of block b_1. The initial value of val is $True$ in each node by $\sigma_1(V_1) = \nu$. When the value of val turns into $False$ from $True$, by the monotonicity of function g_1, the value of the node no longer changes in the whole computing process, so the node is deposited in $V_1^{''}$. The worst case is that the value of val turns into $False$ from $True$ after computing each node of $V_1^{'}$, so the greatest computing times of corresponding node in block b_1 are $|g_1| = 1 + 2 + ... + n_1 \le n_1^2$.

When $i = 2$, $V_1.val = V_1^{k_1}$, the computation sequence of the corresponding node is $V_2^{k_1 0}, V_2^{k_1 1}, V_2^{k_1 2}, ..., V_2^{k_1 \omega}$ in block b_2, the computing times of the corresponding node are $1 + 2 + 3 + ... + n_2$ in block b_2, the number of different values of $V_1.val$ is n_1, so the greatest computing times of corresponding node in block b_2 are $|g_2| = n_1(1 + 2 + 3 + ... + n_2) \le n_1 \cdot n_2^2$.

When $i = 3$, according to Algorithm 1, if $k_1 = 0, k_3 = 0$, V_3 starts to compute from $V_3^{'}$. In this case, the times are n_2 at most. The changing times of corresponding node value are $n_2 \cdot n_3$ in block b_3, and the computing times are not more than $n_2 n_3^2$. When $k_1 \ne 0$, $k_3 = 0$, V_3 starts to compute from $V_3^{x_1 x_2 x_3}$. In this case, the times are $n_1 n_2$ at most, when it reaches the fixpoint, the computing times are $n_1 n_2 n_3$ at most, so the greatest computing times of corresponding node in block b_3 are $|g_3| \le n_2 \cdot n_3^2 + n_1 \cdot n_2 \cdot n_3$.

Summarily, when $i \ge 2$, $|g_{2i}| \le n_2 n_4 n_6 ... n_{2i-2} n_{2i-1} n_{2i}^2$,

$|g_{2i+1}| \le n_2 \cdot n_4 \cdot n_6 \cdot ... \cdot n_{2i} \cdot n_{2i+1}^2 + n_2 \cdot n_4 \cdot n_6 \cdot ... \cdot n_{2i-1} \cdot n_{2i} \cdot n_{2i+1}$.

Thus, we have $\sum_{i=1}^{d} |g_i| = |g_1| + |g_2| + ... |g_d|$

$$\le n_1^2 + n_1 \cdot n_2^2 + (n_2 \cdot n_3^2 + n_1 \cdot n_2 \cdot n_3) + n_2 \cdot n_3 \cdot n_4^2 + (n_2 \cdot n_4 \cdot n_5^2 + n_2 \cdot n_3 \cdot n_4 \cdot n_5) + ... +$$

$$(n_2 \cdot n_4 \cdot n_6 \cdot .... \cdot n_{d-2} \cdot n_{d-1}^2 + n_2 \cdot n_4 \cdot n_6 \cdot .... \cdot n_{d-3} \cdot n_{d-2} \cdot n_{d-1}) + n_2 n_4 n_6 ... n_{d-2} n_{d-1} n_d^2$$
$$< n_2 \cdot n_4 \cdot n_6 \cdot .... \cdot n_{d-2} \cdot |V|^2 < (2 \cdot (n \cdot d/2)/d)^{d/2} \cdot |V|^2 = O(d^2 \cdot n^{d/2+2}).$$

Assume the alternative nesting depth $d \bmod 2 = 0$, through the analysis of the above, then the time complexity analysis Algorithm 1 is $O(d^2 \cdot n^{d/2+2})$.

3.4. Space Complexity Analysis

By Algorithm 1, if $\sigma_i(V_i) = \nu$, $V_i^{x_1 x_2 ...(x_{i-2}+1) x_{i-1} 0} = V_i^{x_1 x_2 ... x_{i-2} x_{i-1} \omega}$, then save intermediate results, $V_i^{'}$ and $V_i^{''}$ $(1 \le i \le d)$ account for $2d$ storage units. When $i = 3$, it accounts for $2n_2$ storage units. When $i = 5$, it accounts for $2n_2 \cdot n_4$ storage units. When $i = d$, it accounts for $2n_2 \cdot n_4 \cdot n_6 \cdot ... \cdot n_d$ storage units, therefore, the total numbers of storage units in Algorithm 1 are:

$$2d + 2n_2 + 2n_2 \cdot n_4 + ... + 2n_2 \cdot n_4 \cdot n_6 \cdot .... \cdot n_d = 2(d + n_2 + n_2 \cdot n_4 + ... + n_2 \cdot n_4 \cdot n_6 \cdot .... \cdot n_d)$$
$$< 2(d + |V|^{d/2} + |V|^{d/2} + ... + |V|^{d/2}) = 2(d + d/2(|V|^{d/2})) = O(d \cdot (d \cdot n)^{d/2})$$

3.5. Comparison of Time Complexity

According to Algorithm 1, we assume that the number of node of each layer is 30, then we can obtain the time of iterative computation of all functions by computing. When the alternation depth d takes a different value, the number of iteration is as Table 1. Table 1 shows that our algorithm is more efficient.

Table 1. Times of Fuction Iterative Computing

d	Algorithm 1	LAFP [15]
1	$9.61*10^2$	$9.61*10^2$
2	$3.07*10^4$	$3.07*10^4$
3	$9.54*10^5$	$9.54*10^5$
4	$2.80*10^6$	$2.97*10^7$
5	$6.01*10^7$	$9.15*10^8$
6	$1.75*10^8$	$2.84*10^{10}$
7	$6.62*10^9$	$8.81*10^{11}$
8	$1.21*10^{10}$	$2.74*10^{13}$

4. Conclusion

In this paper, we present a new efficient algorithm for evaluating PDG fixpoints. As we know, [26] presented a local model checker forμ-calculus, as a tableau system, but it did not analyze the computational complexity. Then [15] presented a new local algorithm for evaluating PDG fixpoints, and time complexity of the LAFP algorithm was exponential relationship with nesting depth. After a detailed analysis, we present a new algorithm by[11]. And our algorithm takes about $d^2 \cdot n^{d/2+2}$ steps. Clearly, the time required by our algori thm is only about the square root of the time required by LAFP algorithm. Furthermore, when $d \bmod 2 = 1$, we only need to design the algorithm in the same way as $d \bmod 2 = 0$. The nested bound algorithm reduces repetitive computation and improves the computational efficiency. The research in this paper is very important to theoretical research and practical application [25, 27], it can improve the efficiency for verifying hardware and software designs.

As we know, two groups of partial ordered relation were presented by Tarski fixpoint theory, our next work is to design a local algorithm by obtaining two groups of partial ordered relation and improve the space complexity.

Acknowledgements

This paper is supported by the National Natural Science Foundation of China under Grant No.61472406, the Natural Science Foundation of Fujian Province under Grant No.2015J01269 and No.2016J01304 and the Talent Introduction Foundation of Minnan Normal University.

References

[1] D Kozen. *Results on the propositional μ-calculus*. LNCS 140: Proc of the 9th Colloquium on Automata, Languages and Programming. Springer. 1982: 348-359.
[2] JW de Bakker. Mathematical theory of program correctness. Prentice-Hall, Inc. 1980.
[3] D Park. Fixpoint induction and proof of program semantics. In: B Meltzer, D Michie. *Editors*. Mach. Int, Edinburgh Univ. 1970: 59-78.
[4] C Stirling. Modal and temporal logics for processes. Springer. 1996: 149-237.
[5] A Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics.* 1955; 5(2): 285-309.
[6] EA Emerson, CL Lei. *Efficient model checking in fragments of the propositional mu-calculus*. Proc. 1st LICS. 1986.
[7] HR Andersen. *Model checking and boolean graphs*. ESOP 92. Springer. 1992: 1-19.
[8] R Cleaveland, M Klein, B Steen. *Faster model checking for the modal mu-calculus*. CAV 92, LNCS. 1992; 663: 410-422.
[9] R Cleaveland. Tableau-based model checking in the prepositional mucalculus. *Acta Informatica*. 1990; 27(8): 725-747.
[10] DE Long, A Browne, EM Clarke, et al. An improved algorithm for the evaluation of fixpoint expressions. *Computer Aided Verification.* 1994: 338-350.
[11] H Jiang. Efficient global model-checking for propositional *μ*-calculus. *Journal of Computer Research and Development*. 2010; 47(8): 1424-1433.
[12] HR Andersen. Model checking and boolean graphs. *Theoretical Computer Science*. 1994; 126(1).
[13] B Vergauwen, J Lewi. *Efficient local correctness checking for single and alternating boolean equation systems*. Proceedings of ICALP 94. Springer. 1994: 304-315.
[14] GS Bhat, R Cleaveland. *Efficient model checking for fragments of the modal μ-calculus*. Proceedings of the Second International Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 96). Springer. 1996: 107-126.
[15] XX Liu, CR Ramakrishnan, SA Smolka. Fully local and efficient evaluation of alternating fixed points. *Tools and Algorithms for the Construction and Analysis of Systems*. 1998: 5-19.
[16] R Mateescu. Local model-checking of modal mu-calculus on acyclic labeled transition systems. *Tools and Algorithm for the Construction and Analysis of Systems.* 2002: 281-295.
[17] JF Jensen, LK Østergaard. Local model checking of weighted CTL. 2012.
[18] D Latella, M Loreti, M Massink. On-the-fly fast mean-field modelchecking. Trustworthy Global Computing. Springer International Publishing, LNCS. 2014: 297-314.
[19] R Guerraoui, M Yabandeh. Local model checking. 2011.
[20] JF Jensen, KG Larsen, J Srba, et al. Local model checking of weighted CTL with upper-bound constraints. *Model Checking Software*. 2013: 178-195.
[21] Hua Jiang, Jianqing Xi. Improved algorithm of golbal model-checking for propositional mu-calculus. *Applied Mechanics and Materials*. 2013; 263: 2314-2319.
[22] Liu Wanwei, et al. A Simple Probabilistic Extension of Modal Mu-calculus. 2015.
[23] Castro Pablo, Cecilia Kilmurray, Nir Piterman. Tractable Probabilistic μ-Calculus that Expresses Probabilistic Temporal Logics. 2015.
[24] EM Clarke, O Grumberg, D Peled. Model checking. MIT Press. 1999.
[25] N Piterman, MY Vardi. Global model-checking of infinite-state systems. *Computer Aided Verification.* 2004: 387-400.
[26] C Stirling, D Walker. Local model checking in the modal mu-calculus. *Theoretical Computer Science.* 1991; 89(1): 161-177.
[27] T Schuele, K Schneider. Global vs. local model checking of infinite state systems. *MBMV*. 2004: 54-64.