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

Hua Jiang1, Qianli Li2, Rongde Lin3

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 \(O(d^{(d^n)^{d/2}})\) (d is the depth of fixpoint nesting, \(n\) is the maximum of number of nodes), space complexity is \(O(d(d^n)^{d/2})\). 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 \(\mu\)-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-space 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 \(\mu\)-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...
The structure of the rest of this paper is organized as follows. In section 2, the equivalence between semantics of propositional $\mu$-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 $O(d^2 \cdot (dh)^{d/2+2})$ and $O(d \cdot (dh)^{d/2})$, 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 $\mu$-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 $\sqsubseteq$ 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 $\forall a \in \{a, b, a_1, a_2, \ldots\}$ to a tuple of state, $T : a \rightarrow (S, S)$. For given a PNF fixpoint formula $\mu R\phi$ or $\nu R\phi$, the semantics denotes as $\mu R\phi_M (S)$ or $\nu R\phi_M (S)$ respectively, which is the least fixpoint or greatest fixpoint of the predicate transformer $\tau: 2^S \rightarrow 2^S$ respectively. So the mapping between two subset of states defined by predicate transformer is a dependency, and thus the computation sequences of fixpoint evaluating is equivalent to a partitioned dependency graphs [15].

Definition 1. A partitioned dependency graph (PDG) is a tuple $(V, E, V_1 \ldots V_n, \sigma)$, where $V$ is a set of vertices, $E \subseteq V \times 2^V$ is a set of hyper-edges, $V_1 \ldots V_n$ is a finite sequence of subsets of $V$ such that $\{V_1, \ldots, V_n\}$ is a partition of $V$, and $\sigma : \{V_1, \ldots, V_n\} \rightarrow \{\mu, \nu\}$ is a function that assigns $\mu$ or $\nu$ 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, $G = (V, E, V_1 \ldots V_n, \sigma)$. Xinxin Liu, et al., [15] regarded $G$ as a nested boolean equation system [13], $\forall x \in V_i, x = \bigvee_{(x, S) \in E} \land_{y \in S} y$. And $\sigma(V_i)$ are nested in $V_1 \ldots V_n$, where $V_1$ and $V_n$ are the outermost block and innermost block respectively.
Example 1. \( G \) is a PDG and \( G = (V,E,V_2,V_3,V_4,\sigma) \), where \( V = \{x_1,x_2,x_3,x_4,x_5,x_6\} \), \( V_1 = \{x_1,x_2\} \), \( V_2 = \{x_3\} \), \( V_3 = \{x_4,x_5\} \), \( V_4 = \{x_6\} \), 
\[ E = \{(x_1,x_2),(x_2,x_3),(x_3,x_4),(x_1,x_4),(x_2,x_5),(x_3,x_6)\} \]
\( \sigma(V_1) = \nu \), \( \sigma(V_2) = \mu \), \( \sigma(V_3) = \nu \), \( \sigma(V_4) = \mu \). Thus, the corresponding nested boolean equation system consists of:

\[
\begin{align*}
\nu: \begin{cases} 
x_1 &= x_3 \land x_4 \\
x_2 &= x_3 \lor x_6
\end{cases} \\
\mu: \begin{cases} 
x_3 &= x_1 \land x_3 \\
x_5 &= x_3 \land x_6
\end{cases}
\]

2.2. Algorithm for PDG fixpoint Evaluating

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 \( d \) \((d \mod 2 = 0)\) 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:

\[
\begin{align*}
\{b_1: V_1 &= \{x_{i_0},x_{i_1},...,x_{i_{n_1}}\}, & \sigma_1(V_1) = \nu \\
& b_2: V_2 = \{x_{20},x_{21},...,x_{2n_2}\}, & \sigma_2(V_2) = \mu \\
& \vdots \\
& b_{d-1}: V_{d-1} = \{x_{(d-1)0},x_{(d-1)1},...,x_{(d-1)n_{(d-1)}}\}, & \sigma_{d-1}(V_{d-1}) = \nu \\
& b_d: V_d = \{x_{d0},x_{d1},...,x_{dn_d}\}, & \sigma_d(V_d) = \mu 
\end{align*}
\]

Let's divides \( V_i \) into three blocks, denoting \( V'_i = V_i \cup V'_i \cup V''_i \) \((1 \leq i \leq d)\), 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 \( V'_i(val) = True \), \( V''_i(val) = False \), \( V'_i(val) = True \), \( V''_i(val) = False \), ..., \( V_{d-1}(val) = True \), \( V_d(val) = False \) 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:

\[
\begin{align*}
V_{1k_1+1} &= g_1(V_{1k_1},V_{2k_2},...,V_{d-1k_{d-1}},V_{dk_d}) \\
V_{2k_1+1} &= g_2(V_{1k_1},V_{2k_2},...,V_{d-1k_{d-1}},V_{dk_d}) \\
& \vdots \\
V_{d-1k_{d-1}+1} &= g_{d-1}(V_{1k_1},V_{2k_2},...,V_{d-1k_{d-1}},V_{dk_d}) \\
V_{dk_{d-1}+1} &= g_d(V_{1k_1},V_{2k_2},...,V_{d-1k_{d-1}},V_{dk_d})
\end{align*}
\]

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 \( \omega \), then
\( V_i.val = V_1^0 \), \( V_i.val \) means the iteration value of the nodes of \( V_1 \). When \( V_i.val = V_1^k \), then the computation sequence of \( V_2 \) is \( V_2^{k_0}, V_2^{k_1}, V_2^{k_2}, \ldots, V_2^{k_{(a-1)}}, V_2^{k_0} \). When \( V_i.val = V_1^k \), \( V_2.val = V_2^{k_2} \), then the computation sequence of \( V_3 \) is \( V_2^{k_0}, V_2^{k_1}, V_2^{k_2}, \ldots, V_2^{k_{(a-1)}}, V_2^{k_0} \).

When \( V_i.val = V_1^k, V_2.val = V_2^{k_2} \), ..., \( V_{d-1}.val = V_{d-1}^{k_{d-1}} \), then the computation sequence of \( V_d \) is \( V_{d}^{k_{d-1} - k_{d-2}} V_{d-1}^{k_{d-1}}, V_{d}^{k_{d-2} - k_{d-3}}, \ldots, V_{d}^{k_{2} - k_{1}} \). Therefore we can obtain \( V_{d}^{k_{d-1} - k_{d-2}} V_{d-1}^{k_{d-1}} \cup V_{d}^{k_{d-2} - k_{d-3}} V_{d-2}^{k_{d-2}} \cup \cdots \cup V_{d}^{k_{2} - k_{1}} V_{d-1}^{k_{2}} \).

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

\[
\begin{align*}
V_{d0}, V_{d1}, V_{d2}, \ldots, V_{d0(0, 0)}, V_{d1(0, 0)}, V_{d2(0, 0)}, \ldots, V_{d0(0, 1)}, V_{d1(0, 1)}, V_{d2(0, 1)}, \ldots, V_{d0(1, 0)}, V_{d1(1, 0)}, V_{d2(1, 0)}, \ldots, V_{d0(1, 1)}, V_{d1(1, 1)}, V_{d2(1, 1)}, \ldots, V_{d0(1, 2)}, V_{d1(1, 2)}, V_{d2(1, 2)}, \ldots, V_{d0(1, 3)}, V_{d1(1, 3)}, V_{d2(1, 3)}, \ldots,
\end{align*}
\]

3. Local Model Checking Algorithm based on Partial Orders
3.1. Partial Ordering Relation of Computing Node Set

Let \( N_{\text{val}}^{(\sigma, i, r)} \) denotes data structure of computing nodes of \( V \), \( r_i (1 \leq r_i \leq n_i) \) is free variable, \( \text{val} \in \{\text{True, False} \}, \sigma \in \{\mu, \nu \} \), \( i \) is nesting level. We will superscript relation names with vectors of iteration indices to show various approximations. We will let \( k_i (0 \leq k_i \leq n) \) and \( \vec{k}_i \) denote vectors of iteration indices. For example, \( V_i^{E_0} \) denotes \( V_i^{k_0 - k_0} \), \( \vec{k}_0 = k_0 \). If \( \vec{k}_i 0 = 00 \ldots 00 \), then \( V_i^{E_0} \) means \( V_0^{00 \ldots 00} \). The notation \( \text{Cas}(\vec{k}_i) \) means that \( \vec{k}_i \) is the closest antecedent sequence. That is to say, \( \vec{k}_i \prec \vec{h}_i \), and \( \exists h_i = k_i + 1 \), where \( 1 \leq i \leq t \), then we have \( \text{Cas}(\vec{h}_i) = \vec{k}_i \).

Let \( A \) and \( M \) be node sets which consist of \( N_{\text{val}}^{(\sigma, i, r)} (1 \leq i \leq d) \), and satisfy both of the following criteria, (1) \( |A| = |M| \), (2) if \( N_{\text{False}}^{(\sigma, i, r)} \in A \), then \( N_{\text{True}}^{(\sigma, i, r)} \not\in A \), if \( N_{\text{True}}^{(\sigma, i, r)} \in A \), then \( N_{\text{False}}^{(\sigma, i, r)} \not\in A ; M \) is similar. where \( N_{\text{val}}^{(\sigma, i, r)} \) is the data structure of computing nodes and \( r_i (1 \leq r_i \leq n_i) \) is free variable.

Definition 2. \( F(A, M) = A \triangleq M \) is one-way, if \( A \) and \( M \) satisfy both of the following criteria

1. \( \forall N_{\text{False}}^{(\sigma, i, r)} \in A \Rightarrow N_{\text{False}}^{(\sigma, i, r)} \in M \lor N_{\text{True}}^{(\sigma, i, r)} \in M \).
2. \( \forall N_{\text{True}}^{(\sigma, i, r)} \in A \Rightarrow N_{\text{True}}^{(\sigma, i, r)} \in M \).
Clearly, $F$ satisfies reflexive, antisymmetrical and transitive, that is to say, $F$ is a partial ordering relation of computing node set.

For the iteration formulas (2), when $V_i.val = V_i^{h_1}, V_2.val = V_2^{k_2}, \ldots, V_{d-1}.val = V_{d-1}^{k_2},$ then the computation sequence of $V_d$ is $V_d^{k_2}, V_d^{k_2}, V_d^{k_2}, \ldots, V_d^{k_2}, V_d^{k_2}.$

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 $V_d^{k_2}, V_d^{k_2}, V_d^{k_2}, \ldots, V_d^{k_2}, V_d^{k_2}$ never change, by the Definition 4.2, the sequence satisfies the formulas $F$, that is to say, the sequence is one-way, at the same time, $g_1, g_2, \ldots, g_{d-1}, g_d$ is monotonous, then:

$$F_d : V_d^{k_2}, V_d^{k_2}, V_d^{k_2}, \ldots, V_d^{k_2}, V_d^{k_2}.$$ 

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

$$F_{d-1} : V_{d-1}^{k_2}, V_{d-1}^{k_2}, V_{d-1}^{k_2}, \ldots, V_{d-1}^{k_2}, V_{d-1}^{k_2}.$$ 

For $\sigma_i(V_i) = \nu$, we have:

$$F : V_i^{h_1}, V_i^{h_2}, \ldots, V_i^{h_t}.$$ 

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

1. Exiting an odd (even) bit $j$ of $k_1k_2...k_i$ and $h_1h_2...h_i$, s.t. $j < h_j$, where $1 < j < t, j \ mod \ 2 = 0$.

2. $k_m = h_m$, where $1 \leq m \leq t, m \neq j$;

$k_1k_2...k_i$ is antecedent than $h_1h_2...h_i$, denoted $k_1k_2...k_i < h_1h_2...h_i$; $k_1k_2...k_i < h_1h_2...h_i$ denotes that $V_i^{k_1k_2...k_i}$ has been computed when $V_i^{h_1h_2...h_i}$ is computed.

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

1. Exiting an odd (even) bit $j$ of $k_1k_2...k_i$ and $h_1h_2...h_i$, s.t. $k_j < h_j$, where $1 < j < t$;

2. $k_1k_2...k_i$ is the closest antecedent sequence of $h_1h_2...h_i$, denoting $Cas(h_1h_2...h_i) = k_1k_2...k_i$.

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

Proof. (abbreviated)

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

1. The even sequence of $k_1k_2,...,k_i$ is equal to the even sequence of $h_1h_2,...,h_i$, denoting $es(k_1k_2,...,k_i) = es(h_1h_2,...,h_i)$.

2. The odd sequence of $k_1k_2,...,k_i$ and the odd
sequence of $h_1h_2,...,h_i$ satisfy the lexicographic order, denoting $lo(os(k_1k_2,...,k_i)) < lo(os(h_1h_2,...,h_i))$.

$k_1k_2,...,k_i$ is a generalized antecedent sequence of $h_1h_2,...,h_i$, denoting $Gas(\bar{h}_i) = \bar{k}_i$, where $\bar{k}_i = k_1k_2,...,k_i$ and $\bar{h}_i = h_1h_2,...,h_i$.

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

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

1. for ($i = 1$; $i < d$; $i++$) do
2. \hspace{1em} $V_i' = V_i$, $V_i' = \emptyset$, $V_i'' = \emptyset$; // initialize
3. end for
4. $i = d$; // begin to compute from the innermost layer
5. while ($i > 0$) do
6. \hspace{1em} if ($i == d$) then
7. \hspace{2em} Do
8. \hspace{3em} dequeue a node $V_i^*$ from $V_i'$;
9. \hspace{3em} $V_i' = V_i' - V_i^*$; //remove $V_i^*$
10. \hspace{3em} $V_i^{\bar{k}_i,(k+1)} = g_i(V_i^{\bar{k}_i},...,V_i^{\bar{k}_{i-1}},V_i^{\bar{k}_i,k_i},V_i^{\bar{k}_{i+1}})$;
11. \hspace{3em} if (val of $V_i^{\bar{k}_i,(k+1)}$ changed) then
12. \hspace{4em} $V_i' = V_i' + V_i^*$;
13. \hspace{4em} else $V_i' = V_i' + V_i^*$;
14. \hspace{3em} end if
15. \hspace{3em} until $V_i' = \emptyset$;
16. \hspace{3em} $i = i - 1$;
17. \hspace{3em} end if
18. \hspace{3em} if ($i != d$) then
19. \hspace{4em} Do
20. \hspace{5em} dequeue a node $V_i^*$ from $V_i'$;
21. \hspace{5em} $V_i' = V_i' - V_i^*$;
22. \hspace{5em} $V_i^{\bar{k}_i,(k+1)} = g_i(V_i^{\bar{k}_i},...,V_i^{\bar{k}_{i-1}},V_i^{\bar{k}_i,k_i},V_i^{\bar{k}_{i+1}})$;
23. \hspace{5em} if (val of $V_i^{\bar{k}_i,(k+1)}$ changed) then
24. \hspace{6em} $V_i' = V_i' + V_i^*$;
25. \hspace{4em} end if
26. \hspace{4em} end if
27. end if
28. end while
29. $V_i' = V_i'$;
3.3. Time Complexity Analysis

When \( i = 1 \), according to 3.2, the computation sequence of the corresponding node is \( V^0_1, V^1_1, V^2_1, \ldots, V^\omega_1 \) in block \( b_1 \). \( n_1 \) is the total number of computing node of block \( b_1 \). The initial value of \( \text{val} \) is \( \text{True} \) in each node by \( \sigma_i(V^i) = \nu \). When the value of \( \text{val} \) turns into \( \text{False} \) from \( \text{True} \), 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 \( V'_i \). The worst case is that the value of \( \text{val} \) turns into \( \text{False} \) from \( \text{True} \) after computing each node of \( V'_1 \), so the greatest computing times of corresponding node in block \( b_1 \) are \( |g_1| = 1 + 2 + \ldots + n_1 \leq n_1^2 \).

When \( i = 2 \), \( V_i.\text{val} = V^k_i \), the computation sequence of the corresponding node is \( V^k_2, V^k_1, V^k_2, \ldots, V^k_\omega \) in block \( b_2 \), the computing times of the corresponding node are \( 1 + 2 + 3 + \ldots + n_2 \) in block \( b_2 \), the number of different values of \( V_i.\text{val} \) is \( n_1 \), so the greatest computing times of corresponding node in block \( b_2 \) are \( |g_2| = n_1(1 + 2 + 3 + \ldots + n_2) \leq n_1 n_2^2 \).

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 \( b_1 \), and the computing times are not more than \( n_2 n_3^2 \). When \( k_3 \neq 0 \), \( k_3 = 0 \), \( V_3 \) starts to compute from \( V^k_3, V^k_3, \ldots, V^k_\omega \). In this case, the times are \( n_1 n_2 \) at most, when it reaches the fixpoint, the computing times are \( n_1 n_2 \cdot n_3 \) at most, so the greatest computing times of corresponding node in block \( b_3 \) are \( |g_3| \leq n_2 \cdot n_3^2 + n_1 \cdot n_2 \cdot n_3 \).

Summarily, when \( i \geq 2 \), \( |g_2| \leq n_2 n_4 n_6 \ldots n_{2i-2} n_2i-1 n_2i, \)
\( |g_{2i+1}| \leq n_2 \cdot n_4 \cdot n_6 \ldots n_{2i} n_{2i+2} + n_2 \cdot n_4 \cdot n_6 \ldots n_{2i-1} \cdot n_2i \cdot n_{2i+1} \).

Thus, we have \( \sum_{i=1}^{d} |g_i| \geq |g_1| + |g_2| + \ldots + |g_d| \)
\( \leq n_2 + n_1 \cdot n_2^2 + (n_2 \cdot n_3^2 + n_1 \cdot n_2 \cdot n_3) + n_2 \cdot n_3 \cdot n_3^2 + (n_2 \cdot n_4 \cdot n_3^2 + n_2 \cdot n_3 \cdot n_4 \cdot n_3) + \ldots + \)
\[(n_2 \cdot n_4 \cdot n_6 \cdots n_{d-2} \cdot n_{d-1} + n_2 \cdot n_4 \cdot n_6 \cdots n_{d-3} \cdot n_{d-2} \cdot n_{d-1} + n_2 n_4 n_6 \cdots n_{d-2} n_{d-1} n_d) < n_2 \cdot n_4 \cdot n_6 \cdots n_{d-2} \mid V \mid < (2 \cdot (n \cdot d / 2) / d)^{d/2} \cdot \mid V \mid = O(d^2 \cdot n^{d/2+2}).\]

Assume the alternative nesting depth \(d \mod 2 = 0\), 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 \(\sigma_i(V) = V_i, V_{i,1,2,\ldots,i,2,0} = V_i,\ldots,i,2,0\), then save intermediate results, \(V_i\) and \(V'_i\) \((1 \leq i \leq d)\) 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 \(2n_2 \cdot n_4 \cdot n_6 \cdots n_d\) storage units, therefore, the total numbers of storage units in Algorithm 1 are:

\[2d + 2n_2 + 2n_2 \cdot n_4 + \cdots + 2n_2 \cdot n_4 \cdot n_6 \cdots n_d = 2(d + n_2 + n_2 \cdot n_4 + \cdots + n_2 \cdot n_4 \cdot n_6 \cdots n_d) < 2(d + \mid V \mid^{d/2} + \mid V \mid^{d/2} + \cdots + \mid V \mid^{d/2}) = O(d \cdot (d \cdot n)^{d/2})\]

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>
<thead>
<tr>
<th>d</th>
<th>Algorithm 1</th>
<th>LAFP [15]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>9.61*10^2</td>
<td>9.61*10^2</td>
</tr>
<tr>
<td>2</td>
<td>3.07*10^4</td>
<td>3.07*10^4</td>
</tr>
<tr>
<td>3</td>
<td>9.54*10^6</td>
<td>9.54*10^6</td>
</tr>
<tr>
<td>4</td>
<td>2.80*10^8</td>
<td>2.97*10^8</td>
</tr>
<tr>
<td>5</td>
<td>6.01*10^10</td>
<td>9.15*10^10</td>
</tr>
<tr>
<td>6</td>
<td>1.75*10^12</td>
<td>2.84*10^12</td>
</tr>
<tr>
<td>7</td>
<td>6.62*10^14</td>
<td>8.81*10^14</td>
</tr>
<tr>
<td>8</td>
<td>1.21*10^16</td>
<td>2.74*10^17</td>
</tr>
</tbody>
</table>

4. Conclusion

In this paper, we present a new efficient algorithm for evaluating PDG fixpoints. As we know, [26] presented a local model checker for \(\mu\)-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 algorithm is only about the square root of the time required by LAFP algorithm. Furthermore, when \(d \mod 2 = 1\), we only need to design the algorithm in the same way as \(d \mod 2 = 0\). 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
[27] T Schuele, K Schneider. Global vs. local model checking of infinite state systems. MBMV. 2004: 54-64.