基于广义符号轨迹赋值模型验证的反例的产生

李义年[1] 曹占涛[2] 郑德生[2] 杨国武[2]
(武汉理工大学理学院 武汉 430063)[1] (电子科技大学计算机科学与工程学院 成都 610054)[2]

摘 要 广义符号轨迹赋值(Symbolic Trajectory Evaluation)引入了符号变量和抽象技术,解决了验证中状态爆炸的问题,但是却为寻找反例制造了很多障碍。针对此,提出了一种高效的寻找反例的算法,它应用集合的概念,通过回溯在父子路径之间进行集合的交集,可以高效地解决抽象引起的问题。并对此算法进行改进,解决了符号变量带来的问题。

关键词 广义符号轨迹赋值,反例,形式化验证,符号模型验证,抽象
中图法分类号 TP301.2 **文献标识码** A

Counter-example Generation in Generalized Symbolic Trajectory Evaluation

LI Yi-nian[1] CAO Zhan-tao[2] ZHENG De-sheng[2] YANG Guo-wu[2]
(College of Science, Wuhan University of Technology, Wuhan 430063, China)[1]
(School of Computer Science and Engineering University of Electronics Sci. & Tech. of China, Chengdu 610054, China)[2]

Abstract Generalized Symbolic trajectory evaluation is a powerful model checking technique which introduces symbolic quaternary and symbolic variables into symbolic quaternary assignments. But to find Counter-example has obstacles. In this paper, we presented an solution to search Counter-example. It uses set Intersection to backward simulation to generalized Counter-example. And we extended the solution to solve the problems from symbolic variable.

Keywords Generalized symbolic trajectory evaluation, Counter-example, Formal verification, Symbolic model-checking, Abstraction

超大规模电路日益复杂,验证已成为整个设计流程中的重要步骤和瓶颈。2003年度的国际半导体技术发展报告(International Technology Roadmap for Semi-conduct, ITRS 2003)[1]指出,验证已经成为集成电路设计流程中开销最大的工作。在目前的工程项目中,验证工程师的数目超过了设计工程师,对于复杂的设计更是达到了2∶1或者3∶1的比率;验证的时间占整个设计周期的50%~80%。

目前形式化验证[2,3]的研究热点之一是符号模型检验方法,因为在传统的模拟中,激励与响应都是由二进制位组成的。在一个由 n 个输入位的组合电路中,输入空间有 2^n 个向量,传统模拟器每次只能模拟一个向量。与之相反,符号模拟的输入是变量,并以变量产生输入表达式。在每个周期中,一个输入就是一个自由变量。由于变量表示0或1,符号模拟器同时模拟输入变量可以取到点的全部集合。该方法需要用户提供待验证系统的描述和期望成立的断言描述,并完全自动化地验证该断言在待验证系统上是否成立。

相对于另一种主要的形式化验证手段——定理证明,符号模型检验的一个主要的优点在于能够高效地通过产生的反例,来解释断言不成立的原因。但符号模型检验面对的最大问题是,当系统很大时,就会发生状态爆炸。基于三值抽象技术的符号轨迹赋值(Symbolic trajectory evaluation, STE)[3]和一些经典的符号模拟验证(SMC)[4]很大程度地减少了状态爆炸问题,但STE的局限是无法验证无限时间的性质。

GSTE(Generalized symbolic trajectory evaluation,广义符号轨迹赋值)[5]对STE进行了扩展,不仅可以验证无限时间的性质,而且比STE更高效和容易使用。

GSTE和STE可以高效地验证大型系统并防止发生状态爆炸的关键是引入了抽象技术。由于在模拟的过程中,将一些值表示为不确定值 X,在模拟过程中就隐藏了一些变量的信息,从而为寻找反例产生了很多障碍。

本文设计了一种高效的发掘出由于抽象而被隐藏的信息,从而找出正确反例的算法。

1 广义符号轨迹赋值 GSTE

电路模型和断言图模型的定义,对于STE和GSTE比较重要。其具体定义如下。

1.1 电路模型和断言图

一个电路 M 就是一个布尔节点的集合 N,一个状态就是对 N 的一次赋值。结点集合 N 可以被分成两个集合:状态节点集合 N_S 和输入节点集合 N_I。对每个状态结点 $n \in N_S$

到稿日期:2010-01-20 返修日期:2010-03-25 本文受国家自然科学基金(60973016)资助。
李义年(1963-),男,讲师,主要研究方向为计算数学;**曹占涛**(1983-),男,硕士生,主要研究方向为形式化验证等;**郑德生**(1983-),男,博士生,主要研究方向为形式化验证、计算数学等,E-mail:zheng_de_sheng@163.com;**杨国武**(1966-),男,博士,教授,博士生导师,主要研究方向为形式化验证、计算数学和逻辑综合等。

存在一个迁移关系函数 $X_n(N)$，迁移函数定义了状态间的传输关系。

迁移关系也可以等价地记为 $R(N,N')$，$R(N,N')$ 等于对于所有 $n\in N_S$ 有 $(n'=X_n(N))$，其中 N' 是 N 经过迁移函数计算后的状态。

本文定义 *trace* 是一个状态序列 $\sigma=[s0,s1,\cdots]$，对每一个 i，每个状态结点 $n\in N_S$ 有 $S_{i+1}(n)=X_n(S_i(n))$，$S_{i+1}(n)$ 对所有的 $n\in N_I$ 的输入节点无限制。

断言图主要用来描述所要验证的电路性质，将断言图定义为 5 元组，$G=(V,v_i,E,ant,cons)$

其中，

1. V 是有限的结点集合。

2. $v_i\in V$ 是初始结点集合。

3. E 是 $V\times V$ 的子集，表示边集，满足于：任意的 $u\in V$，存在 $v\in V,(u,v)\in E$。

4. ant：对每一个边 $e\in E(E\to 2^S)$ 有一个条件 $ant(e)$。

5. cons：对每一个边 $e\in E(E\to 2^S)$ 有一个目标 $cons(e)$。

1.2 STE 和 GSTE

STE 是一种基于格论的模型检测技术。1991 年 Bryant 和 Beatty 等人探索将模拟的方法和形式化方法结合，以此充分结合两种验证方法的优点，克服其中的不足，他们进行了最初的试验并取得了一定的成功。后来 Segar 和 Bryant 等人不断进行改进和完善，1995 年新方法最终以完整的理论形式出现并正式定名为 STE[3]。今天 STE 验证方法已是数字电路验证的最有效和最快速的验证方法之一，它不仅能够完成传统的模拟验证方法可以完成的验证任务，而且尤其适合验证流水线等数字时序电路，现正在被许多公司使用，如 Intel，IBM，Motorola。STE 在集成电路验证方面显示出强大的优越性，例如，它曾经成功应用于检验几种 Motorola 的拥有百万门级电路的存储单元。在 Intel，它曾被应用于检测浮点数单元和 12,000 个门的复杂数据路径单元。但 STE 的缺陷是不能进行无限时间逻辑的验证。

GSTE 理论是由 STE[5] 理论发展而来，GSTE 理论是在 STE 基础之上，对 STE 理论的一种扩展，GSTE 和 STE 一样是基于格伦的形式化模型检测技术，它是一种经典的模型检测技术，在验证领域的应用是很灵活的，而且可以减少状态爆炸，在验证大型集成电路方面，它已经显示了很大的潜力。和 STE 相比，它在相同的时间和空间效率下能去验证所有的 ω-regular 属性。

1.3 四值运算及符号索引的反例解决

STE 为了解决状态爆炸，使用四值{0,1,*X*,*T*}来代替布尔变量的 0,1。图 1 表示了四值之间的关系，其中 *X* 表示包含比 0 或者 1 更少的信息。*T* 表示比 0 和 1 还要多的信息。

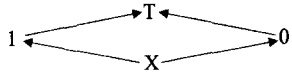

图 1 四值关系

在四值抽象中，任何状态集合都可以被精确地表示，如果一个点在状态集合中的所有状态中，保持不变，就可以用 0,1 来表示，否则用 *X* 来表示，因为冲突而产生的空用 *T* 来表示。

四值操作技术的引入极大地减小了计算中的状态空间，因为对于具有 n 个布尔节点的电路，具有 2^n 个不同的状态，从而具有 2^{2^n} 个不同的状态集合。但是采用了四值运算技术后，只有 4^n 个四值转换。四值抽象的操作是由集合操作转换过来的，如图 2 所示，使用起来也很方便。

&	X	0	1	T
X	X	0	X	T
0	0	0	0	T
1	X	0	1	T
T	T	T	T	T

|	X	0	1	T
X	X	X	1	T
0	X	0	1	T
1	1	1	1	T
T	T	T	T	T

!	
X	X
0	1
1	0
T	T

Y	X	0	1	T
X	X	0	1	T
0	0	0	T	T
1	1	T	1	T
T	T	T	T	T

1	X	0	1	T
X	X	X	X	X
0	X	0	X	0
1	X	X	1	1
T	X	0	1	T

图 2 四值运算

但是四值运算也带来了一些问题，由于很多常量节点在抽象过程中丢失，就会带来伪报错的问题，为了解决此问题，采用了符号索引的技术。

例如对于 3 个节点 p,q 和 r，有如下的集合：

$\{[p=0,q=1,r=1]\}$

$\{[p=1,q=0,r=0]\}$

$\{[p=1,q=0,r=1]\}$

采用四值抽象技术后，将会得到如下集合：$[p=X,q=X,r=X]$。

很明显，对以后的模拟都会只产生 *X*，为了解决此问题，现在所引入的一个方法是用符号常量 *C*，上述集合可以被精确化为如下索引表达式：

$(!\ C\to[p=0,q=1,r=1])\wedge(C\to[p=1,q=0,r=X])$

如图 3 所示，当 *C* 是 0 时，第一个四元组结构是有效的，当 *C* 是 1 时，第二个四元组结构是有效的。

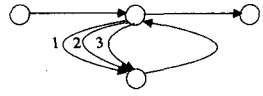

图 3 断言路径

符号常量在 GSTE 中并不总是成立[5]，所以在 GSTE 中引入了符号变量。

如对于 3 个断言路径

(1) $c=0\&s1=1\&s2=0$

(2) $c=0\&s1=0\&s2=1$

(3) $c=0\&s1=1\&s2=1$

可以写成如下形式：$c=0\&s1=c1\&s1=!\ c1|c2$，如图 4 所示。

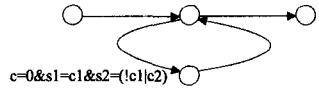

图 4 符号变量断言路径

四元组结构及符号变量的引入，使模拟过程更高效，更大程度地减少断言图的边，但是对反例查找又引入了新的障碍。由于变量索引是在模拟过程中引入，在反例的回溯过程中，可能会产生错误的回溯路径。

2 寻找反例的算法设计

GSTE 理论中采用了抽象技术和符号变量索引技术，从而大大地减少了状态爆炸，但负作用是为反例的寻找增加了

难度。原因是在模拟过程中引用了抽象技术、符号变量索引技术，再加上系统的复杂性，模拟路径就会很长，经过层层的抽象后，很多在前向模拟中的信息就会被隐藏，如果发生了错误，需要回溯，当回溯的时候，就需要隐藏信息，如何将隐藏的信息挖掘出来，是面临的一个问题。

2.1 相关研究

Păsăreanu 等人[6]通过扩展 Java PathFinder(JPF)model-checker 来产生精确的反例，他们应用的两个技术一个是不完全的，另一个没有考虑输入不相关性。SLAM[7]对其进行了一些限制，他的目的是只验证 boolean 型程序，使用启发式算法来判断路径是否是可达的，但往往会得到“不知道是否可达”的结果。

一个更一般的方法被 Clarke 等发表，其算法采用的方法是，如果不能产生精确的反例，就对抽象进行重新定义。此算法允许来自抽象集合路径的一个精确路径集合的计算。如果结果集是空的，很明显，这个程序的反例是不存在的。此算法可以计算依靠于抽象到具体的方向运算。Yan Chen[8]在 GSTE 中做了一个自动重定义的算法，但是此算法有很多缺陷，以及由于需要对抽象进行重新定义，需要非常专业的知识背景才行，应用面不够广。

2.2 算法的基本思想

当验证算法发生断言错误时，验证的过程同时也会提供一个符号模拟的历史路径。

定义 1 历史路径由三元组 (e,s,d) 组成，其中 e 是 GSTE 算法模拟的一条边，s 和 d 分别是边 e 模拟以前的状态和模拟后的状态。

定义 2 反例 CE=$[t_1,t_2,\cdots,t_l]$对于每一个 $1\leqslant i\leqslant l$，$t_i=(e_i,s_i,d_i)$，其中 e_i 是转化步骤 i，并且 s_i 是 e_i 模拟以前的状态，d_i 是 e_i 模拟以后的状态。

通过定义 1 和定义 2，由 t_i 就可以通过转化关系找到 t_{i-1} 三元组 $(e_{i-1},s_{i-1},d_{i-1})$，按此方法一直后向模拟，直到达到断言图的初始向量边，就可以找到造成此断言错误的一个反例 CE(counterexample)。

在模拟过程中，遇到需要抽象的地方，就对此节点的抽象信息进行记录。当在某些点发生错误时，正向模拟停止，此时必定有 sim(e)不属于 cons(e)发生，设 SC=sim(e)−cons(e)。

通过定义 1 和定义 2 的方法来表示 SC，并通过转换关系的反函数对其求值，从而可以获得 SC 对应的上一个状态的值，继续回溯上一状态，如果发现某一个状态引入了抽象，即对此抽象进行了还原，继续回溯，即可得到一个正确的反例。

2.3 算法设计

设 Tree 为模拟过程中产生的路径图，并设产生错误的那个路径的边为第 1 层，后退 1 层为第二层，后退 i 层，为图的第 $i+1$ 层。I 为抽象节点重要信息记录表，每回溯一步的时候，根据记录信息进行判断，如果需要则扫描 I，将抽象还原。

对引入符号变量的路径节点，将此节点信息放到一个 hash 表中，如果回溯到此路径节点，通过搜索 hash 表，得到引入符号变量前的状态，按错误信息，从路径中选择一条路径，继续进行回溯，从而可以正确地找到反例。

err 为对应的出现断言不满足的边。

Link 为一个链表。

Link. Input(*err*)：表示状态信息放入队列。

s'=sim(*err*)−cons(*err*)：产生错误的点。

算法的伪代码实现如下：

```
输入：Tree，pre，err
输出：错误路径
算法 1  CE(Tree,pre,err)
{
e=Link. tail;
for(对 e 的每一个前驱边 e′)
{//sim(e)′为计算前的 sim(e)；
  If(sim(e)′⋃post(sim(e′))⋂ant(e) ⋂ s′
    ! =NULL)
  {
    s_pre= pre(sim(e)′ ⋃post(sim(e′)) ⋂ant(e) ⋂ s′);
    //如果有抽象，则将抽象还原
    If(need)
    {
      s_pre = Scanf( I,s_pre );
    }
    //如果 hash 表有此符号变量则将选路
    if(hash(s_pre))
    {
      s_pre = Hash(s_pre);
    }
    Link. Input(e′);
      If( ( Link. tail==初始边)&(sim(Link. tail) ⋂s_pre! =
          NULL) )
    {
        逆向输出队列 Link。
        程序停止。
    }
    else if( sim(Link. tail)⋂s_pre! =NULL )
    {
      s′ = s_pre;
      CE( T,pre,e′)
    }
    else
    {
  Link. delete(tail)
    }
  }
}
```

3 算法分析

3.1 算法分析

很明显，正向模拟路径生成一个图，在此图中对于每一步回溯，我们采用了求交集的方法，从而可以找到错误子节点的来源节点。根据抽象记录信息，如果有地方采用了抽象，将扫描 I，将抽象进行还原。如果没有交运算，有可能会达到一个不包含错误状态的初始状态。

对引入符号变量的路径节点，将此节点信息放到一个 hash 表中，提高了查找效率，可以更快地找到反例。

3.2 例证

本节将通过一个例子，来解释算法的流程，其中图 5 是一个 Model 图，图 6 是一个断言图，运用 GSTE 算法后，由表 1 可以得出，Model 并没有满足断言图的要求，产生了错误。

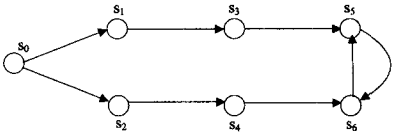

图 5 Model

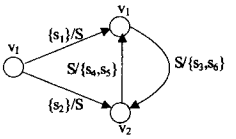

图 6 断言图

表 1 GSTE(G,post)

	V1,V1	V1,V2	V1,V2	V2,V1
0	{s1}	T	{s2}	T
1	{s1}	{s3}	{s2}	{s4}
2	{s1}	{s3,s6}	{s2}	{s4,s5}
4	{s1}	{s3,s6}	{s2}	{s4,s5}

其中，*err* 为 V1V2。

首先求 s'：

$s'=\mathrm{sim}(err)-\mathrm{cons}(\mathrm{err})=\{s6\}$

应用 2.3 节提出的算法来解决，详细过程如表 2 所列。

表 2 CE(Tree,pre,err)

	V1,V1	V1,V2	V1,V2	V2,V1	Link	s'
0	{s1}	{s3,s6}	{s2}	{s4,s5}	V1V2	{s6}
1	{s1}	{s3,s6}	{s2}	{s4,s5}	V1V2→V2V1	{s4,s5}
2	{s1}	{s3,s6}	{s2}	{s4}	V1V2→V2V1→V1V2	

通过表 2 可以看出，此方法可以很好地解决反例查找问题，可以方便地查找到反例。

结束语 本文在 GSTE 的理论背景下，提出了一种寻找反例算法，算法通过在模拟过程中记录下每个边 e 的每一个 sim(e)，无论计算过程中怎么抽象，都可以找到产生错误的前一条边。并将采用符号变量的路径节点放入 hash 表中，每次都检索 hash 表中是否有此节点，如果有就对其进行路径选择，从而找到正确的反例路径。

如果此算法引入启发式的算法，通过它来解决处理边的循环和分裂，反例的查找会更高效。

参 考 文 献

[1] ITRS2003[OL]. http://public.itrs.net/

[2] 韩俊刚，杜惠敏. 数字硬件的形式验证[M]. 北京：北京大学出版社，2001

[3] 刘楠，朱文也，祝跃飞，等. 基于树语言逼近的安全协议形式化分析[J]. 计算机科学，2010(1)

[4] Jin Yang, Seger C-J H. Generalized Symbolic Trajectory Evaluation[C]// Proceedings of 2001 IEEE International Conference on Computer Design. 2001:360-365

[5] Jin Yang, Seger C-J H. Introduction to generalized symbolic trajectory evaluation[J]. IEEE transactions on very large scale integration(VLSI) systems, 2003, 11(3)

[6] Matthew B, Dwyer Corina S, et al. Finding feasible counter-examples when model checking Java programs[C]// Tools and Algorithms for the Construction and Analysis of Systems (TACAS), volume 2031 of Lecture Notes in Computer Science. Springer-Verlag, 2001

[7] Ball T, Rajamani S K. Checking temporal proper-ties of software with boolean programs[C]// Workshop on Advances in Verication(with CAV 2000). 2000

[8] Chen Yan, He Yujing, Xie Fei, et al. Automatic Abstraction Refinement for Generalized Symbolic Trajectory Evaluation[C]// Proceedings-Formal Methods in Computer Aided Design, FMCAD 2007. 2007:111-118

(上接第 211 页)

本权重分为两种情况：当训练样本明显地分为若干目标类时，应当首先确定这些目标类的权重比例，然后把每个样本的权重调整限定在类内完成。比如 3.3 节中涉及到的 Iris 样本集，可首先将其中包含的 3 个目标类的权重比例设为 $a:b:c$ ($a+b+c=1$)，并使得这个比例在每轮循环中都得以保持，然后在每一轮的循环中，各预测目标类按照该权重比例，在目标类内部更新样本权重，进行标准化[9]；当训练样本不能够明显地分为若干目标类时，可以根据训练样本的数量设置单个样本的权重上限和权重下限，将单个样本的权重限定在一定区间内，避免最终得到的分类器对样本产生过学习或偏见现象。

3. 尽量增加训练样本数目，样本选取尽量离散化。从上述对连续型 Adaboost 算法的探究可知，算法最重要的基础是概率理论，由训练结果得到检测结果的过程包含一个先验概率问题，即由训练样本在各个分区上的着落概率决定检测目标值的分类结果。因此为避免随机性，应当尽量增加训练样本的数目，以满足算法的概率理论基础。同时，为扩大所得分类器的适用范围，提高所得分类系统的鲁棒性，样本的选取应当尽量离散化，如人脸样本库中的样本应当来源于不同人种、不同年龄段，带有不同表情，在不同的光照条件下获得。只有样本离散化达到一定程度同时样本数目达到一定水平的样本库所训练出来的分类器，其性能才有一定保证。

参 考 文 献

[1] Schapire R E. The strength of weak learnablhty[J]. Machine Learning, 1990, 5(2):197-227

[2] Freund Y. Boosting a weak learning algorithm by majority[J]. Information and computation, 1995, 141(2):256-285

[3] Song Chunlei, Wang Long. Learning Theory and Robust Control [J]. Journal of Control Theory and Application, 2000, 17(5): 633-636

[4] Valiant L G. A Theory of the Learnable[J]. Communications of the ACM, 1984, 27(11):1134-1142

[5] Zhao Nan. Face Detection Based on Adaboost[D]. Peiking University, 2005

[6] Freund Y. Schapire R E. Experiments with a New Boosting Algorithm[C]// Proc. the 13th Conf. Machine Learning. San Francisco: M organ Kaufmann, 1996:148-156

[7] Schapire R E, Singer Y. Improved boosting algorithms using confidence-rated predictions [J]. Machine Learning, 1999, 37 (3):297-336

[8] Quinlan J R. Bagging, bosting. and C4. 5[C]// Proceedings of the Thirteenth National Conference on Artificial Intelligence. 1996:725-730

[9] Li Bin. Enhancing Method for Adaboost[J]. Mini-Micro Systems, 2004, 25(5)