基于启发式 SCCs 的广义 Büchi 自动机判空检测算法

王 曦,徐中伟
(同济大学电子与信息工程学院,上海 201804)

摘 要: 基于自动机理论模型检测的一个关键算法是判断有穷状态系统是否满足属性的判空检测.对标准 Büchi 自动机作判空检测,容易引起状态爆炸.本文以 TGBA 为研究对象,提出基于启发式 SCCs 的广义 Büchi 自动机判空检测算法.该算法在 on-the-fly 算法的基础上结合启发式深度优先搜索和 SCCs 检测算法,能较快地判断 TGBA 的非空性.通过正确性证明、复杂性分析和实验验证了该算法的正确可行性.在 TGBA 非空的情况下,该算法的时空性能比已有算法更优.

关键词: 模型检测; Büchi 自动机; on-the-fly 算法; 判空检测
中图分类号: TP301 **文献标识码:** A **文章编号:** 0372-2112 (2012) 01-0095-08
电子学报 URL: http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2012.01.016

Heuristic SCCs Emptiness Checking Algorithm for Generalized Büchi Automata

WANG Xi, XU Zhong-wei
(*School of Electronics & Information Engineering, Tongji University, Shanghai* 201804, *China*)

Abstract: The key operation of the automata-theoretic approach for model-checking is an emptiness checking algorithm, which can tell whether a finite state system satisfies its properties. It is usually done on standard Büchi automata with a single acceptance condition, whose state size is very large and the state space explosion is prone to happen. In this paper, a heuristic SCCs emptiness checking algorithm for generalized büchi automata is proposed, which is based on the on-the-fly algorithm, and can compute accepting runs of transition-based generalized Büchi automaton by heuristic depth first searching and detecting for strongly connected components. The correctness and feasibility of our method have been confirmed by theoretical proofs and experimental results. Compared with the traditional methods, while transition-based generalized Büchi automaton is not empty, the time and memory consumption are reduced in our method.

Key words: model checking; Büchi automata; on-the-fly algorithm; emptiness checking

1 引言

近年来,模型检测(model checking)[1]算法及其应用研究成了学术界和产业界研究的热点之一[2].基于自动机理论的模型检测[3]算法通常由三部分组成:(1)将表示系统属性的 LTL (Linear Temporal Logic)公式取反,并转化为广义 Büchi 自动机[4~7];(2)将广义 Büchi 自动机转化为 Büchi 自动机;(3)将 Büchi 自动机与系统模型作同步积运算[1],采用 on-the-fly 判空算法判断同步积自动机是否为空.

On-the-fly 判空算法通常用于带单个可接受条件的 Büchi 自动机的判空检测[8~12],在最坏情况下的运行时间与 Büchi 自动机的大小成线性关系.但 Büchi 自动机的状态空间与 LTL 公式的长度成指数关系,验证过程中容易引起状态爆炸.带多个可接受条件的广义 Büchi 自动机与 Büchi 自动机相比,其状态空间更少.由于广义 Büchi 自动机的判空检测比较复杂,目前有关这方面的研究不多.文献[13 ~ 15]在通常情况下能检测出广义 Büchi 自动机的非空性,但在检测过程中对结点后继的选择是任意的,作出非空性判断的时空消耗比较大.

本文提出基于启发式 SCCs 的广义 Büchi 自动机判空检测算法,以下简称 HSCCsEC 算法.HSCCsEC 算法以 TGBA (Transition-based Generalized Büchi Automaton)为研究对象,在 on-the-fly 算法的基础上,结合启发式深度优

收稿日期:2011-04-06;修回日期:2011-08-20
基金项目:国家科技支撑计划重大项目(No.2009BAG11B00,No.2011BAG01B03);国家自然科学基金(No.60674004,No.61075002)

先搜索和 SCCs 检测算法,能较快地判断出 TGBA 的非空性.在最坏情况下,该算法的空间复杂度为 $O(m|F|+m|s|)$,时间复杂度为 $O(n+m)$,其中 m 为 TGBA 中的迁移关系数,$|F|$ 为可接受条件集 F 的长度,$|s|$ 为存储结点所需空间,n 为结点数.通过理论证明和实验验证了 HSCCsEC 算法的正确可行性.与已有算法相比,HSCCsEC 算法作出非空性判断的时空性能更优.

2 基本概念及相关研究

广义 Büchi 自动机 GBA (Generalized Büchi Automaton)[13]:假设 C 为一个 GBA,其结构定义为 5 元组 $C=<\Sigma,Q,\delta,q_0,\pounds>$,其中 Σ 是有穷的非空字母表,Q 是有穷状态集,$\delta\subseteq Q\times\Sigma\times Q$ 是迁移关系,$q_0\in Q$ 是 C 的初始状态,$\pounds\subseteq 2^Q$ 是可接受条件集.

TGBA:是多个可接受条件标识在迁移关系上的广义 Büchi 自动机[14].假设 B 为一个 TGBA,其结构定义为 5 元组 $B=<AP,S,F,s_0,\Delta>$,其中 AP 是有限字母表,S 是有穷状态集,F 是 B 的多个可接受条件的集合,简称可接受条件集,$s_0\in S$ 是初始状态,$\Delta\subseteq S\times(2^{AP}\setminus\{\varnothing\})\times 2^F\times S$ 是迁移关系,把迁移关系中 F 的子集称为可接受条件.

定义 1 在 TGBA 的迁移关系 $\Delta_i=(s_i,l_j,f_j,s_j)$中,称 Δ_i 为结点 s_i 的后继迁移,s_j 为 Δ_i 的目标结点,f_j 为 s_j 的输入可接受条件.

定义 2 运行序列:$B=<AP,S,F,s_0,\Delta>$ 为一个 TGBA,B 的一个运行序列是从初始状态开始的无穷迁移序列 $<s_0,l_0,f_0,s_1>,<s_1,l_1,f_1,s_2>,\cdots,<s_j,l_j,f_j,s_{j+1}>,\cdots$. B 的一个可接受运行序列是指如果 $\forall f\in F$,$\forall i>0,\exists j>i$,使得 $f\in f_j$ 成立.

定义 3 平凡 SCCs:没有后继的单个状态.

定义 4 非平凡 SCCs 的根结点:进入非平凡 SCCs 时最先遍历的结点.

以广义 Büchi 自动机作判空检测,能有效地减少系统状态空间的搜索.文献[13]提出了嵌套的判空搜索算法,以下简称 Tauri 算法.Tauri 算法通过深度优先搜索找到一个标识了 GBA 可接受条件的状态 s_i 后,采用重复搜索的方式将 s_i 的可接受条件传递给后继结点,再重复上述过程,直到发现标识了 GBA 所有可接受条件的结点为止.文献[14]描述了基于 SCCs 搜索的判空算法,以下简称 Couv 算法.Couv 算法对 TGBA 进行深度优先搜索,根据结点的相应遍历顺序对属于同一非平凡 SCCs 的结点做合并运算,若其满足可接受条件集,返回 true,TGBA 非空;否则,继续搜索,若遍历完 TGBA 后,没有找到满足可接受条件集的非平凡 SCCs,返回 fals,TGBA 为空.文献[15]采用基于 SCCs 搜索算法对 GBA 作判空检测,以下简称 Gais 算法.Gais 算法采用深度优先搜索策略,根据结点的深度优先数和活动标志对非平凡 SCCs 作判断,若其满足可接受条件集 F,则 GBA 非空;否则,继续搜索,若搜索完 GBA 后,没有找到包含 F 的非平凡 SCCs,则 GBA 为空.

3 HSCCsEC 算法

HSCCsEC 算法的基本思想:从 TGBA 的初始结点 s_0 出发,对 s_0 后继的搜索运用式(1)所示检测函数作为启发式深度优先搜索的启发信息:

$$Value(T_{ij})=\begin{cases}1, & \text{若}(s_i,a,s_j)\in T,\text{且 } a\in F\\ 0, & \text{其他}\end{cases} \tag{1}$$

式(1)中 T_{ij}表示结点 s_i 的后继迁移,其中 $0\leqslant i\leqslant n-1$,$n$ 为结点数;$1\leqslant j\leqslant|trans(s_i)|$,$|trans(s_i)|$指 s_i 的后继迁移数.若存在 $Value(T_{ij})$为 1,算法选择 T_{ij}为搜索目标,否则任选一后继迁移 T_{ij}为搜索目标,按上述过程进一步搜索 T_{ij}指向的目标结点 s_j 及其后继迁移,在搜索过程中若发现有结点为平凡 SCCs,则对该结点及其父节点作判断,对 TGBA 中不能到达和不能构成非平凡 SCCs 的结点做删除操作,以后无需处理;若发现有满足 TGBA 可接受条件集 F 的非平凡 SCCs 存在,则终止算法并输出"Yes",TGBA 非空,否则,对此 SCCs 作进一步判断,若其为极大非平凡 SCCs,则对其做删除操作,以后不再处理;如果搜索完 TGBA 后,没有找到满足 F 的非平凡 SCCs,则算法退出并输出"No",TGBA 为空,否则,选择另一兄弟结点重复上述过程继续搜索,查找另一个满足 F 的的非平凡 SCCs.

3.1 HSCCsEC 算法描述

$A=<AP,S,T,s_0,F>$ 为一个 TGBA,由于判空检测只与 F 中的元素有关,故将 T 简化为 $T\in S\times 2^F\times S$.

HSCCsEC 算法的输入:广义 Büchi 自动机 A;

HSCCsEC 算法的输出:如果 A 非空,输出"Yes",否则输出"No".

HSCCsEC 算法的初始化定义:

Visited: array of $<ina\in 2^F, snode\in S$, Boolean $flag>$,其中 *snode* 表示已访问的结点,*ina* 表示 *snode* 的输入可接受条件,*flag* 表示 *snode* 的状态标志;

PostTrans, Part: stack of $<$Boolean $t, trans\in\{(q,a,q')|state\in S, q=state,(q,a,q')\in T\}>$,其中 *trans* 表示最近访问结点 *state* 的后继迁移,*t* 表示 *state* 的状态标志.

HSCCsEC 算法由子算法 1,子算法 2,子算法 3 和判空主程序构成,以下将详细描述.

子算法 1: PostTransDividing(Boolean $t, TQ\subseteq T$)

子算法 1 将结点 s 的后继迁移按是否带可接受条件做划分,分别存放在 PostTrans 栈的顶部和底部,具体算法描述如下:

```
PostTransDividing(Boolean t, TQ⊆T) //最近访问结点 s 的状态标志 t 及其所有的后继迁移 TQ;
1  { If TQ = ∅ then PostTrans.push(t,∅)
2  Else
3  {repeat
4     {If∃(Q,a,Q')∈TQ && a∈F then
5     part.push(t,< Q,a,Q' >) //将 t 和带有可接受条件的迁移关系压入栈 part 中;
6     Else PostTrans.push(t,< Q,a,Q' >);//将 t 和不带可接受条件的迁移关系压入 PostTrans 中;
7     TQ = TQ \ < Q,a,Q' >;//删除已读取的迁移关系;
8     If TQ = ∅ then break; //直到 TQ 为空;
9     } //end repeat
10    While part.top()≠∅ do //当栈 part 非空时;
11    {PostTrans.push(part.top());//将 part 的栈顶元素压入 PostTrans 中;
12       part.pop();} //弹出 part 的栈顶元素;
13    }//end Else of line 2;
14 }// end 子算法 1;
```

子算法 2: remove(s)

子算法 2 对无环子图中不能到达和不能构成非平凡 SCCs 的结点实施删除操作,这些结点不能构成 A 的可接受运行序列,以后遇到时无需处理.具体算法描述如下:

```
remove(s) //s 是无环子图的叶结点;
1  {update Visited.(∅,s,0);//对 s 作删除标志:将 s 的输入可接受条件和状态标志置为空集∅和 0;
2  Boolean temp = true;
3  Pick out(q1 a q1') from PostTrans.top(); //从 PostTrans 的栈顶元素中读取迁移关系(q1 a q1');
4  q = pre(s); //用 q 表示 s 的父结点;
5  If q≠q1 then //若 q 的后继结点都已访问过,即结点 q 的后继迁移不在 PostTrans 中;
6  {For all flag in Visited.(ina,post(q),flag) do
7     If flag≠0 then temp = false;//若 q 还有后继结点没带删除标志,则 temp 赋值为 false;
8  if temp then //q 所有的后继结点都有删除标志;
9  remove(q);} //递归调用 remove(q),对 q 作删除操作;
10 }// end remove(s).
```

子算法 3: Her_SCC(Node s,Boolean cur)

子算法 3 在 on-the-fly 算法的基础上,通过启发式深度优先搜索和 SCCs 检测算法,能较快地找到包含所有可接受条件的非平凡 SCCs,从而能较快地判断出 A 的非空性.具体算法描述如下:

```
Her_SCC(Node s,Boolean cur) //最近遍历结点 s 及其状态标志 cur;
1  {PostTransDividing(cur,{< q,a,q' > ⊆ T | q = s});//调用子算法 1 对 s 的后继迁移做划分;
2  While PostTrans.top()≠∅ do // 当 PostTrans 栈非空时,说明 A 还有未访问的迁移关系;
3  {If (PostTrans.top().trans = ∅) then // s 没有后继迁移,是平凡 SCCs;
4  {PostTrans.pop();remove(s);} // 弹出 PostTrans 栈顶元素,调用子算法 2 删除结点 s;
5  Pick out (q a q') from PostTrans.top() // 从 PostTrans 栈顶读取新的迁移关系(q a q');
6  PostTrans.pop(); // 弹出 PostTrans 栈顶元素;
7  read (j,q'_cur) from Visited.(q');//从 Visited 中读取 q'的存储位置和状态标志值给变量 j 和 q'_cur;
8  If j is undefined then // j 未定义,q'是未访问结点;
9     {Length = length + 1;
10    Visited[length] = (a,q',1) //将结点 q'及相应可接受条件 a、状态标志 1 写入 Visited 中;
11    Her_SCC(q',1);//以 q'为最近访问结点,递归调用 Her_SCC(q',1)继续搜索;
12 };//end If j is undefined;
13 Else if q'_cur≠0 then //否则,若 q'是已访问过、不带删除标志的结点,则有非平凡 SCCs 存在;
14    {f = ∅;f = f∪a; k = j + 1;
15    for i = k to length do
16    f = f∪Visited[i].a;//对非平凡 SCCs 包含的可接受条件进行计算,结果为 f;
17    Length = length + 1;
18    Visited[length] = (f,q_q',1);//将标识结点 q_q'、可接受条件 f 和状态标志 1 写入 Visited 中;
19    if f = F then exit "Yes"; //如果 f = F,输出"Yes"并退出,说明 A 非空;
20    Else if PostTrans.top().trans.q is not in Visited[j..length].q then //发现不满足 F 的极大非平凡 SCCs;
21    update Visited[j..length].(∅,snode,0);//将此 SCCs 做删除标志;
22    } //end Else if (q'_cur≠0)
23 } //end While
24 If PostTrans.top() = ∅ then exit "No";} //若 PostTrans 栈为空,输出"No"并退出,说明 A 为空.
```

判空主程序:EmptyCheck()

EmptyCheck()指定 A 的初始状态 s_0 的顺序号为 1,将其存入数组单元 Visited[1]中,对 A 的判空检测通过调用子算法 3 实现.具体算法描述如下:

```
EmptyCheck()
1  {length = 1;
2  Visited[length] = (∅,s0,1);//将s0和状态标志 1 写入 Visited[1]中;
3  Her_SCC(s0,1);}//以s0为最近访问结点,调用子算法 3 继续搜索.
```

3.2 HSCCsEC 算法实例

图 1 所示 TGBA 的可接受条件集 $F = \{\{a\},\{b\}\}$,HSCCsEC 算法对其作判空检测的过程如表 1.

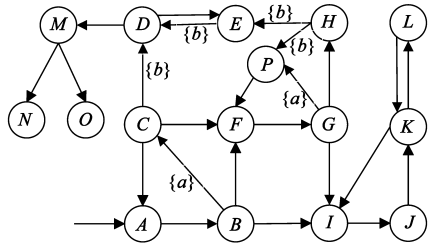

图1　HSCCsEC算法实例中的TGBA

表 1 HSCCsEC 算法的判空检测过程

step	node	Visited	Transitions in PostTrans
0	A	$(\varnothing,A,1)$;	$(A,\varnothing,B)$
1	B	$(\varnothing,A,1)$;$(\varnothing,B,1)$;	$(B,\varnothing,I)$;$(B,\varnothing,F)$;$(B,\{a\},C)$;
2	C	$(\varnothing,A,1)$;$(\varnothing,B,1)$;$(\{a\},C,1)$;	$(B,\varnothing,I)$;$(B,\varnothing,F)$;$(C,\varnothing,F)$;$(C,\{b\},D)$;
3	D	$(\varnothing,A,1)$;$(\varnothing,B,1)$;$(\{a\},C,1)$;$(\{b\},D,1)$;	$(B,\varnothing,I)$;$(B,\varnothing,F)$;$(C,\varnothing,F)$;$(D,\varnothing,E)$;$(D,\varnothing,M)$;
4	M	$(\varnothing,A,1)$;$(\varnothing,B,1)$;$(\{a\},C,1)$;$(\{b\},D,1)$;$(\varnothing,M,1)$;	$(B,\varnothing,I)$;$(B,\varnothing,F)$;$(C,\varnothing,F)$;$(D,\varnothing,E)$;$(M,\varnothing,N)$;$(M,\varnothing,O)$;
5	O	$(\varnothing,A,1)$;$(\varnothing,B,1)$;$(\{a\},C,1)$;$(\{b\},D,1)$;$(\varnothing,M,1)$;$(\varnothing,O,1)$;	$(B,\varnothing,I)$;$(B,\varnothing,F)$;$(C,\varnothing,F)$;$(D,\varnothing,E)$;$(M,\varnothing,N)$;$\varnothing$;
6		$(\varnothing,A,1)$;$(\varnothing,B,1)$;$(\{a\},C,1)$;$(\{b\},D,1)$;$(\varnothing,M,1)$;$(\underline{\varnothing},O,\mathbf{0})$;	$(B,\varnothing,I)$;$(B,\varnothing,F)$;$(C,\varnothing,F)$;$(D,\varnothing,E)$;$(M,\varnothing,N)$;
7	N	$(\varnothing,A,1)$;$(\varnothing,B,1)$;$(\{a\},C,1)$;$(\{b\},D,1)$;$(\varnothing,M,1)$;$(\underline{\varnothing},O,\mathbf{0})$;$(\varnothing,N,1)$;	$(B,\varnothing,I)$;$(B,\varnothing,F)$;$(C,\varnothing,F)$;$(D,\varnothing,E)$;$\varnothing$;
8		$(\varnothing,A,1)$;$(\varnothing,B,1)$;$(\{a\},C,1)$;$(\{b\},D,1)$;$(\underline{\varnothing},M,\mathbf{0})$;$(\underline{\varnothing},O,\mathbf{0})$;$(\varnothing,N,\mathbf{0})$;	$(B,\varnothing,I)$;$(B,\varnothing,F)$;$(C,\varnothing,F)$;$(D,\varnothing,E)$;
9	E	$(\varnothing,A,1)$;$(\varnothing,B,1)$;$(\{a\},C,1)$;$(\{b\},D,1)$;$(\underline{\varnothing},M,\mathbf{0})$;$(\underline{\varnothing},O,\mathbf{0})$;$(\underline{\varnothing},N,\mathbf{0})$;$(\varnothing,E,1)$;	$(B,\varnothing,I)$;$(B,\varnothing,F)$;$(C,\varnothing,F)$;$(E,\{b\},D)$;
10	D	$(\varnothing,A,1)$;$(\varnothing,B,1)$;$(\{a\},C,1)$;$(\underline{\varnothing},D,\mathbf{0})$;$(\underline{\varnothing},M,\mathbf{0})$;$(\underline{\varnothing},O,\mathbf{0})$;$(\underline{\varnothing},N,\mathbf{0})$;$(\underline{\varnothing},E,\mathbf{0})$;$(\underline{\varnothing},E\_D,\mathbf{0})$;	$(B,\varnothing,I)$;$(B,\varnothing,F)$;$(C,\varnothing,F)$;
11	F	$(\varnothing,A,1)$;$(\varnothing,B,1)$;$(\{a\},C,1)$;$(\varnothing,D,\mathbf{0})$;$(\varnothing,M,\mathbf{0})$;$(\varnothing,O,\mathbf{0})$;$(\underline{\varnothing},N,\mathbf{0})$;$(\underline{\varnothing},E,\mathbf{0})$;$(\underline{\varnothing},E\_D,\mathbf{0})$;$(\varnothing,F,1)$;	$(B,\varnothing,I)$;$(B,\varnothing,F)$;$(F,\varnothing,G)$;
12	G	$(\varnothing,A,1)$;$(\varnothing,B,1)$;$(\{a\},C,1)$;$(\underline{\varnothing},D,\mathbf{0})$;$(\underline{\varnothing},M,\mathbf{0})$;$(\underline{\varnothing},O,\mathbf{0})$;$(\underline{\varnothing},N,\mathbf{0})$;$(\underline{\varnothing},E,\mathbf{0})$;$(\underline{\varnothing},E\_D,\mathbf{0})$;$(\varnothing,F,1)$;$(\varnothing,G,1)$;	$(B,\varnothing,I)$;$(B,\varnothing,F)$;$(G,\varnothing,H)$;$(G,\{a\},P)$;
13	P	$(\varnothing,A,1)$;$(\varnothing,B,1)$;$(\{a\},C,1)$;$(\underline{\varnothing},D,\mathbf{0})$;$(\underline{\varnothing},M,\mathbf{0})$;$(\underline{\varnothing},O,\mathbf{0})$;$(\underline{\varnothing},N,\mathbf{0})$;$(\underline{\varnothing},E,\mathbf{0})$;$(\underline{\varnothing},E\_D,\mathbf{0})$;$(\varnothing,F,1)$;$(\varnothing,G,1)$;$(\{a\},P,1)$;	$(B,\varnothing,I)$;$(B,\varnothing,F)$;$(G,\varnothing,H)$;$(P,\varnothing,F)$;
14	F	$(\varnothing,A,1)$;$(\varnothing,B,1)$;$(\{a\},C,1)$;$(\underline{\varnothing},D,\mathbf{0})$;$(\underline{\varnothing},M,\mathbf{0})$;$(\underline{\varnothing},O,\mathbf{0})$;$(\underline{\varnothing},N,\mathbf{0})$;$(\underline{\varnothing},E,\mathbf{0})$;$(\underline{\varnothing},E\_D,\mathbf{0})$;$(\varnothing,F,1)$;$(\varnothing,G,1)$;$(\{a\},P,1)$;$(\{a\},P\_F,1)$;	$(B,\varnothing,I)$;$(B,\varnothing,F)$;$(G,\varnothing,H)$;
15	H	$(\varnothing,A,1)$;$(\varnothing,B,1)$;$(\{a\},C,1)$;$(\underline{\varnothing},D,\mathbf{0})$;$(\underline{\varnothing},M,\mathbf{0})$;$(\underline{\varnothing},O,\mathbf{0})$;$(\underline{\varnothing},N,\mathbf{0})$;$(\underline{\varnothing},E,\mathbf{0})$;$(\underline{\varnothing},E\_D,\mathbf{0})$;$(\varnothing,F,1)$;$(\varnothing,G,1)$;$(\{a\},P,1)$;$(\{a\},P\_F,1)$;$(\varnothing,H,1)$;	$(B,\varnothing,I)$;$(B,\varnothing,F)$;$(H,\{b\},P)$;$(H,\{b\},E)$;
16	E	$(\varnothing,A,1)$;$(\varnothing,B,1)$;$(\{a\},C,1)$;$(\underline{\varnothing},D,\mathbf{0})$;$(\underline{\varnothing},M,\mathbf{0})$;$(\underline{\varnothing},O,\mathbf{0})$;$(\underline{\varnothing},N,\mathbf{0})$;$(\underline{\varnothing},E,\mathbf{0})$;$(\underline{\varnothing},E\_D,\mathbf{0})$;$(\varnothing,F,1)$;$(\varnothing,G,1)$;$(\{a\},P,1)$;$(\{a\},P\_F,1)$;$(\varnothing,H,1)$;	$(B,\varnothing,I)$;$(B,\varnothing,F)$;$(H,\{b\},P)$;
17	P	$(\varnothing,A,1)$;$(\varnothing,B,1)$;$(\{a\},C,1)$;$(\underline{\varnothing},D,\mathbf{0})$;$(\underline{\varnothing},M,\mathbf{0})$;$(\underline{\varnothing},O,\mathbf{0})$;$(\underline{\varnothing},N,\mathbf{0})$;$(\underline{\varnothing},E,\mathbf{0})$;$(\underline{\varnothing},E\_D,\mathbf{0})$;$(\varnothing,F,1)$;$(\varnothing,G,1)$;$(\{a\},P,1)$;$(\{a\},P\_F,1)$;$(\varnothing,H,1)$;$(\{\{a\},\{b\}\},H\_P,1)$;	$(B,\varnothing,I)$;$(B,\varnothing,F)$
18	$f=F$,输出“Yes”并终止算法		

表 1 中 step 表示操作步骤,node 表示最近访问结点,node 栏内单元格为空,表示对最近访问结点做删除操作,Visited 栏中下划线表示的空集 $\underline{\varnothing}$ 和粗体的**0**表示对相应结点做删除标志,Transitions in PostTrans 栏显示了栈 PostTrans 中的迁移关系.

3.3 正确性证明

定义 5 将 TGBA 中存在的状态转换序列 $s\rightarrow s_1,s_1\rightarrow s_2,\cdots,s_{i-1}\rightarrow s_i,\cdots,s_j\rightarrow q$ 称为从状态 s 到状态 q 可达,记为 $s\rightarrow\cdots\rightarrow q$.

定义 6 将 TGBA 中与 $s_i\rightarrow\cdots\rightarrow s_j'$ 相应的迁移序列 $(s_i,a_i,s_i'),\cdots,(s_j,a_j,s_j')$称为从$(s_i,a_i,s_i')$到$(s_j,a_j,s_j')$可达,记为$(s_i,a_i,s_i')\rightarrow\cdots\rightarrow(s_j,a_j,s_j')$.

引理 1 若 Visited$[i].flag\neq 0$, Visited$[j].flag\neq 0$, $i<j$,则 Visited$[i].snode\rightarrow\cdots\rightarrow Visited[j].snode$ 成立.

证明　假设 HSCCsEC 算法搜索 TGBA 时,已将结点 s_{i-1}存放在 Visited[$i-1$]中,且 s_{i-1}的后继迁移为(s_{i-1}, a_i, s_i),那么当子算法 3 读取(s_{i-1}, a_i, s_i)时,执行程序行 5 – 10,将 s_i 写入 Visited[i]中,然后递归调用子算法 3,继续搜索,当读取(s_{j-1}, a_j, s_j)时,将 s_j 写入 Visited[j]中,此时(s_{i-1}, a_i, s_i)→⋯→(s_{j-1}, a_j, s_j),故 Visited[i]. *snode*→⋯→Visited[j]. *snode* 成立.

引理 2　若 $f \subseteq F$,且 f 在 TGBA 的非平凡 SCCs 中,则 f 在无穷运行的迁移序列中无穷多次出现.

证明

情形 1　f 在一个简单的非平凡 SCCs 中.

假设非平凡 SCCs 包含的迁移序列为(s_{i1}, a_{i2}, s_{i2}), (s_{i2}, a_{i3}, s_{i3}), ⋯, (s_{ij}, f, s_{ij+1}), ⋯, (s_{in-1}, a_{in}, s_{in}), (s_{in}, a_{in+1}, s_{i1}),其根结点 s_{i1}已写入 Visited 中,当 HSCCsEC 算法搜索此 SCCs 时,通过调用子算法 3,依次将(s_{i1}, a_{i2}, s_{i2})→⋯→(s_{in-1}, a_{in}, s_{in})中的迁移条件、目标结点和状态标志 1 写入 Visited 中,当继续读取(s_{in}, a_{in+1}, s_{i1})时,由于 s_{i1}已访问过且其状态标志不为 0,此时找到了一个非平凡 SCCs,其无穷运行的迁移序列为 $C1$: ((s_{i1}, a_{i2}, s_{i2}), (s_{i2}, a_{i3}, s_{i3}), ⋯, (s_{ij}, f, s_{ij+1}), ⋯, (s_{in-1}, a_{in}, s_{in}))*(“$*$”表示无穷多次运行).故 f 在 $C1$ 中无穷多次出现.

情形 2　f 在嵌套有子 SCCs 的非平凡 SCCs 中,且其根结点相同.

假设非平凡 SCCs 由 SCCs1 与 SCCs2 构成,SCCs1 嵌套在 SCCs2 中,其共同的根结点 s_{i1}已存储在 Visited 中,SCCs1 包含的迁移序列为(s_{i1}, a_{1i2}, s_{1i2}), (s_{1i2}, a_{1i3}, s_{1i3}), ⋯, (s_{ii}, a_{1i+1}, s_{1i+1})⋯, (s_{1in}, a_{1in+1}, s_{i1}),SCCs2 包含的迁移序列为(s_{i1}, a_{1i2}, s_{1i2}), (s_{1i2}, a_{1i3}, s_{1i3}), ⋯, (s_{ii}, a_{2i+1}, s_{2i+1}), ⋯, (s_{2in}, a_{2in+1}, s_{i1}),那么,HSCCsEC 算法先遍历 SCCs1 再遍历 SCCs2,形成的无穷运行迁移序列为 $C2$: ((s_{i1}, a_{1i2}, s_{1i2}), (s_{1i2}, a_{1i3}, s_{1i3}), ⋯, (s_{ii}, a_{1i+1}, s_{1i+1}), ⋯, (s_{1in}, a_{1in+1}, s_{i1}), (s_{i1}, a_{1i2}, s_{1i2}), ⋯, (s_{ii}, a_{2i+1}, s_{2i+1}), ⋯, (s_{2in}, a_{2in+1}, s_{i1}))*,由于 f 标识在非平凡 SCCs 的迁移关系上,故 f 在 $C2$ 中无穷多次出现.同理,对于非平凡 SCCs 嵌套有多个子 SCCs 的情形,f 在其无穷运行的迁移序列中无穷多次出现.

情形 3　f 在交织的非平凡 SCCs 中,其由多个有公共迁移关系、但根结点不同的子 SCCs 构成.

假设非平凡 SCCs 由 SCCs1 与 SCCs2 构成,SCCs1 包含的迁移序列为(s_{11}, a_{12}, s_{12}), ⋯, (s_i, a_{i+1}, s_{i+1}), ⋯, (s_{j-1}, a_j, s_j), (s_j, a_{1j}, s_{1j}), ⋯, (s_{1n}, a_{1n+1}, s_{11}),SCCs2 包含的迁移序列为(s_{21}, a_{22}, s_{22}), ⋯, (s_i, a_{i+1}, s_{i+1}), ⋯, (s_{j-1}, a_j, s_j), (s_j, a_{2j}, s_{2j}), ⋯, (s_{2n}, a_{2n+1}, s_{21}),那么 HSCCsEC 算法遍历 SCCs1 再遍历 SCCs2,形成无穷迁移序列为 $C3$: ((s_{11}, a_{12}, s_{12}), ⋯, (s_i, a_{i+1}, s_{i+1}), ⋯, (s_{j-1}, a_j, s_j), (s_j, a_{1j}, s_{1j}), ⋯, (s_{1n}, a_{1n+1}, s_{11}), (s_{11}, a_{12}, s_{12}), ⋯, (s_i, a_{i+1}, s_{i+1}), ⋯, (s_{j-1}, a_j, s_j), (s_j, a_{2j}, s_{2j}), ⋯, (s_{2n}, a_{2n+1}, s_{21}), (s_{21}, a_{22}, s_{22}), ⋯, (s_i, a_{i+1}, s_{i+1}), ⋯, (s_{j-1}, a_j, s_j), (s_j, a_{1j}, s_{1j}), ⋯, (s_{1n}, a_{1n+1}, s_{11}))*.由情形 2 知,f 在 $C3$ 中无穷多次出现.同理可证,当非平凡 SCCs 由多个有公共迁移关系、但根结点不同的子 SCCs 构成时,f 也在无穷运行的迁移序列中无穷多次出现.综上所述,引理 2 得证.

定理 1　$A = < AP, S, T, s_0, F >$ 为 TGBA,若 HSCCsEC 算法输出“Yes”,那么 A 中包含一个从初始状态可达的非平凡 SCCs,且满足每一个 $f_i \in F$,因此 A 非空.

证明　由子算法 3 中程序行 13 可知,对 f 进行计算的前提是最近访问结点 q 的后继结点 q' 是已访问过且未作删除标志的结点,此时执行程序行 14 – 16 对非平凡 SCCs 包含的可接受条件 f 进行计算,结果为 f,由程序行 19 知,算法输出“Yes”的唯一前提条件是 $f = F$,说明此 SCCs 满足每一个 $f_i \in F$.由引理 2 可知,f_i 在此 SCCs 的无穷运行迁移序列中无穷多次出现.假设此 SCCs 的根结点为 s_i,下面证明其从初始状态可达.

在 EmptyCheck()的程序行 2,将($\varnothing$, s_0, 1)写入 Visited[1],在程序行 3 调用子算法 3 对 A 进行搜索,当访问到(s_{i-1}, a_i, s_i)时,将(a_i, s_i, 1)存入 Visited 中.由引理 1 知,s_0→⋯→s_i 成立,故此 SCCs 从初始状态可达.

总之,若算法输出“Yes”,那么 A 中存在满足 F 无穷多次出现的无穷运行序列,且从初始状态可达,故 A 中存在可接受运行,即 A 非空.

定理 2　$A = < AP, S, T, s_0, F >$ 为 TGBA,若 A 存在可接受运行,则 HSCCsEC 算法输出“Yes”.

证明　假设 HSCCsEC 算法输出“No”,由子算法 3 中程序行 24 可知,使算法终止并输出“No”的唯一条件是栈 PostTrans 为空,此时 A 已访问完毕,在此之前程序行 19 是唯一能使算法终止的语句,因此,算法输出“No”前提是算法遍历完 A 后,没有找到满足 $f = F$ 的非平凡 SCCs,此时 A 中不存在可接受运行,这与题设矛盾,因此,若 A 存在可接受运行,则 HSCCsEC 算法输出“Yes”,判断 A 非空.

3.4　复杂性分析

引理 3　A 为一个 TGBA,HSCCsEC 算法对 A 中状态和迁移关系至多搜索一次.

证明　HSCCsEC 算法从遍历 A 的初始状态开始,通过子算法 3 对 A 进行搜索.当子算法 3 从 PostTrans 栈读取一个迁移关系后,将其出栈,以后不再访问;若目标结点没有访问过,则将其作为最近访问结点,进行新的一轮启发式深度优先搜索;若目标结点是已访问过、

作了删除标志的结点,则无需处理,继续从 PostTrans 栈读取新的迁移关系,若目标结点已访问过、且未作删除标志,此时发现有非平凡 SCCs 存在,若其满足条件 $f = F$,则终止算法并输出"Yes",否则,继续读取新的迁移关系,直到找到满足 $f = F$ 的非平凡 SCCs 或 PostTrans 栈空为止.综上所述,HSCCsEC 算法对 A 中状态和迁移关系至多搜索一次.

定理 3 $A = < AP, S, T, s_0, F >$ 为 TGBA,HSCCsEC 算法在最坏情况下的空间复杂度为 $O(m|F| + m|s|)$,其中 m 为 A 中迁移关系数,$|F|$ 为 F 的长度,$|s|$ 为存储 A 中结点所需空间.

证明 HSCCsEC 算法在搜索过程中,使用栈 PostTrans,Part 和数组 Visited 作为辅助数据结构.对于最近访问结点 s_i,子算法 1 将其所有后继迁移按是否包含可接受条件作划分,分别保存在 PostTrans 的底部和顶部,然后清空栈 Part,因此栈 PostTrans 和 Part 所占用的空间最后只需考虑 PostTrans 即可.

子算法 3 从 PostTrans 栈顶读出迁移关系($q\ a\ q'$)后,弹出 PostTrans 栈顶元素,若 q' 没有访问过,则将(a, q',1)存入 Visited 中,若 q' 已访问过且未做删除标志,则将(f, q_q',1)存入 Visited 中,搜索完 A 后,栈 PostTrans 为空.这一过程表明 PostTrans 和 Visited 所占空间最后只需考虑 Visited 即可.

在最坏情况下数组 Visited 和栈 PostTrans,Part 所占用空间最终合并为 Visited 所占用的空间.对于 $T_i = (q_{i-1}\ a_i\ q_i)$,假设存储 q_i 所需空间为 $space(q_i)$,那么存储(a_i, q_i,1)或(f, q_q_i,1)所需最大空间为 $|F| + space(q_i) + 1$.HSCCsEC 算法将 q_0 保存在 Visited[1]中,占用的存储空间为 $space(q_0) + 1$,遍历完 A 后,Visited 占用的空间为:$space(q_0) + 1 + \sum_{i=1}^{m}(|F| + space(q_i) + 1)$,令 $|s| = \max(space(q_i))$,得到 HSCCsEC 算法在最坏情况下的空间复杂度为 $O(m|F| + m|s|)$.

定理 4 $A = < AP, S, T, s_0, F >$ 为 TGBA,HSCCsEC 算法在最坏情况下的时间复杂度为 $O(n + m)$,其中 n 为 A 的结点数,m 为 A 中迁移关系数.

证明 由引理 3 知,HSCCsEC 算法在最坏情况下需要搜索 A 的结点和迁移关系各一次.A 中包含的状态和迁移关系数之和为 $n + m$,因此,HSCCsEC 算法在最坏情况下所用时间为 $O(n + m)$.

4 实验与分析

4.1 算法对比分析

本文在 DELL PC 机(2.93GHz Intel(R) Core(TM)2 Duo E7500 处理器、1.96GB 内存)上,采用C++编程实现了 HSCCsEC 算法,与 Tauri 算法、Couv 算法和 Gais 算法作对比,实验分为 2 组.

第 1 组实验根据文献[16]提出的 Globally 模式、Before 模式、After 模式、Between and 模式和 After-until 模式这 5 类常用属性,在每类属性模式下分别选取长度为 5,10,15,20 的 LTL 公式共 20 个,通过 LTL2BA 工具[7]将 LTL 公式转换成广义 Büchi 自动机,并构造迁移数为 122,状态数为 50 的系统模型,在 LTSA 中生成同步积自动机共 20 个,实验结果见表 2.表 2 中第三行至第七行的"状态"与"迁移"栏为各类属性模式下相应状态数与迁移数的平均值,"空间"栏为相应判空检测算法所遍历状态空间占总状态空间的百分比,最后一行为各栏数据汇总的平均值.

表 2 广义 Büchi 自动机判空检测算法所搜索的状态空间

属性模式	同步积自动机(平均)		Gais 算法(平均)			Couv 算法(平均)			Tauri 算法(平均)			HSCCsEC 算法(平均)		
	状态	迁移	状态	迁移	空间%	状态	迁移	空间%	状态	迁移	空间%	状态	迁移	空间%
Globally	196.75	618.75	196.5	248	54.51	73	195.75	32.96	184	613.5	97.79	27.5	80	13.18
before	163	387.75	163	189.25	63.96	98.75	168.25	48.48	172.25	417.25	107.04	54.75	108	29.55
After	139.25	327.75	134.5	162	63.49	73.75	122.75	42.08	153.25	380.25	114.24	14.25	45.75	12.85
Between and	120.75	346.25	120.75	163.75	60.92	56.25	154.25	45.08	155	388.75	116.44	15	61.75	16.43
After-until	96.25	238.75	95.25	119.75	64.17	40.75	109.5	44.85	105.25	247.75	105.37	8.25	41.25	14.78
average	143.2	383.85	142	176.55	60.44	68.5	150.1	41.48	153.95	409.5	106.91	23.95	67.35	17.32

第 2 组实验从文献[17]的 benchmark 数据中选取 10 类不同状态空间大小的广义 Büchi 自动机,实验结果见表 3 和表 4.

分析表 2、表 3 和表 4 可知,HSCCsEC 算法所需遍历的状态数和迁移数比较少,判空检测所需时间和内存空间也比较少,因而具有明显的优势.

4.2 算法应用研究

本文中以文献[18]中铁路车站联锁系统中的进路建立子系统为例验证 HSCCsEC 算法的可行性.进路建立子系统安全防护设计模型如图 2 所示.系统安全属性取反的 TGBA 模型如图 3 所示,约定可接受条件集 $F = \{testErro, selectOk\}$.图 2、图 3 中各动作行为的中文含义见表 5.

对图 2 和图 3 两个模型做同步积运算后,HSCCsEC 算法对同步积自动机作判空检测,输出"No",系统的安全防护设计满足安全属性的要求.

表 3　广义 Büchi 自动机判空检测算法所检测的状态数和迁移数

广义 Büchi 自动机大小		Gais 算法		Couv 算法		Tauri 算法		HSCCsEC 算法	
状态	迁移	状态	迁移	状态	迁移	状态	迁移	状态	迁移
1459	3705	1460	1467	1428	3627	1461	5164	8	21
3161	4609	3163	3163	2267	3131	3182	7763	24	49
5593	10793	5592	5592	5566	10738	5651	16494	58	128
5604	10805	5604	5604	3640	7136	5604	16415	68	139
7372	11147	7349	7349	7306	11054	7350	18492	9	18
10307	16007	10141	10158	10114	15785	10141	26001	47	105
24223	76762	23349	23349	1468	3751	23440	97319	82	268
39354	108439	39308	39308	38458	105724	39350	147711	39	125
41433	114574	40637	40637	37582	104097	40694	153119	49	163
68274	190834	65918	65918	65068	180737	65992	249392	64	213

表 4　广义 Büchi 自动机判空检测算法所花费的时间和内存空间

广义 Büchi 自动机大小		Gais 算法		Couv 算法		Tauri 算法		HSCCsEC 算法	
状态	迁移	时间(s)	内存(KB)	时间(s)	内存(KB)	时间(s)	内存(KB)	时间(s)	内存(KB)
1459	3705	0.11967	8508	9.37491	9756	117.78892	9720	0.00023	7544
3161	4609	0.21107	9748	7.06191	10256	515.426189	11804	0.00058	8236
5593	10793	0.37513	11740	77.40675	13476	1709.16254	15524	0.00122	9044
5604	10805	0.35189	10580	27.86317	13740	1705.38039	20080	0.00094	8936
7372	11147	0.44059	10876	71.44794	15116	2737.19811	15932	0.00012	9260
10307	16007	0.65459	13768	263.57179	16176	5482.59261	19736	0.00076	10608
24223	76762	1.77943	24100	8.42825	21708	31359.75976	36720	0.00207	15580
39354	108439	2.95222	31592	771.49672	44040	86699.24696	53872	0.00073	19716
41433	114574	2.98126	38272	724.71693	45884	92818.99161	56780	0.00093	23220
68274	190834	4.86388	52912	1955.43179	76128	245138.69383	89576	0.00124	29036

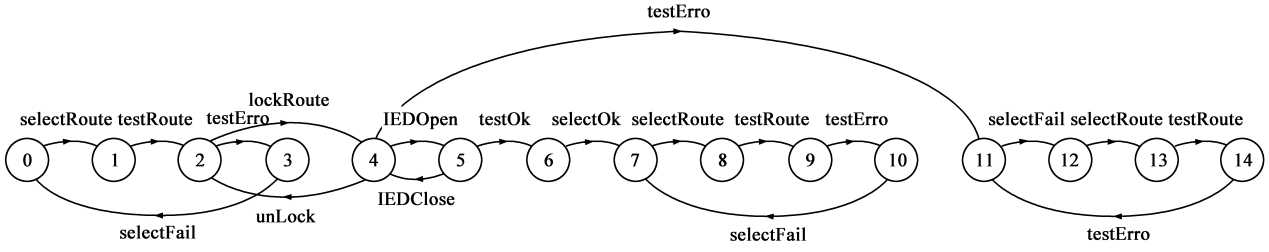

图2　进路建立子系统的状态迁移图

表 5　图 2 ~ 图 4 中的中英文对照

英文	中文含义	英文	中文含义
selectRoute	进路预选	testOk	进路测试正常
testRoute	进路测试	selectOk	进路选择成功
testErro	进路测试失效	selectFail	进路选择失败
lockRoute	锁闭进路	lEDOpen	开放信号灯
unLock	不锁闭进路	lEDClose	关闭信号灯

若将图 2 中迁移关系“(s10, selectFail, s7)”改成(s10,selectOk,s7),与图 3 所示模型作同步积生成的同步积自动机如图 4,HSCCsEC 算法对其作判空检测,输出“Yes”,修改后的系统模型不满足安全属性,此时形成相应反例为:selectRoute - > testRoute - > testErro - > selectOk.

以上实验表明,HSCCsEC 算法能对进路建立子系统的安全防护设计是否满足安全属性进行正确的验证,因此,HSCCsEC 算法是切实可行的.

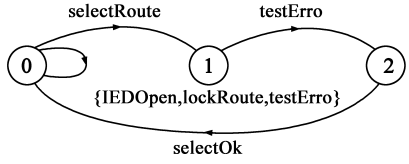

图3　系统安全属性取反的TGBA模型

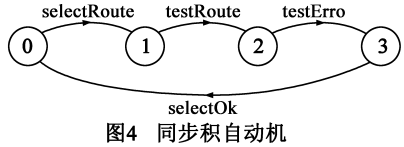

图4　同步积自动机

5　结束语

本文以 TGBA 为研究对象,提出了 HSCCsEC 算法,并作了正确性证明与复杂性分析.HSCCsEC 算法在 on-the-fly 算法的基础上结合启发式深度优先搜索和 SCCs 检测算法,克服了传统的深度优先搜索算法的盲目性,

加强了对 TGBA 中不能到达和不能构成非平凡 SCCs 的结点和不满足可接受条件集的极大非平凡 SCCs 的处理,能较快地判断出 TGBA 的非空性.在广义 Büchi 自动机非空的情况下,通过实验与分析验证了 HSCCsEC 算法的时空性能更优,且形成相应反例也比较容易.HSCCsEC 算法为缓解形式化验证所面临的状态空间爆炸问题提供了切实可行的解决途径,为软件开发过程中系统模型的修改和维护提供了方便,也为安全苛求系统的安全性验证与评估提供了有力支撑,丰富了基于模型的形式化开发方法.

参考文献

[1] Baier C, Katoen J P. Principles of Model Checking[M]. Massachusetts: The MIT Press, 2008.

[2] 林惠民,张文辉.模型检测:理论、方法与应用[J].电子学报,2002,30(12A):1907 – 1912.
LIN Hui-min, ZHANG Wen-hui. Model checking: theories, techniques and applications[J]. Acta Electronica Sinica, 2002, 30(12A):1907 – 1912. (in Chinese)

[3] Edelkamp S, Lafuente A L, Leue S. Directed explicit model checking with HSF-Spin[A]. Proceedings of the 8th International SPIN Workshop on Model Checking of Software[C]. New Youk: Springer-Verlag, 2001.57 – 79.

[4] Gerth R, Peled D, Vardi M Y, Wolper P. Simple on-the-fly automatic verification of linear temporal logic[A]. Proceedings of PSTV'95[C]. London: Chapman & Hall, Ltd, 1995.3 – 18.

[5] Giannakopoulou D, Lerda F. From states to transitions: improving translation of LTL formulae to büchi automata[A]. Proceedings of FORTE 2002[C]. New Youk: Springer-Verlag, 2002.308 – 326.

[6] Somenzi F, Bloem R. Efficient büchi automata from LTL formulae[A]. Proceedings of CAV 2000[C]. London: Springer-Verlag, 2000.247 – 263.

[7] Gastin P, Oddoux D. Fast LTL to büchi automata translation [A]. Proceedings of CAV 2001[C]. London: Springer-Verlag, 2001.53 – 65.

[8] Schwoon S, Esparza J. A note on on-the-fly verification algorithms[A]. Proceedings of TACAS'05[C]. London: Springer-Verlag, 2005.174 – 190.

[9] Couvreur J M. On-the-fly verification of linera temporal logic [A]. Proceedings of FM'99[C]. London: Springer-Verlag, 1999.253 – 271.

[10] Cerna I, Pelanek R. Relating hierarchy of temporal properties to model checking[A]. Proceedings of MFCS'03[C]. Berlin: Springer-Verlag, 2003.318 – 327.

[11] Geldenhuys J, Valmari A. Tarjan's algorithm makes on-the-fly LTL verification more efficient[A]. Proceedings of TACAS 2004[C]. Berlin: Springer-Verlag, 2004.205 – 219.

[12] Geldenhuys J, Valmari A. More efficient on-the-fly LTL verification with Tarjan's algorithm[J]. Theoretical Computer Science, Elsevier, 2005, 345(1):60 – 82.

[13] Tauriainen H. Nested emptiness search for generalized Büchi automata[J]. IEEE Computer Society, 2004, 70(1): 165 – 174.

[14] Couvreur J M, Lutz A D, Poitrenaud D. On-the-fly emptiness checks for generalized Büchi automata[A]. Proceedings of SPIN[C]. New Youk: Springer-Verlag, 2005.169 – 184.

[15] Gaiser A, Schwoon S. Comparison of algorithms for checking emptiness on Büchi automata[A]. Proceedings of MEMICS'09[C]. Dagstuhl: Schloss Dagstuhl Leibniz-Zentrum fuer Informatik, 2009.69 – 77.

[16] Dwyer M B, Avrunin G S, et al. Patterns in property specifications for finite state verification[A]. Proceedings of the 21st International Conference on Software Engineering[C]. Los Alamitos: IEEE Computer Society Press, 1999.411 – 420.

[17] Pelánek R. Beem: benchmarks for explicit model checkers [A]. Proceedings of the 14th International SPIN Conference on Model Checking Software[C]. Berlin: Springer-Verlag, 2007.263 – 267.

[18] 王曦,徐中伟,梅萌.基于模型检测的软件安全性验证方法[J].武汉大学学报(理学版),2010,56(2):156 – 160.
Wang Xi, Xu Zhong-wei, Mei Meng. A software safety verification method based on model checking[J]. Wuhan University (Nat. Sci. Ed.), 2010, 56(2):156 – 160. (in Chinese)

作者简介

王 曦 女,1974 年 4 月生,湖南双峰人.1996 年毕业于贵州大学,其后在中国测绘科学研究院工作,现为同济大学电子与信息工程学院博士生,主要从事模型检测、安全软件的形式化验证与评估等研究.
E-mail: wang_xi_happy@163.com

徐中伟 男,1964 年 6 月生,江苏无锡人.1984 年毕业于上海铁道学院,2000 年博士毕业于同济大学,现为同济大学教授,博士生导师,主要从事基于通信的列车控制系统,软件、通信协议的安全性形式验证和测试评估等研究.