UIO 序列优化搜索算法的研究

孙海平, 高明伦
(合肥工业大学微电子设计研究所, 安徽合肥 230009)

摘 要: UIO 序列是对有限状态机进行功能测试的有效手段, 在 VLSI、通信协议等时序系统中有很强的实际应用背景. 本文基于可区分状态组这一概念设计了一个搜索算法, 进一步利用搜索信息建立了一个基于“小于”关系的启发策略, 有效的剪枝策略的设计将尽可能消除没有意义的搜索分枝, 新设计出的多路 OPEN/CLOSED 表存储机制也加快了相关的判别、处理过程. 根据实验结果, 分析了优化措施对于改进了搜索过程、减少搜索信息的产生、提高搜索速度有显著的贡献. 该算法与以往的算法相比, 在时间复杂度和空间复杂度两方面都得到了很大改进.

关键词: 有限状态机; UIO 序列; 启发式搜索; 优化策略; 功能测试
中图分类号: TN606 文献标识码: A 文章编号: 0372-2112 (2002) 05-0667-05

Study on Optimized Search Algorithm for UIO Sequences Generation

SUN Hai-ping, GAO Ming-lun
(*The Institute of VLSI Design, Hefei University of Technology, Hefei, Anhui 230009, China*)

Abstract: Unique Input/Output (UIO) sequence is an efficient method to perform functional test of Finite State Machine (FSM), which arises in many applications, such as VLSI designs and communication protocols, etc. A heuristic algorithm based on Distinguished State Group (DSG) is described to generate UIO sequences. The optimizing methods include a hellristic strategy based on a specific 'less' relation, several pruning strategies, and a novel access mechanism of multiple OPEN/CLOSED lists, and these methods eliminate onσ sense nodes and branches to a great extent. According to the experimental results, the practicability of all the measures is analyzed. Compared with brute algorithms, the optimized one is improved in terms of time and space complexities.

Key words: finite state machine (FSM); unique input/output sequence; heuristic search; optimization strategy; functional test

1 引言

随着系统功能和结构复杂性的发展, 结构测试(白盒测试)面临着组合爆炸的窘境, 势必将被功能测试等黑盒测试所取代. 而 UIO(Unique Input/Output) 序列就是有限状态机输出的特征序列, 可以从它的设计规格说明书(specification)中直接计算出 UIO 序列、进而生成所需的测试序列, 而无需了解该有限状态机的具体实现, 因而可以采用黑箱原理实施行之有效的功能测试和协议一致性测试[1~3].

功能测试的引入极大地降低了测试生成的计算量. 但由于计算 UIO 序列仍是 NP 难题, 问题本身所固有的属性导致了随着有限状态机的状态数、输入向量位数等参数的增大, 必然引起 UIO 序列的求解复杂度呈指数性增长, 仍然会面临组合爆炸的现象. 文献[1, 4~6]讨论了不同的求解思路和方法.

为了更好的实现 UIO 序列的求解过程, 有必要以恰当的数据结构描述有限状态机和算法中涉及的各项数据、并采用有效的搜索方法和存储机制来加速问题的求解.

2 基础知识

2.1 有限状态机的形式化描述

定义 1 有限状态机 FSM 是个五元组$(I, O, S, \delta, \lambda)$. 其中 I 是输入向量空间, O 是输出向量空间, S 是状态集, $\delta: S \times I^{\rightarrow} S$ 是状态变迁函数, $\lambda: S \times I^{\rightarrow} O$ 是输出函数. I、O 和 S 都是非空有限集.

定义 2 输入向量$(x_{m-1}, x_{m-2}, \cdots, x_0)$是 m 位二值向量, 其中每个 $x_i \in \{0, 1\}$.

根据定义 1 和定义 2 可知, 对于输入向量空间中的每个输入向量都对应 $|S|$ 个状态变迁项. 整个有限状态机存在着 $2^m \times |S|$ 个状态变迁项. 如此庞大的数目会导致搜索效率极低. 为此可以引入计算机辅助逻辑设计理论中广泛使用的多维体[7, 8]的概念, 在输入标记的基础上把源状态、目的状态及输出向量都完全相同的 $2^k (0 \le k < m)$ 个变迁项简化成一个变迁项, 从而减少产生的搜索分枝数, 加快搜索速度.

定义 3 输入标记$(y_{m-1}, y_{m-2}, \cdots, y_0)$是 m 位向量,

收稿日期: 2001-04-19; 修回日期: 2002-01-07
基金项目: 国家自然科学基金(No. 69876010); 国家教委高等学校博士学科专项科研基金(No. 98035901)

$\forall y_i \in \{0, 1, -\}$; 也是输入向量集, $(y_{m-1}, y_{m-2}, \cdots, y_0) = \{(x_{m-1}, x_{m-2}, \cdots, x_0) \mid (x_i = y_i \neq \text{`}-\text{'}) \vee (x_i \in \{0, 1\}) \wedge (y_i = \text{`}-\text{'}), i = 0, 1, \cdots, m-1\}$. 其中,"-"表示数字逻辑中的无关项.

输入向量既可以视为是单个的二值向量, 也可以拓广地视为是只含有该二值向量的独点集. 这样集合的包含、交、并、属于等集合运算都可以应用到输入向量和输入标记上.

2.2 UIO 序列的定义

定义 **4** 状态 s 在输入向量序列 x 的作用下的输出向量序列记作 $\lambda(s, x)$.

定义 **5** 状态 s 的UIO 序列 x 是能唯一标志状态 s 的输入向量序列, 在该输入向量序列作用下状态 s 的输出向量序列有别于任意其它状态 $t(\forall t \in S \wedge t \neq s)$ 的输出向量序列, 即 $\lambda(t, x) \neq \lambda(s, x)$.

求解UIO 序列时会遇到以下客观限制:

(1)只有最简 FSM 才可以求解 UIO 序列[1]. 如果 FSM 中存在两个等价状态, 则这两个状态彼此一定不能够通过输出向量序列来加以区分, 它们的 UIO 序列都不存在.

(2) 有些 FSM 的部分状态可能不存在 UIO 序列[2], 求解算法应该有能力判别UIO 序列的存在性, 在 UIO 序列不存在的情况下应有能力终止求解过程.

(3) 从测试时间等角度考虑, 要求 UIO 序列尽可能短.

3 搜索的求解思路

从UIO 序列的定义及对其长度的客观要求可以看出, 最直观的方法是试探所有可能的输入向量序列, 通过不断加长序列长度可以求得最短UIO 序列. 实际上这是穷举法. 在试探长度 $k+1$ 的序列的时候, 如果能够直接利用到长度为 k 的序列的计算结果, 则成为状态空间搜索的求解方法.

3.1 状态空间的设计

假定状态 s 的UIO 序列是 $x_1x_2\cdots x_i\cdots x_n$. 根据定义 5 知, 该序列作用下与状态 s 具有相同输出序列的状态所构成的集合必然是空集 ϕ.

x_1 作用下, $S-\{s\}$ 中部分状态的输出与状态 s 的输出相同. 这些状态在 x_1 作用下迁移到的目的状态构成的集合是 $\{\delta(s_t, x_1) \mid s_t \in S-\{s\} \wedge \lambda(s_t, x_1) = \lambda(s, x_1)\}$, 记做 TBD_1. 记 $s_{jump,1} = \delta(s, x_1)$.

同理在 x_2 作用下, TBD_1 中部分状态的输出与状态 $s_{jump,1}$ 的输出相同. 这些状态在 x_2 作用下迁移到的目的状态构成的集合记 $\{\delta(s_t, x_2) \mid s_t \in TBD_2 \wedge \lambda(s_t, x_2) = \lambda(s_{jump,1}, x_2)\}$, 记做 TBD_2. 记 $s_{jump,2} = \delta(s_{jump,1}, x_2)$.

如此迭代, 最终必得到集合 $TBD_r = \phi$. 求解 UIO 序列就是要通过搜索找出能使 TBD_r 成为空集 ϕ 的最短序列 $x_1x_2\cdots x_n$. 在此过程中 TBD_k 表示在长度为 k 的输入序列作用下待区分状态集, $|S|-1 = |TBD_0| \geq |TBD_1| \geq \cdots \geq |TBD_n| = 0$ 成立. 搜索过程就是 TBD 集不断收敛的过程.

搜索过程中每项信息叫做节点. 据以上分析可知, 每个节点可以表示成一个三元组 $\langle s_{jump}, TBD, Sqnc_{frag}\rangle$, 其中 $Sqnc_{frag}$ 表示得到此三元组所使用的输入序列. 为简化描述, 下文在引用元组中某元素的时候采用面向对象技术中的路径式命名方式, 即"元组名. 元素名". 例如三元组 $Node = \langle s_{jump}, TBD, Sqnc_{frag}\rangle$ 中的 s_{jump} 元素用 $Node.s_{jump}$ 来表示.

3.2 搜索的一般过程

人工智能中对于状态空间的通用搜索方法有着一般化的描述[9]. 现结合求解状态 的 UIO 序列描述如下:

设立 *OPEN* 表和 *CLOSED* 表. *OPEN* 表中存放尚未处理的数据, 而处理过的数据存放在 *CLOSED* 表中.

初始时, *CLOSED* 表是空表, *OPEN* 表中只有一个元素 $\langle s, S-\{s\}, ''\rangle$, 其中 $''$ 表示空序列.

OPEN 表非空的情况下, 总是对 *OPEN* 表首元素 *Node* 做以下扩展处理:

(1) 把节点 *Node* 从 *OPEN* 表中移出, 放入 *CLOSED* 表的尾部;

(2)找出节点 *Node* 在不同输入向量集(即输入标记) 作用下所能产生的所有新节点 $Node_{new}$, 并把有实际意义的新节点添加到 *OPEN* 表中. 如果产生的某个新节点的 *TBD* 是空集, 则说明已搜索出状态 s 的 UIO 序列, 计算过程结束.

4 可区分状态组及其求解算法

上文介绍的搜索过程中, 新节点的产生依赖于可区分状态组(规则)这一概念. 可区分状态组是定义在具有相同性质的输入符号集(即输入标记)的基础上的, 是通过预分析产生知识(规则)合并相同分枝的技术, 避免了对集合中的每一个输入符号分别判别求解, 从而加快了搜索过程.

4.1 可区分状态组

定义 **6** 状态 s 的可区分状态组 *DSG*(规则) 定义在状态组 $SG(SG \subseteq S-\{s\})$ 之上. $\exists\, i_{DV}$, 使 SG 满足以下两个条件: (1) $\forall s_i \in SG, \lambda(s_i, i_{DV}) \neq \lambda(S, i_{DV})$; (2) $\forall s_i \notin SG, \lambda(s_i, i_{DV}) = \lambda(S, i_{DV})$. 其中 i_{DV} 叫做该 *DSG* 的区分向量. 该 *DSG*(规则) 的所有区分向量构成了区分向量集 *DVS*, 记 $DSG = \langle s, SG, DVS\rangle$.

定义 **7** 状态 s 的所有可区分状态组(规则) 构成的集合记做 $SDSG(s)$.

定义 **8** 节点 $\langle s_{jump}, TBD, Sqnc_{frag}\rangle$ 在 $DSG \in SDSG(s_{jump})$ 作用下, 可以得到新节点 $\langle \delta(s_{jump}, DSG.DVS), \{\delta(s_i, DSG.DVS) \mid s_i \in TBD-DSG.SG\}, Sqnc_{frag} \bullet DSG.DVS\rangle$, 其中 $\bullet$ 表示向量序列的拼接运算.

4.2 可区分状态组的求解

状态 s 在不同的 *DVS* 作用下可有不同的可区分状态组(规则). 由于状态 s 上各变迁的输入标记客观上构成了对整个输入向量空间的划分 $\pi_I(s)$, 因而在划分出的各个子空间上分别求解可以得到该状态全部的可区分状态组(规则). 由于划分 $\pi_I(s)$ 自身具有各元素(即输入标记) 互不相交的性质, 所以在 $\pi_I(s)$ 划分出的各个子空间上求解出的 *DSG* 不会重复.

文献 1 介绍了求解状态 s 关于变迁 (i, s, d, o) 的所有可区分状态组的过程需分成两步. 现修正其中的错误, 介绍求解

过程如下:

首先, 求出那些在输入标记 i 作用下与状态 s 一定可以区分的状态构成的集合 $ADSG$, 以及一定不可以区分的状态构成的集合 $AUSG$. 需要注意的是在判别状态 s' ($s' \neq s$) 是否属于集合 $ADSG$ 和 $AUSG$ 的时候, 要考虑到 i 可能和 s' 上的多个变迁的输入标记相交. 只有在每个与 i 相交的输入标记作用下都可以(或都不可以)使状态 s' 的输出有别于状态 s 的输出, s' 才属于集合 $ADSG$(或 $AUSG$).

既不属于集合 $ADSG$ 也不属于集合 $AUSG$ 的那些状态只在输入标记 i 的某个真子集作用下输出才有别于状态 s. 所以其次的工作是确定使这些状态有别于状态 s 的 DVS, 进而确定出每个 DSG. 为了减少 DSG 规则的数目已减少搜索分枝, 在确定 DSG 是应该尽可能扩大可区分状态的数目.

但是, $S - \{s\} - ADSG - AUSG$ 是空集表明在 $ADSG$ 不为空集的情况下 i 本身就是区分向量, 这时有唯一的一个 DSG 规则$\langle \delta(s, i), DSG, i\rangle$.

5 搜索的优化

可区分状态组把具有相同性质的大量输入向量汇聚成少数的区分向量集 DVS, 减少了搜索过程中新节点的计算量. 但是新节点插入到 $OPEN$ 表中的位置的不同会产生不同的求解效果. 深度搜索把新节点插入 $OPEN$ 表的头部; 而宽度搜索把新节点插入到 $OPEN$ 表的尾部. 这些方法都是盲目搜索的方法.

下面介绍在本启发式搜索算法中用到的优化搜索措施.

5.1 启发策略

由于搜索过程总是对 $OPEN$ 表首元素加以处理, 因而 $OPEN$ 表中元素的次序直接影响搜索效果. 由于: (1)搜索过程是个不断收敛的过程, UIO 序列求出的时候 TBD 集已演变成空集. 所以 TBD 集合愈小, 意味着愈接近最终求解目标. (2)同时也希望搜索出的UIO 序列最短. 所以 $OPEN$ 表中序的定义如下:

定义 **8**　$\{Node = \langle s_{jump}, TBD, Sqnc_{frag}\rangle\}$ 上小于关系 $\prec$: $Node_1 \prec Node_2$, 当且仅当$|Node_1.Sqnc_{frag}| < |Node_2.Sqnc_{frag}| \vee |Node_1.Sqnc_{frag}| = |Node_2.Sqnc_{frag}| \wedge |Node_1.TBD| < |Node_2.TBD|$ 成立.

新节点插入到 $OPEN$ 表中时总是遵循"小节点在前、大节点在后"的原则可加速搜索过程.

5.2 剪枝策略

5.2.1　删除落后节点　定义 **9**　对于节点 $Node_1$ 和 $Node_2$, 如果 $Node_1.s_{jump} = Node_2.s_{jump}$ 成立且 $Node_1.TBD \subseteq Node_2.TBD$ 成立, 称在搜索进度上 $Node_1$ 领先于 $Node_2$, 或称 $Node_2$ 落后于 $Node_1$. 否则称在搜索进度上 $Node_1$ 与 $Node_2$ 无关.

新节点插入到 $OPEN$ 表中时, 有必要检查新节点是否落后于 $OPEN$ 表和 $CLOSED$ 表中的节点, 如果落后则没有必要把新节点插入到 $OPEN$ 表中; 同时也有必要把 $OPEN$ 表中落后于新节点的那些节点都删去, 从而尽可能减少待搜索节点、加快搜索进度.

5.2.2　合并状态截枝　状态 s 在输入向量序列 x 作用下 $\lambda(s, x) \in TBD(s, x)$, 说明 $\exists\, t \in S - \{s\} \wedge \delta(s, x) = \delta(t, x) \wedge \lambda(s, x) = \lambda(t, x)$, 即状态 s 和状态 t 在输入向量序列 x 作用下不可区分, 在此情况下无论怎样延展该序列都无法求出 UIO 序列. 所以扩展出的节点$\langle s_{jump}, TBD, Sqnc_{frag}\rangle$一旦满足条件 $s_{jump} \in TBD$, 就没有必要将该节点放入 $OPEN$ 表中等待扩展.

5.2.3　避免重复使用同一个 DSG 规则　如果求解状态 s 的 UIO 序列过程中某一输入序列使状态 s 再次变迁到状态 s_{jump}, 则以往在 s_{jump} 时所使用过的 DSG 规则没必要再次使用. 因为每次 $TBD_{new} = TBD - DSG.SG$, 一个 DSG 规则在使用过一次之后, 不论再使用多少次, 都不会减少待区分状态. "删除落后节点"是一种能够实现避免重复使用同一个 DSG 规则的方法.

5.3 多路 OPEN/CLOSED 表技术

启发策略是把新节点插入 $OPEN$ 表时, 使 $OPEN$ 表中的节点保持特定的大小; 剪枝策略判别新节点是否是"已落后"节点、$OPEN$ 表中哪些节点是落后于新节点的. 这一切都依赖于对 $OPEN$ 表和 $CLOSED$ 的遍历. 尤其是剪枝策略中遍历的共同特点是处理节点 $Node_{new}$ 的遍历过程中只判别与特定状态 $Node.s_{jump}$ 相关的节点.

因此使用多路 $OPEN/CLOSED$ 表存储结构替代单一的 $OPEN/CLOSED$ 表结构, 状态集 S 中的每个状态都拥有自己的 $OPEN/CLOSED$ 表, 节点 $Node$ 存放在 $OPEN[Node.s_{jump}]$ 或 $CLOSED[Node.s_{jump}]$ 中.

采用这样的多路表存储结构, 不仅缩短了应用剪枝策略所需的遍历时间, 而且也缩短了向 $OPEN$ 表中插入新节点所需的时间.

5.4 实验结果

针对文献[10]中给出的国际上的标准测试电路, 笔者使用 Delphi 编程比较了基于可区分状态组这一概念的盲目搜索和本文介绍的优化搜索算法. 两种算法求出的结果是相同的, 但是运算量相差很大. 表 1 列出了标准有限状态机的各项参数. 表 2 对比了盲目搜索算法和启发式搜索算法的计算量, 不仅统计了两种算法中生成的节点数、扩展的节点数等数据, 而且给出了启发式搜索过程中各剪枝策略直接剪除的节点数. 从表 2 可以看出, 除了 sand 效果不显著外, 其余的电路用启发式算法都得到了很大的成功.

表 1　标准 FSM 的各项参数

标准 FSM	S	T	I	DSG	UIO	L_{max}
s208	18	153	11	153	18	8
s298	218	1096	3	1096	128	12
s420	18	137	19	137	18	8
scf	121	166	27	165	85	8
tbk	32	1569	6	608	0	
train11	11	25	2	21	0	
sand	32	184	11	184	32	17
styr	30	166	9	167	26	4

S: 状态数; T: 变迁数; I: 输入向量位数; DSG: 可区分状态组数目; UIO: 有 UIO 序列的状态数; L_{max}: 各状态最短 UIO 序列长度的最大长度.

表2 算法计算量对比

标准 FSM	盲目搜索			启发式搜索			剪枝策略直接剪除的节点数		
	G	*E*	*A*	*G*	*E*	*A*	*P*1	*P*2	*P*3
s208	2184	268	347	535	74	100	15	25	316
s298	306844	61302	77494	118370	23753	30562	757	6318	21756
s420	1924	271	355	468	74	102	17	25	267
scf	6402	4581	4698	495	356	411	5	16	46
tbk	87248	4592	4560	9060	474	490	48	336	2646
train11	383	179	168	44	22	11	0	2	22
sand	1673	218	229	1660	217	225	1	0	58
styr	2009	224	300	383	53	87	14	21	103

G: 搜索生成的节点数; *E*: 被扩展的节点数; *A*: 添加到 *OPEN* 表中的节点数; *P*1: 删除 *OPEN* 表中的落后节点; *P*2: 删除落后新节点; *P*3: 合并状态截枝

通过实验, 我们发现“合并状态截枝”策略直接剪除的节点最多. 其次是“删除落后节点”策略, 相对而言“删除 *OPEN* 表中的落后节点”这一优化策略使用的次数较少. 事实上, 直接剪除的节点多并不直接意味着效果就一定好. 关键在于要尽可能早地剪除那些没有希望求出结果的节点, 避免从那些没有“希望”的节点上扩展出更多的没有希望求出结果的节点. 优化算法在状态机 sand 上之所以效果不显著, 就在于剪枝策略在该电路上不能尽早发挥作用.

而其余的优化措施, 如“有序插入”、“多路 *OPEN*/*CLOSED* 表存储技术”等, 虽然直接剪除那些没有“希望”的节点, 但是分别能协助算法调整节点在 *OPEN* 表中的位置和加快查找、存取等方面的速度, 缩短了搜索求解所需的时间.

6 结束语

启发、剪枝等都是人工智能中常用的手段[9]. 本文在 UIO 序列生成这一特定应用领域, 基于可区分状态组基础上这一概念的启发式搜索算法, 合理地设计了与其性质相适应的启发策略和剪枝策略, 提出了存储搜索信息的新机制, 这一切都极大地改进了搜索效果.

致谢: 本文有关工作得到了电子科技大学 CAT 室陈光教授的指导和帮助, 在此表示感谢.

附录 **1**

算法 求解状态 s 关于变迁(i, s, d, o)的所有可区分状态组

输入:

变迁表$\{T_i\}: i=1, 2, \cdots, E$

目标变迁(i, s, d, o)

输出:

状态 s 关于变迁(i, s, d, o)的所有可区分状态组 $DSG=\langle s_{jump}, SG, DVS\rangle$

使用到的函数:

$$T(s_t, i)=\{(i_p, s_p, d_p, o_p) \mid s_p=s_t, i_p \cap i \neq \phi\}$$

算法过程:

```
//先确定完全可区分的状态组 ADSG 和完全不可区分的状态组 AUSG
SU= S- {s}, ADSG= AUSG= φ, k= 0
for s_u ∈ SU do {
  ST= T(s_u, i)
  if o ∉ {o_u | (i_u, s_u, d_u, o_u) ∈ ST} then do
    ADSG= ADSG+ {s_u}
  else
  if {o_u | (i_u, s_u, d_u, o_u) ∈ ST}= {o} then do
    AUSG= AUSG+ {s_u}
}
//根据部分可区分的状态决定各个可区分状态组及其区分向量
SU= SU- ADSG- AUSG, SU_t= SU
if EMPTY(SU_t) then do
  if not EMPTY(ADSG) then do {
    DSG_1. s_jump= δ(s, i), DSG_1. SG= ADSG, DSG_1. DVS= i
  }
else
  while not EMPTY(SU_t) do {
    k= k+ 1, DSG_k. s_jump= δ(s, i), DSG_k. SG= ADSG
    s_t= SU_t 中的第一个元素
    SU_t= SU_t- {s_t}, DSG_k. SG= DSG_k. SG+ {s_t}
    i_t= ⋃_{(i_p, s_p, d_p, o_p) ∈ T(s_t, i)} (i ∩ i_p), DSG_k. DVS= i_t
    //尽可能扩大 DSG
    for s_u ∈ SU- {s_t} do {
      ST= {(i_p, s_p, d_p, o_p) | (i_p, s_p, d_p, o_p) ∈ T(s_u, i), o_p ≠ o}
      i_r= ⋃_{(i_p, s_p, d_p, o_p) ∈ ST} (i_t ∩ i_p)
      if i_r ≠ φ then do {
        DSG_k. SG= DSG_k. SG+ {s_u}, DSG_k. DVS= i_r
        SU_t= SU_t- {s_u}, i_t= i_r
      }
    }
  }
```

附录 **2**

算法 UIO 序列的启发式搜索

说明: 为简化算法描述, 避免重复使用同一个 *DSG* 规则没有使用额外的数据结构, 而是通过落后节点的判别来实现的.

输入:

变迁表 $\{T_i\}$, $i = 1, 2, \cdots, E$

所有状态的可区分状态组集 $SDSG[s_{tar}] = \{DSG_i = < s_{tar}, SG, DVS > \}$, $s_{tar} \in S$

目标状态 s

输出:

状态 s 的UIO 序列 *UIOS* 及UIO 序列的存在标志 *Found*

算法过程:

```
把< s, S- {s},'' > 放入 OPEN 表中
Found= FALSE, UIOS = ''
while not Found and not EMPTY({FIRST(OPEN[s_i]) | s_i ∈ S}) do {
    把 OPEN 表中的第一个节点 Node 从 OPEN 表中移出, 并放入 CLOSED 表中
    s_t = Node.s_jump
    for DSG ∈ SDSG[s_t] do {
        TBD_next = {δ(s_j, DSG.DVS) | s_j ∈ Node.TBD- DSG.SG}
        s_newjump = δ(s_t, DSG.DVS)
        Node_new = 〈s_newjump, TBD_next, Node.Sqnc_frag • DSG.DVS〉
        if {Node_t | Node_t ∈ OPEN[s_newjump] ∪ CLOSED[s_newjump] ∧
                Node_t.TBD ⊆ TBD_new} = Φ then do{
            for Node_i ∈ OPEN[s_newjump] do
                if TBD_next ⊂ Node_i.TBD then do
                    把 Node_i 从 OPEN[s_newjump] 表中删去
            把 Node_new 按照启发优先序插入到 OPEN[s_newjump] 表中
            if TBD_next = Φ then do{
                Found= TRUE
                UIOS= Node_new.Sqnc_frag
            }
        }
    }
}
```

参考文献:

[1] Dechang Sun, Bapiraju Vinnakota, Wanli Jiang. Fast state verification [A]. 35th Design Automation Conference [C]. 1998. 619- 624.

[2] D Lee, M Yannakakis. Principles and methods of testing finite state machines: A survey [J]. Proceedings of IEEE, 1996, 84(8): 1090- 1122.

[3] D Lee, M Yannakakis. Testing finite state machines: state identification and verification [J]. IEEE Transactions on Computers, 1994, 43(3): 306- 320.

[4] D Schin, Y- N Shen, F Lombardi. An approach for UIO generation for FSM verification and validation [A]. Proceedings of ISCAS [C]. 1994, 4: 303- 306.

[5] A V Aho, A T Dahbura, D Lee, et al. An optimization technique for protocol conformance test generation based on UIO sequences and rural Chinese postman tours [J]. IEEE Transactions on Communications, 1991, 39(3): 1604- 1615.

[6] 曾成碧, 陈光　. 时序系统的状态组区别序列测试方法 [J]. 微电子学, 2000, 30(3): 188- 192.

[7] 刘明业. 计算机辅助逻辑设计理论 [M]. 北京: 科学出版社, 1996: 3- 7.

[8] M E Ulug, B A Bowen. A unified theory of the algebraic topological methods for the synthesis of switching systems [J]. IEEE Transactions on Computers, 1974, G- 23 (3): 255- 267.

[9] 傅京孙, 蔡自兴, 徐光　. 人工智能及其应用 [M]. 北京: 清华大学出版社, 1987: 19- 59.

[10] NCSU Collaborative Benchmarking Laboratory. LGSynth'91 Benchmark Information [EB/OL]. http://www.cbl.ncsu.edu/CBL_Docs/lgs91.html, 1997-03-25.

作者简介:

孙海平　男, 1973 年 3 月出生于上海, 1996 年毕业于合肥工业大学, 获计算机应用专业学士学位, 1997 年开始在该校攻读计算机应用专业硕士学位, 1999 年转入该校微电子设计研究所攻读博士学位, 主要研究领域: VLSI 设计方法, 计算机体系结构, 计算机算法, Email 地址: hp. sun@263. net.

高明伦　男, 1945 年 7 月出生于北京, 教授, 博士生导师, 1968 年本科毕业于清华大学无线电系, 1981 年于合肥工业大学电气工程系获硕士学位, 1991 年于(美)代顿大学电气系获博士学位, 研究领域: VLSI 设计方法, Email 地址: gaopan@mail.hf.ah.cn