

# 基于 SCC 空性检测中状态空间的缩减方法

晏荣杰<sup>1),2)</sup> 张文亮<sup>1),2)</sup> 唐稚松<sup>1)</sup>

(中国科学院软件研究所计算机科学国家重点实验室 北京 100190)

(中国科学院研究生院 北京 100039)

**摘 要** 对 Couvreur 提出的基于强连通图的空性检测算法进行改进,使基于嵌套的深度优先搜索与基于强连通图搜索算法的优势结合起来,在对基于迁移的扩展(具有多个可接受条件)Büchi 自动机进行空性检测过程中,使用一个布尔变量标识一个状态,不仅节省了内存消耗,而且一般情况下的性能明显优于已有的算法,最坏情况等同于 Couvreur 的算法.同时反例寻找过程等同于基于强连通图的检测算法.

**关键词** 空性检测;基于迁移的扩展 Büchi 自动机;可接受条件

**中图法分类号** TP301

## Truly Bitstate-Hashing for SCC-Based Emptiness Checking Algorithms

YAN Rong-Jie<sup>1),2)</sup> ZHANG Wen-Liang<sup>1),2)</sup> TANG Zhi-Song<sup>1)</sup>

<sup>1)</sup>(State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing 100190)

<sup>2)</sup>(Graduate University of Chinese Academy of Sciences, Beijing 100039)

**Abstract** The paper proposes an improvement for SCC-based emptiness checking algorithms, which combines the advantages of SCC-based and nested depth first search for finding a cycle meeting all sets of acceptance conditions. The new algorithm uses the method of bitstate-hashing during emptiness checking for transition-based generalized Büchi automata, and the performance outperforms that of Couvreur's. Meanwhile, the performance for finding a counterexample is the same as that of SCC-based ones.

**Keywords** emptiness checking; transition-based generalized Büchi automaton; acceptance conditions

## 1 引 言

基于自动机理论的模型检测方法为并发及反应式系统的正确性分析提供了强有力的支持.在这个框架内,系统及正确性需求(specification)都通过有限自动机来表示.检测系统是否满足需求(又称为性质)的过程可以通过检测描述系统与需求自动机合成(product)后的自动机是否能找到一个可接受状

态序列完成.如果找到,则系统违反了需求,否则是满足的.这个过程就称为空性检测(emptiness checking).

检测系统  $M$  是否满足线性时序逻辑(Linear Temporal Logic, LTL)公式  $\phi$  描述的性质的过程,相当于检测 Büchi 自动机  $A_M \cap A_{\neg\phi}$  是否有可接受环<sup>[1]</sup>,其中  $A_M$  是描述系统的 Büchi 自动机,  $A_{\neg\phi}$  是描述性质  $\phi$  取反后的 Büchi 自动机.由于 LTL 公式在转成 Büchi 自动机时常会产生多个可接受条件,

即扩展的 Büchi 自动机(可接受条件集  $|F| \geq 1$ )<sup>[2]</sup>, 因此上述过程可以等价于寻找是否存在一个包含所有可接受条件的无限状态序列(从初始状态可达), 我们把这个序列称为可接受环(cycle).

对 LTL 性质进行检测的算法可以分为符号化(symbolic)方法及显式状态(explicit state)方法. 符号化方法通过状态集合的压缩表示缩减状态空间, 状态空间遍历过程以宽度优先搜索(breadth first search)为主. 文献[3]对各种符号化方法进行了综述及比较. 这类方法虽然节省了状态空间对内存的占用, 但检测速度较慢.

相对于符号化方法, 显式状态的 LTL 公式检测算法一般通过深度优先搜索(depth first search)方法遍历 Büchi 自动机的状态空间来寻找这样的可接受环. 对应不同的搜索策略, 显式状态的检测算法又可以分为两类:

嵌套式的深度优先搜索(Nested Depth-First-Search, NDFS). 基于 Courcoubetis 等<sup>[4]</sup>提出的 NDFS 方法<sup>[5-10]</sup>, 其本质是通过两个嵌套的深度优先搜索过程的交替进行寻找满足可接受条件的环. 这类方法的优点是可以使用偏序缩减(partial order reduction)<sup>[6]</sup>, bitstate-hashing<sup>[11]</sup>等方法缩减系统状态空间、减少内存占用. 不足是每个状态的遍历次数至少为  $|F| + 1$ <sup>[8]</sup>, 而且寻找的反例路径会比较长. 各种改进算法集中在如何加快第二次深度优先搜索结束的过程<sup>[5,7]</sup>、加快反例路径的寻找<sup>[12]</sup>及对扩展的 Büchi 自动机的检测<sup>[8,10]</sup>等.

由于满足可接受条件的环可以看做是具有所有可接受条件的强连通图(Strongly Connected Components, SCC), 另外一种算法就是基于强连通图的搜索过程. 采用 SCC 的空性检测方法<sup>[5,13-15]</sup>本质基于 Tarjan 的强连通图搜索算法<sup>[16]</sup>. 这类算法很容易对扩展的 Büchi 自动机进行检测, 但不足是比 NDFS 的算法耗费内存. 因此基于 SCC 方法的改进算法集中在缩减状态空间耗费的内存<sup>[14-15]</sup>、加快强连通图的搜索速度<sup>[5]</sup>等方面.

一般基于 SCC 的空性检测算法为了判断已遍历过的状态之间的关系, 从而合并同一个连通图中的相关状态, 通常至少采用一个整数记录遍历过状态的先后顺序. 这也是 SCC 无法采用 bitstate-hashing 的一个原因. 文献[14]将一个整数表示的信息通过 2 个布尔变量的标志来表示, 可以节省占用的内存空间, 但无法减少状态的数量.

本文采用的方法综合了 SCC 及 NDFS 方法的优

点, 即只用 1 个布尔变量记录状态是否被访问过, 且强连通图的搜索过程与 SCC 的类似. 与文献[13-14]主要的不同在于: 每次找到一个不满足可接收条件的最大强连通图后, 本文不是对已经访问的最大强连通图中所有的状态置位, 而是直接删除这些状态及状态关联的迁移. 这样不仅缩减了状态空间, 而且加快了整个空性检测过程.

本文第 2 节介绍基本概念; 第 3 节给出 Couvreur 的算法(简称为 Cou99 算法)及改进后的算法, 并证明改进算法的正确性; 第 4 节讨论改进算法的复杂度、局限性及相关工作; 第 5 节给出实验结果; 最后是结论及下一步工作.

## 2 基本概念

LTL 公式转成 Büchi 自动机后通常带有多个可接受条件, 我们把这样的自动机称为扩展的 Büchi 自动机(generalized Büchi automaton)<sup>[2]</sup>. 根据不同的算法, 可接受条件可以标识在自动机的状态上, 也可以标识在状态之间的迁移上. 将可接受条件标识在迁移上的扩展 Büchi 自动机称为基于迁移的扩展 Büchi 自动机(Transition based Generalized Büchi Automaton, TGBA). 下面给出它的形式定义.

**定义 1**(TGBA).

字母集  $\Sigma$  上的基于迁移的扩展 Büchi 自动机是标号及可接收条件出现在迁移上的 Büchi 自动机. 它的形式定义为元组  $A = \langle \Sigma, Q, \delta, q_0, \mathcal{F} \rangle$ , 其中

$\Sigma$  是字母集;

$Q$  是状态的有限集合;

$\mathcal{F}$  是可接收条件的有限集合;

$q^0 \in Q$  为初始状态;

$\delta \subseteq Q \times (2^\Sigma \setminus \{\emptyset\}) \times 2^\mathcal{F} \times Q$  为迁移关系, 表示每条迁移的标号由非空的字母集  $\Sigma$  的子集及可接收条件集组成.

自动机  $A$  的一个状态序列  $\sigma$  是一条从初始状态  $q_0$  出发的由迁移关系  $\delta$  生成的无限序列  $q_0 q_1 \cdots q_i \cdots$ , 对于任意的  $q_i, q_{i+1} \in Q$ , 有  $\langle q_i, l_i, f_i, q_{i+1} \rangle \in \delta$ . 如果状态序列  $q_0 \cdots q_i \cdots q_j$  中,  $q_i = q_j$ , 则称  $q_i$  到  $q_j$  的状态序列为一个环. 从  $q_0$  出发的这样的状态序列中, 若对于任意的  $f \in \mathcal{F}$  及  $i \geq 0$ , 存在  $j \geq i$ , 使得  $f \in f_j$ , 则称这个序列是可接受的. 即由每个可接受条件标注的迁移出现了无限多次, 只有所有可接受条件都出现在同一个环上时才满足. 这样, 空性检测问题转化成检测状态空间中是否存在这样的一条可接

受序列。

一般用于空性检测的 Büchi 自动机都是非扩展的 ( $|\mathcal{F}| \leq 1$ )，可接受条件集标注在状态上。但是，LTL 公式直接转成的 Büchi 自动机一般是扩展的 Büchi 自动机，而将自动机从扩展转成非扩展的过程常导致自动机状态的成倍增长。尽管两种自动机在空性检测过程中生成的状态空间大小相同，使用 LTL 直接转成扩展 Büchi 自动机进行空性检测的过程相对简单，所以基于扩展自动机的空性检测逐渐成为研究热点<sup>[5,8-10,13-14]</sup>。

### 3 空性检测算法

显式状态的空性检测存在两类算法：嵌套的深度优先搜索算法及计算强连通图的算法。本文讨论的是基于强连通图的空性检测算法<sup>[13]</sup>的改进，即将 NDFS 算法中可以应用的 bitstate-hashing 技术用于基于 SCC 的算法中，只用一个布尔变量表示状态是否访问过。为了便于分析及比较，这里给出文献<sup>[13]</sup>中算法的框架，然后再给出改进算法。

空性算法主要是寻找状态空间中存在的无限路径，即强连通图的过程，在 Büchi 自动机形成的状态空间中，若任意两个状态之间都有彼此可达的路径，则这些状态构成了一个强连通图。如果图中只存在一个状态，并且这个状态没有指向自身的迁移，则称这个图是平凡的，否则为非平凡的。

#### 3.1 基于 SCC 的空性检测算法

文献<sup>[5]</sup>描述的算法(文献<sup>[13]</sup>中对应算法的变型)中主要的数据结构有：

(1)  $H$  表示的是状态到整数的映射。该整数记录了对应的状态在深度优先搜索过程中遍历的相对次序。

(2)  $root$  是存放 SCC 中根状态(在搜索过程中首先被遍历的 SCC 中的状态，通过它可达该 SCC 中的其它任意状态)的栈，其中  $order$  存放状态被遍历的相对次序， $la$  为所遍历迁移上的可接受条件， $acc$  用来存放该 SCC 中所有迁移的可接受条件集， $rem$  为该 SCC 中包含的状态的集合。

(3)  $todo$  记录了所有待访问状态及其相应的迁移。

该算法首先按照深度优先搜索方式对状态空间进行遍历；在遇到重复状态后，根据重复状态的相对遍历次序对  $root$  中属于同一 SCC 的状态进行合并。若合并后的 SCC 中遍历迁移的可接受条件集合

为  $\mathcal{F}$ ，则算法返回 true；若找到的 SCC 为最大强连通图(Maximal SCC, MSCC)，且不满足可接受条件，则对  $H$  中的相应状态做标志，说明它们不能构成满足可接受条件的序列，再遇到这样的状态无需做任何处理。否则，继续进行深度优先搜索过程。若所有状态都遍历后仍没有可接受序列，则返回 false，即 Büchi 自动机的状态空间中不存在满足可接受条件的无限序列。

#### Couvreur 的空性检测算法。

```

1.  $\langle \Sigma, Q, \delta, q_0, \mathcal{F} \rangle$  be the input.
2.  $todo$ : stack of  $\langle state \in Q, succ \subseteq \delta \rangle$ 
3.  $root$ : stack of  $\langle order \in N, la \subseteq \mathcal{F}, acc \subseteq \mathcal{F}, rem \subseteq Q \rangle$ 
4.  $H$ : map of  $Q \mapsto N$ 
5.  $max \leftarrow 0$ 
6.
7.  $main() \{$ 
8.    $push(\emptyset, q_0)$ 
9.   while  $\neg todo.empty()$  do
10.    if  $todo.top().succ = \emptyset$  then
11.      $pop()$ 
12.    else
13.     {
14.       $pick\ one \langle -, -, a, d \rangle$  off  $todo.top().succ$ 
15.      if  $(d \notin H)$  then
16.        $push(a, d)$ 
17.      else
18.       if  $H[d] > 0$  then
19.        if  $merge(a, H[d]) = \mathcal{F}$  then
20.         return true
21.        end if
22.       end if
23.      end if
24.     }
25.   end if
26. end while
27. return false
28. }
29.
30.  $push(a \subseteq \mathcal{F}, q \in Q) \{$ 
31.    $max \leftarrow max + 1$ 
32.    $H[q] \leftarrow max$ 
33.    $root.push(\langle max, a, \emptyset, \emptyset \rangle)$ 
34.    $todo.push(\langle q, \{ \langle s, l, a, d \rangle \in \delta \mid s = q \} \rangle)$ 
35. }
36.
37.  $pop() \{$ 
38.    $\langle q, - \rangle \leftarrow todo.pop()$ 
39.    $root.top().rem.insert(q)$ 
40.   if  $H[q] = root.top().order$  then

```

```

41. for all  $s \in \text{root.top}().\text{rem}$  do
42.    $H[s] \leftarrow 0$ 
43. end for
44.  $\text{root.pop}()$ 
45. end if
46. }
47.
48. merge ( $a \subseteq \mathcal{F}, t \in N$ ) {
49.    $r \leftarrow \emptyset$ 
50.   while ( $t < \text{root.top}().\text{order}$ ) do
51.      $a \leftarrow (a \cup \text{root.top}().\text{acc} \cup \text{root.top}().\text{la})$ 
52.      $r \leftarrow r \cup \text{root.top}().\text{rem}$ 
53.      $\text{root.pop}()$ 
54.   end while
55.    $\text{root.top}().\text{acc} \leftarrow \text{root.top}().\text{acc} \cup a$ 
56.    $\text{root.top}().\text{rem} \leftarrow \text{root.top}().\text{rem} \cup r$ 
57.   return  $\text{root.top}().\text{acc}$ 
58. }

```

### 3.2 改进算法

除了记录状态本身外,为了进行强连通图的寻找及合并,基于 SCC 的空性检测算法通常还要使用额外的整数.在状态很多时,这些整数也会浪费一定的存储空间.文献[14]通过使用两个布尔变量标识状态是否访问过及是否属于不满足可接受条件的最大强连通图.

而本文的改进主要是针对不满足可接受条件的最大强连通图进行处理.原算法在 SCC 的搜索过程中,若当前 SCC 中所有状态的后继都已经遍历过,且其中所有迁移的可接受条件并集不等于  $\mathcal{F}$ ,则它将从  $\text{root}$  栈中弹出,并在  $H$  中做相应的标志.在后面的深度优先搜索过程中,若遍历的状态  $s$  对应的整数  $H[s]=0$ ,则放弃该结点,继续  $\text{todo}$  中其它迁移的遍历过程.

若当前的连通图包含的状态集记为  $Na$ ,要被移出的最大强连通图满足这样的条件:对于任意的  $q \in Na$ ,若  $\langle q, l, f, q' \rangle \in \delta$ ,则  $q' \in Na$ .也就是说这类 SCC 中的状态不再影响满足可接受条件序列的搜索过程.但是,可能有这样的情况:存在  $q \in Q - Na$  及迁移关系  $\langle q, l, f, q' \rangle \in \delta$ ,使得  $q' \in Na$ .也就是说可能存在  $Na$  外的状态,其迁移的目标结点属于  $Na$ .如果出现这样的情况,如图 1 所示,其中  $e_2$  和  $e_3$  两条迁移构成了不满足可接受条件的最大强连通图.在原算法中,要根据迁移生成后继,并且判断对应状态在  $H$  中的标志为 0 时才遍历其它迁移.例如,图 1 中,即使  $H[A]=0, H[B]=0$ ,但是其它的迁移生成的后继在  $H$  中发现对应值为 0 后才不进行

深度优先搜索.在后继状态的计算开销比较大的情况下,这是比较浪费时间的.同时若大部分 SCC 都是不可接受的,存储这些 SCC 的状态也很浪费空间.

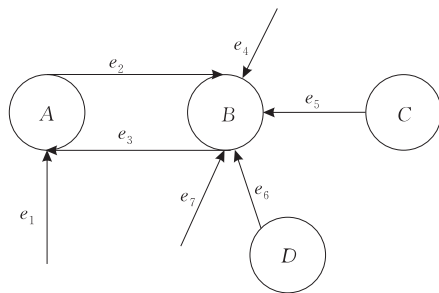


图 1 具有不满足可接受条件最大强连通图的 Büchi 自动机

根据这种情况,本文在找到一个不满足可接受条件的最大强连通图时( $\text{root}$  栈顶状态与  $\text{todo}$  栈顶遍历完所有后继的状态相同),即  $\text{root}$  栈顶元素为一个不满足可接受条件的最大强连通图的根状态时,做如下操作:

- (1) 删除不满足可接受条件的最大强连通图中对应状态相连的所有迁移;
- (2) 删除不满足可接受条件的最大强连通图中的对应状态;
- (3)  $\text{root}$  栈顶元素弹出.

只有在  $\text{root}$  栈顶的状态与  $\text{todo}$  栈顶已经完全遍历完、后继的状态相同时才执行删除对应状态及迁移的操作,这说明  $\text{todo}$  栈顶状态已经完全遍历了后继迁移,而且静态展开的迁移条件不会再发生变化.

由于同一图中任何两个最大强连通图中状态的交集为空,因此对于任何一个强连通图  $\text{SCC}_i$  及不满足可接受条件的最大强连通图  $\text{MSCC}_j$ ,且  $\text{SCC}_i \not\subseteq \text{MSCC}_j$ ,若状态  $s \in \text{SCC}_i, t \in \text{MSCC}_j$ ,且存在任意一条迁移  $s \rightarrow t$ ,则迁移  $s \rightarrow t$  不属于  $\text{SCC}_i$  的一部分.因此删除对应的迁移不会影响其它满足可接受条件连通图的形成.在这里将删除迁移的操作称为剪枝.

这样,可以不用  $H[s]$  记录不满足可接受条件最大强连通图中的状态及标志信息,而只用 1 个布尔变量表示状态是否遍历过.但这样引发的问题是  $\text{root}$  中同一 SCC 状态合并的判断.为了保证 SCC 中状态合并的正确性及整个过程的可终止性,  $\text{root}$  中需要记录状态信息, SCC 中状态的合并以遍历过程中遇到的状态是否与  $\text{root}$  中的相同为准.因此,在已经遍历过的状态空间中去掉不构成可接受序列的状态及迁移,不仅可以缩减状态空间,也可以加快可接受环的寻找.

## 改进后的空性检测算法.

```

1.  $\backslash\backslash Let \langle \Sigma, Q, \delta, q_0, \mathcal{F} \rangle$  be the input.
2. todo: stack of  $\langle state \in Q, succ \subseteq \delta \rangle$ 
3. root: stack of  $\langle w \in Q, la \subseteq \mathcal{F}, acc \subseteq \mathcal{F}, rem \subseteq Q \rangle$ 
4.
5. main() {
6.   push( $\emptyset, q_0$ )
7.   while  $\neg todo.empty()$  do
8.     if todo.top().succ =  $\emptyset$  then
9.        $\langle q, - \rangle \leftarrow todo.pop()$ 
10.      root.top().rem.insert(q)
11.      if q = root.top().w then
12.        if  $\neg todo.empty()$  then
13.          remove()
14.        end if
15.        root.pop()
16.      end if
17.    else {
18.      {
19.        pick one  $\langle -, -, a, d \rangle$  off todo.top().succ
20.        if d.visited = 1 then
21.          if merge(a, d) =  $\mathcal{F}$  then
22.            return true
23.          end if
24.        else
25.          push(a, d)
26.        end if
27.      }
28.    end if
29.  end while
30.  return false
31. }
32.
33. push ( $a \subseteq \mathcal{F}, q \in Q$ ) {
34.  q.visited = 1
35.  root.push( $\langle q, a, \emptyset, \emptyset \rangle$ )
36.  todo.push( $\langle q, \langle \langle s, l, a, d \rangle \in \delta \mid s = q \rangle \rangle$ )

```

```

37. }
38.
39. remove () {
40.  for all q  $\in$  root.top().rem do
41.    delete  $\langle \langle s', l, a, s \rangle \in \delta \mid s = q \rangle$  from  $\delta$ 
42.    delete q
43.  end for
44. }
45.
46. merge ( $a \subseteq \mathcal{F}, t \in Q$ ) {
47.  r  $\leftarrow \emptyset$ 
48.  while ( $\neg ((t = root.top().w) \mid (t \in root.top().rem)))$  do
49.    a  $\leftarrow (a \cup root.top().acc \cup root.top().la)$ 
50.    r  $\leftarrow (r \cup root.top().rem)$ 
51.    root.pop()
52.  end while
53.  root.top().acc  $\leftarrow root.top().acc \cup a$ 
54.  root.top().rem  $\leftarrow root.top().rem \cup r$ 
55.  return root.top().acc
56. }

```

## 3.3 实例

以文献[13]给出的例子作为比较,下面给出改进算法可接受环的遍历过程(如表 1 所示).为了区别与原方法的不同,这里假设同时存在一条  $H \rightarrow F$  的迁移,如图 2 所示,其中  $\mathcal{F} = \{\{a\}, \{b\}\}$ .

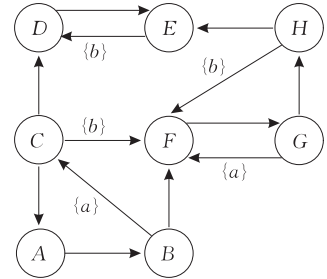


图 2 简单的扩展 Büchi 自动机

表 1 改进算法的空性检测过程

node	todo	transition	root	visited
1 A	A		$\langle A, \emptyset, \emptyset, \emptyset \rangle$	A. 1
2 A	A, B	$A \rightarrow B$	$\langle A, \emptyset, \emptyset, \emptyset \rangle, \langle B, \emptyset, \emptyset, \emptyset \rangle$	A. 1, B. 1
3 B	A, B, C	$B \rightarrow C$	$\langle A, \emptyset, \emptyset, \emptyset \rangle, \langle B, \emptyset, \emptyset, \emptyset \rangle, \langle C, \{a\}, \emptyset, \emptyset \rangle$	A. 1, B. 1, C. 1
4 C	A, B, C	$C \rightarrow A$	$\langle A, \emptyset, \{a\}, \emptyset \rangle$	A. 1, B. 1, C. 1
5 C	A, B, C, D	$C \rightarrow D$	$\langle A, \emptyset, \{a\}, \emptyset \rangle, \langle D, \emptyset, \emptyset, \emptyset \rangle$	A. 1, B. 1, C. 1, D. 1
6 D	A, B, C, D, E	$D \rightarrow E$	$\langle A, \emptyset, \{a\}, \emptyset \rangle, \langle D, \emptyset, \emptyset, \emptyset \rangle, \langle E, \emptyset, \emptyset, \emptyset \rangle$	A. 1, B. 1, C. 1, D. 1, E. 1
7 E	A, B, C, D, E	$E \rightarrow D$	$\langle A, \emptyset, \{a\}, \emptyset \rangle, \langle D, \emptyset, \{b\}, \emptyset \rangle$	A. 1, B. 1, C. 1, D. 1, E. 1
8 E	A, B, C, D		$\langle A, \emptyset, \{a\}, \emptyset \rangle, \langle D, \emptyset, \{b\}, \{E\} \rangle$	A. 1, B. 1, C. 1, D. 1, E. 1
9 D	A, B, C		$\langle A, \emptyset, \{a\}, \emptyset \rangle$	A. 1, B. 1, C. 1
10 C	A, B, C, F	$C \rightarrow F$	$\langle A, \emptyset, \{a\}, \emptyset \rangle, \langle F, \{b\}, \emptyset, \emptyset \rangle$	A. 1, B. 1, C. 1, F. 1
11 F	A, B, C, F, G	$F \rightarrow G$	$\langle A, \emptyset, \{a\}, \emptyset \rangle, \langle F, \{b\}, \emptyset, \emptyset \rangle, \langle G, \emptyset, \emptyset, \emptyset \rangle$	A. 1, B. 1, C. 1, F. 1, G. 1
12 G	A, B, C, F, G	$G \rightarrow F$	$\langle A, \emptyset, \{a\}, \emptyset \rangle, \langle F, \{b\}, \{a\}, \emptyset \rangle$	A. 1, B. 1, C. 1, F. 1, G. 1
13 G	A, B, C, F, G, H	$G \rightarrow H$	$\langle A, \emptyset, \{a\}, \emptyset \rangle, \langle F, \{b\}, \{a\}, \emptyset \rangle, \langle H, \emptyset, \emptyset, \emptyset \rangle$	A. 1, B. 1, C. 1, F. 1, G. 1, H. 1
14 H	A, B, C, F, G, H	$H \rightarrow F$	$\langle A, \emptyset, \{a\}, \emptyset \rangle, \langle F, \{b\}, \{a, b\}, \emptyset \rangle$	A. 1, B. 1, C. 1, F. 1, G. 1, H. 1

由于在去掉状态  $D, E$  构成的连通图时, 所有与  $D, E$  相关的迁移都被去掉, 所以在状态  $H$  遍历时, 只剩下  $H \rightarrow F$  的迁移, 避免了原算法还要通过  $H \rightarrow E$  的迁移访问状态  $E$  的过程, 因此改进算法寻找可接受环过程遍历迁移的次数要少于原算法, 且存储的状态个数也少于原算法.

3.4 算法的正确性

改进后的算法具有下面的性质:

- 1.  $root$  是所有构成强连通图的根状态的栈;
- 2. 被删除的是已经遍历过的, 并且不可能到达  $root$  上其它强连通图的状态;
- 3.  $root$  中状态组成序列的相对次序与  $todo$  中状态组成序列的相对次序一致.

这 3 条性质可以看成是不变式, 用来证明改进算法的正确性. 改进算法的正确性证明与文献[13]中的证明一致. 由于没有采用整数记录状态在深度优先搜索过程中的遍历次序, SCC 的合并方法略有不同, 这里给出相关证明. 为了方便表示,  $root$  中与状态  $t$  处于同一元组的  $acc, rem$  分别记为  $t_{acc}, t_{rem}$ .

下面证明在算法第 7 行, 满足这 3 条性质.

- 1. 首先, 从初始状态刚刚进入深度优先搜索循环过程时, 在第 7 行满足这几条性质.
- 2. 现在考虑若执行  $s$  到  $t$  的迁移后, 是否也满足这几条性质. 要区分下面几种情况:

若  $t$  未被访问过 ( $t.visited=0$ ), 则  $t$  及相应的信息加入到  $todo, root$ , 同时  $t.visited$  置 1. 在下次循环进入之前, 上面的三条性质依然满足.

$t.visited=1$ , 表示它已经被访问过, 但没有被删除. 这样存在两种情况:

(a)  $t$  在  $root$  中是某个强连通图的根状态, 则可以按照  $merge$ , 合并从  $root$  栈顶到  $t$  间的状态, 这时  $todo$  中的状态没有变化, 只是  $root$  弹出从栈顶到  $t$  之间的状态, 同时  $t_{rem}, t_{acc}$  也发生变化;

(b)  $t$  不是  $root$  中的根状态. 由于  $t.visited=1$ , 说明  $t$  没有被删除. 这说明  $t$  已经是某个非平凡强连通图的一部分, 若其所在连通图根状态为  $r$ , 合并过程中把它插入到  $t$  所在连通图的根状态  $r$  的  $rem$  中. 因此,  $merge$  合并到  $r$  结束,  $todo$  的状态也没有发生改变.  $root$  中被合并的状态也是从  $r$  可以到达的, 所以合并到同一个 SCC 中.

这两种情况下, 它们合并之前的 SCC 都是不可接受的 (否则此次遍历之前算法已经退出).

因此上述几条性质仍然满足. 这里没有像原算法一样考虑  $t$  已被删除的情况, 是因为若  $t$  属于一

个已经遍历过的不满足可接受条件的最大强连通图, 则在删除  $t$  时, 所有与它有关的迁移都已经被删除, 所以  $t$  不可能再被遍历到.

3. 接着考虑, 若  $todo.empty()$  成立, 则表示所有从初始状态出发的可达状态都已经遍历过, 并且构成的强连通图都不满足可接受条件, 随着  $todo$  栈顶状态的弹出,  $root$  栈顶与  $todo$  栈顶相同的状态也被删除. 仍然满足上面的 3 条性质. 因此,  $todo$  为空后,  $root$  也为空, 即不存在可接受序列. 证毕.

4 改进算法的特点及相关工作

改进算法主要通过迁移的剪枝过程达到缩减状态空间, 加快检测速度的目标. 本节根据改进算法讨论它的性能、特点及局限性, 并给出空性算法的相关工作.

4.1 反例路径的寻找

基于 SCC 的空性检测算法的一个显著特点是寻找反例速度很快, 这是因为构成反例的强连通图就是  $root$  栈顶状态所在的强连通图. 利用  $todo$  栈的特点, 可以更方便反例的寻找.

若从  $s$  到  $t$  的迁移中,  $t.visited=1$ , 并且调用的  $merge$  函数返回  $root.top().acc=\mathcal{F}$ . 那么我们不妨以  $t$  为出发点寻找反例.

根据证明中的分析,  $t$  所在的强连通图合并后用它在强连通图的根状态  $r$  表示, 而这些状态存于  $rem$  中. 即  $root$  栈顶的状态  $r$  及  $rem$  中的状态构成了满足可接受条件集合的强连通图. 该强连通图可以通过以  $r$  出发的宽度优先搜索确定最短路径. 若  $t$  本身是强连通图的根状态, 则直接从  $t$  出发进行宽度优先搜索. 下一步是寻找从初始状态到  $r$  的序列. 从初始状态开始进行宽度优先搜索寻找到  $r$  的序列, 从而构成反例.

在前面的证明过程中陈述了原算法及改进算法都具备的三条性质, 其一就是  $todo$  与  $root$  栈中状态压栈的相对次序一致. 存在于  $todo$  的状态中遍历过的迁移可能属于满足可接受条件的强连通图. 因此, 在以  $rem$  中的状态为参照进行宽度优先搜索时, 可以将搜索的范围进一步限制在这些状态已经遍历过的迁移中, 这样可以加快反例路径寻找的速度.

4.2 改进算法的复杂性分析

基于 SCC 的空性检测算法中, 每条迁移只遍历一次, 同一状态最多遍历两次. 假设每个状态用  $s$  个 bits 表示, 每个整数占用  $n$  个 bits, 自动机的状

态数目为 $|Q|$ ，迁移的个数为 $|T|$ 。

首先考虑空间复杂度。若忽略可接受条件占用的状态空间，空性检测的最坏情况为没有可接受序列，需要遍历整个状态空间。改进算法中，若没有可接受序列的同时，也没有可以去掉的状态及迁移，则表示状态是否访问过的标志位所占的空间为 $|Q|$  bits，而 *root* 由于存放的是状态，但每个 SCC 之间不会有重复状态，若只考虑与状态有关的信息占用的空间，加上表示状态访问过的布尔变量，存储这些状态占用的最大空间为 $(s \times |Q| + |Q|)$  bits。而 Cou99 算法中 *root* 及 *H* 占用的存储空间为 $2n \times |Q| + s \times |Q|$ 。改进算法中 *todo* 占用的最大空间与 Cou99 的一致，即存储待遍历的状态及迁移。

改进算法的时间复杂度分析：由于采用深度优先搜索，不存在强连通图的最坏情况下，改进算法不进行迁移的剪枝，时间复杂度与原算法一致，即与自动机的大小成正比，为 $O(|Q| \times |T|)$ 。若存在不满足可接受条件的强连通图，对其中的所有状态进行迁移剪枝的时间与连通图中状态的个数成正比。对状态进行删除的过程与原算法一致，首先都要找到对应状态，区别只是原算法对状态进行标识，而改进算法执行删除操作。

4.3 算法的局限性

改进算法是通过删除迁移来提高空性检测性能的，所以要保证删除的迁移不会影响后面强连通图的生成。因此，表示系统的多个 Büchi 自动机首先要静态合成，或在得到所有迁移规则后，保证删除的强连通图中的状态及相关迁移不影响后面连通图的生成。

一般基于 SCC 的空性检测过程中，如果性质成立，则说明系统与性质求反后合成的自动机形成的状态空间中没有满足可接受条件的环。检测过程通常是生成状态空间的同时进行状态的遍历，寻找满足可接受条件集的连通图。不管是否存在不满足可接受条件的最大强连通图，都要保存遍历过的所有状态及其它辅助信息，直到遍历完所有状态后仍未发现满足可接受条件的强连通图时为止。这是基于 SCC 空性检测过程中最浪费空间和时间的情况，也是希望的结果，即性质成立。而改进算法恰恰避免了这种空间和时间的浪费。尽管改进算法的前提需要描述系统的自动机的静态合成，但在已知所有迁移关系中寻找各个迁移组成的连通图仍然是一个耗时的过程，而减少的迁移可以缩小深度优先搜索中栈的深度，减少相同状态的访问次数，大大加快连

通图的搜索过程。在性质成立时可以较快地得到检测结果。

4.4 相关工作

文献[7]除了给出基于 NDFS 算法的改进外，还对基于 NDFS 及 SCC 的空性算法进行了详细的讨论和比较。文献[5]中的图 1 进一步给出了显式状态的空性检测算法的发展、改进过程。同时在实验部分给出了现有几种算法及改进算法的性能比较。仔细分析实验数据，不难总结出下面的几点：

- (1) 基于 NDFS 的算法中 *todo* 栈的深度一般都小于基于 SCC 的算法；
- (2) 增加启发式的改进算法的性能不一定总是优于原算法；
- (3) 基于 SCC 的算法状态遍历的次数通常少于 NDFS 算法状态遍历的次数；
- (4) Cou99 算法一般情况下优于其它算法。

在基于 SCC 的空性检测算法中，文献[5]中讨论的 Cou99 算法的两个启发式方法 Cou99 shy-和 Cou99 shy 通过重新调整每个状态后继迁移的顺序，优先遍历那些目标状态已经存在的迁移。Cou99 shy-仅考虑 *todo* 栈顶状态后继迁移的排序，而 Cou99 shy 则考虑最新遍历的强连通图中所有状态的迁移间的排序。这些改进看起来是很自然的，因为遍历过的状态越早再次访问越容易合并强连通图。但这样做的问题是调整迁移耗费的时间及遍历的迁移数目及栈占用的空间增加。

文献[15]在文章中指出，存在可接受环的情况下，它遍历的状态少于基于 NDFS 的算法。但与 Cou99 算法相比，该算法没有明显优势，而且它仅可以对一般的 Büchi 自动机进行空性检测，扩展的 Büchi 自动机首先要转化成一般的 Büchi 自动机才能使用该算法。

文献[14]采用两个布尔变量表示状态的遍历信息，但遍历的状态、迁移及栈的深度都与 Cou99 算法一致。

而本文给出的算法不仅可以节省状态空间对内存的耗费，遍历的状态、迁移个数与 Cou99 算法相比会因为迁移的剪枝而减少。

5 实 验

为了与 Cou99 算法进行比较，本文采用 Spot<sup>[17]</sup>中提供的例子 Client-Server, Echo Election Algorithm with Extinction in an Arbitrary Network

(EEAEAN)进行比较. 表 2 和 3 中分别列出不同例子对应的自动机与性质自动机合成后的状态、迁移、可接受条件个数及两种方法在可接受环的搜索过程中遍历的惟一的状态、迁移个数和 todo 栈的最大深度、是否找到可接受环. 由于遍历的迁移的个数与时间成正比,而遍历的状态及 todo 栈的最大深度与内存耗费成正比,实验结果中未列出性质验证过程中

耗费的时间与空间.

表 2 中检测的性质为  $\Box(\omega 1 \rightarrow \Diamond s1)$ ,即 Client 只要提出请求,最终肯定会被 Server 接收并提供服务.cl3serv1 表示 3 个 Client、1 个 Server 的情况,cl3serv3 为 Client,Server 各 3 个的情况.R 表示缩减后的模型描述.

表 2 Client-Server 的实验比较

情况	状态	迁移	个数	Cou99				改进后的方法			
				状态	迁移	DFS 深度	可接受环	状态	迁移	DFS 深度	可接受环
cl3serv1	783	2371	1	783	2371	511	N	521	656	138	N
cl3serv3	21394	85387	1	631	839	159	Y	631	646	159	Y
cl3serv1R	461	899	1	461	899	304	N	231	249	58	N
cl3serv3R	16669	44459	1	363	447	71	Y	363	370	71	Y

Client-Server 的性质取反转成 Büchi 自动机后具有一个可接受条件. 因此与模型的自动机合成后的状态空间中若没有可接受环,则表明性质是成立的. 分析表 2 中的实验结果,在 3 个 Client 和 1 个 Server 的模型下,没有找到可接受环,满足待检测性质,而 3 个 Client 及 Server 的模型不满足性质.

比较 Cou99 算法及改进算法的实验结果,由于两者的遍历策略一致,因此在找到可接受环时遍历的状态个数及栈的深度相同. 在遍历过程中,合成自动机的状态空间中若存在不满足可接受条件的最大强连通图,改进算法进行迁移的剪枝,因此减少了遍历的迁移及状态的个数.

EEAEAN 存在两种不同的算法. 第 1 种算法

中,一旦某个结点成为 Leader,那么在以后所有的选举中,该结点始终都会成为 Leader. 而第 2 种算法保证每个结点都有成为 Leader 的机会. 因此对同一性质进行检测的结果可能有所不同. 表 3 给出分别用两种算法进行检测的实验结果,7 条性质对应的公式分别为:

- (1)  $!\left(\left(\Diamond \Box \text{noLeader} \cup \text{zeroLeads}\right)\right);$
- (2)  $!\left(\left(\Diamond \Box \text{noLeader} \cup \text{threeLeads}\right)\right);$
- (3)  $!\left(\Diamond \text{zeroLeads}\right);$
- (4)  $!\left(\Box \Diamond \text{zeroLeads}\right);$
- (5)  $!\left(\Diamond \text{threeLeads}\right);$
- (6)  $!\left(\Box \left(\text{noLeader} \rightarrow \Diamond \text{zeroLeads}\right)\right);$
- (7)  $!\left(\Box \left(\text{noLeader} \parallel \text{zeroLeads}\right)\right).$

表 3 EEAEAN 的实验比较

情况	状态	迁移	个数	Cou99				改进后的方法			
				状态	迁移	DFS 深度	可接受环	状态	迁移	DFS 深度	可接受环
1	97679	412076	1	97679	412076	49104	N	48644	48683	329	N
2	97679	413444	1	136	136	136	Y	136	136	136	Y
3	47887	134916	0	47887	134916	115	N	47887	47886	115	N
4	97679	412076	1	97679	412076	49104	N	48644	48683	329	N
5	49332	139780	0	136	136	136	Y	136	136	136	Y
6	97679	412076	1	97679	412076	49104	N	48644	48683	329	N
7	49332	139780	1	49332	139780	49104	N	49332	139362	446	N
1	287922	1221437	1	365	365	365	Y	365	365	365	Y
2	287922	1222805	1	365	365	365	Y	365	365	365	Y
3	47887	134916	0	47887	134916	115	N	47887	47886	115	N
4	289812	1232783	1	289812	1232783	145172	N	144970	14996	787	N
5	145400	413351	0	365	365	365	Y	365	365	365	Y
6	289812	1225799	1	289812	1225799	145172	N	144970	14996	787	N
7	241808	687630	1	557	557	557	Y	557	557	557	Y

分析 EEAEAN 的实验结果,因为两种 EEAEAN 的算法不同,所以性质 1 及 7 的检测结果不同. 可以看出在性质成立,也就是不存在可接受环时,改进方法遍历的状态、迁移个数明显少于 Cou99,而

且由于遍历的状态少,所以 todo 栈的深度也小. 在性质不成立时,由于采用的迁移的遍历策略相同,所以用于发现可接受环所遍历的状态个数相同,但改进后的算法遍历的迁移个数少,而 Cou99 方法遍



历的迁移多, 同一状态可能被访问多次。

根据以上两个实验结果可以看出, 改进后的算法在一般情况下遍历的迁移个数及栈的深度方面优于 Cou99 算法, 最坏情况与 Cou99 的性能一致。

## 6 结 论

基于 SCC 的空性检测算法过程中应用迁移剪枝及 bitstate-hashing 方法, 避免了只能对不满足可接受条件集的强连通图中状态进行标记, 而不能删除的方法<sup>[13]</sup>, 或用 2 个布尔变量标识遍历过状态的做法<sup>[14]</sup>。这种改进方法不仅减少了遍历的迁移个数, 避免了部分状态的重复访问, 同时降低了存放待遍历状态及迁移的栈的深度, 降低了对内存的需求。因此, 这推广了基于 SCC 空性检测应用范围。

尽管改进方法适用的前提为系统自动机静态合成, 但对于一般的离散系统来说, 即使已知迁移关系, 寻找强连通图的空性检测过程也是比较耗时的。而改进方法可以通过减少遍历迁移次数、待遍历状态栈的深度加快空性检测速度。

下一步的工作, 首先是研究空性检测的动态过程中如何对状态进行压缩表示, 同时结合现有的状态空间缩减技术, 如偏序缩减、对称缩减简化现有的模型。文献[13]中指出这种空性检测算法与偏序缩减是相容的, 即在状态空间的遍历过程中可以用偏序缩减减少遍历的状态数目。虽然在基于 NDFS 的算法中较易使用, 如何把它用于基于 SCC 的算法中还有待进一步研究。

## 参 考 文 献

- [1] Vardi M Y, Wolper P. An automata-theoretic approach to automatic program verification (preliminary report)//Proceedings of the LICS 1986. Cambridge, 1986; 332-344
- [2] Gerth R, Peled D, Vardi M Y, Wolper P. Simple on-the-fly automatic verification of linear temporal logic//Proceedings of the 15th Work. Protocol Specification, Testing, and Verification, Warsaw, 1995; 3-18
- [3] Fisler K, Fraer R, Kamhi G, Vardi M Y, Yang Zijiang. Is there a best symbolic cycle-detection algorithm? //Proceedings of the Tools and Algorithms for Construction and Analysis of Systems. LNCS 2031. Genova, Italy, 2001; 420-434
- [4] Courcoubetis C, Vardi M Y, Wolper P, Yannakakis M. Memory-efficient algorithm for the verification of temporal properties//Proceedings of the 2nd International Workshop on Computer Aided Verification (CAV'90). Lecture Notes in Computer Science 531. New Brunswick, NJ, USA, 1991; 233-242
- [5] Couvreur J M, Duret-Lutz A, Poitrenaud D. On-the-fly emptiness checks for generalized Büchi automata//Godefroid P. Proceedings of the 12th International SPIN Workshop on Model Checking of Software. Lecture Notes in Computer Science 3639. San Francisco, USA, 2005; 143-158
- [6] Holzmann G J, Peled D A, Yannakakis M. On nested depth first search//Grégoire J C, Holzmann G J, Peled D A. Proceedings of the 2nd Spin Workshop. DIMACS; Series in Discrete Mathematics and Theoretical Computer Science 32. Rutgers University, New Brunswick, NJ, USA; American Mathematical Society, 1996; 23-32
- [7] Schwoon S, Esparza J. A note on on-the-fly verification algorithms//Proceedings of the 11th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'05). Lecture Notes in Computer Science. Edinburgh, United Kingdom, 2005; 174-190
- [8] Tauriainen H. On translating linear temporal logic into alternating and nondeterministic automata. Laboratory for Theoretical Computer Science, Helsinki University of Technology, Espoo, Finland; Research Report A83, 2003
- [9] Tauriainen H. Nested emptiness search for generalized Büchi automata//Proceedings of the 4th International Conference on Application of Concurrency to System Design (ACSD'04). Hamilton, Canada, 2004; 165-174
- [10] Tauriainen H. A note on the worst-case memory requirements of generalized nested depth-first search. Laboratory for Theoretical Computer Science, Helsinki University of Technology, Espoo, Finland; Research Report A96, 2005
- [11] Holzmann G J. Design and Validation of Computer Protocols. New Jersey: Prentice Hall, 1991
- [12] Gastin P, Moro P, Zeitoun M. Minimization of counterexamples in SPIN//Proceedings of the 11th International SPIN Workshop on Model Checking of Software (SPIN'04). Lecture Notes in Computer Science 2989. Trieste, Italy, 2004; 92-108
- [13] Couvreur J-M. On-the-fly verification of temporal logic//Wing J M, Woodcock J, Davies J. Proceedings of the World Congress on Formal Methods in the Development of Computing Systems (FM'99). Lecture Notes in Computer Science 1708. Toulouse, France, 1999; 253-271
- [14] Li Yi-Ge, Xie Kang-Lin, Hao Tao. Combining Couvreur's algorithm with bitstate-hashing for emptiness check//Proceedings of the 1st International Multi-Symposiums on Computer and Computational Sciences, Volume 2 (IMSCCS'06). Hangzhou, China, 2006; 283-286
- [15] Geldenhuys J, Valmari A. Tarjan's algorithm makes on-the-fly LTL verification more efficient//Jensen K, Podolski A. Proceedings of the 10th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'04). Lecture Notes in Computer Science 2988. Barcelona, Spain, 2004; 205-219

[16] Tarjan R. Depth-first search and linear graph algorithms. SIAM Journal on Computing, 1972, 1(2): 146-160

[17] Duret-Lutz A, Poitrenaud D. SPOT: An extensible model checking library using transition-based generalized Büchi au-

tomata//Proceedings of the 12th IEEE/ACM International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS'04). Vollandam, The Netherlands, 2004: 76-83



**YAN Rong-Jie**, born in 1977, Ph.D.. Her research interests involve formal methods, temporal logic, model checking algorithms of real-time systems, and symbolic model checking strategies.

**ZHANG Wen-Liang**, born in 1983, master. His research interests include model checking of real-time systems, formal property verification in EDA.

**TANG Zhi-Song**, born in 1925, professor, Ph. D. supervisor. His interests cover logic, automata theory, programming language and compiling method, formal semantics, and software engineering.

Background

The projects are the fundamental research of computer software theory, which aims to ensuring the correctness and reliability of safety critical systems by formal methods. Automata-theoretic approach is one of the effective methods to ensure the correctness of concurrent systems. The verification of linear temporal logic (LTL) formulas relies on an algorithm for finding accepting cycles in the product of the system and a Büchi automaton for the negation of the formula, which reduces to the checking of emptiness of a product Büchi automaton.

State space explosion is deteriorated in emptiness checking. Though many related algorithms have made some improvements, there is hardly a balance between the time and memory consumption for the emptiness checking. The au-

thors' objective is to improve the time performance and reduce the memory consumption during the emptiness checking for linear temporal logic formulas. The paper proposes an improvement for SCC-based emptiness checking algorithms, which combines the advantages of SCC-based and nested depth first search for finding a cycle meeting all sets of acceptance conditions. The work in this paper will enrich the methodology research in our projects.

Their group has made some progresses on the algorithms and symbolic data structures to relieve the problem of state space explosion, and developed several model checking tools during the years of research, which can be downloaded from the website.