

论有穷状态验证方法的局限性^{*}

葛 玮 龚晓庆 郝克刚

(西北大学信息科学与技术学院软件工程研究所 西安 710069)

摘 要 程序有穷状态验证方法是介于程序验证和程序测试之间的一种方法,一方面它如同程序验证一样可以证明某程序具有某些要求的性质,或找出反例证明该程序不具有所要求的性质。另一方面它又不像程序验证那样复杂,要求验证人员具有较高的形式化推理的专业理论和数学水平。但是,现有的有穷状态验证方法有很大的局限性,它要求所论证的性质是有穷自动机所接受的事件序列的集合,或等价地说该性质能表示成为正则表达式。众所周知,有穷自动机所能接受的语言类,按 Chomsky 字的集合的分类是很小的类。本文讨论了这种局限性,并尝试突破只能使用有穷自动机的限制,提出了一种新的验证方法——有穷路径验证法。在这种方法中,所论证的性质表示可以推广到使用任何一类自动机。作为代价,描写系统的模型限制是无环的。对于有环的描写系统的模型,本文提出了一种称之为“有穷路径测试”的方法。同一般的程序测试一样,用这种方法通过测试不能正面地验证程序的正确性,可是如果通不过测试,则能帮你发现反例,找出程序的错误。与一般的程序测试不同的是这里的测试是相对于模型的路径,而不执行实际的程序。

关键词 程序验证,程序测试,有穷自动机,有穷状态验证,有穷路径验证

Limitations of the Finite State Verification Approach

GE Wei GONG Xiao-Qing HAO Ke-Gang

(Software Engineering Institute, Northwest University, Xi'an 710069)

Abstract Finite state verification for programs is an approach between theorem proving and testing. On the one hand, it can prove that a program has some required properties, or find counterexamples to prove the program does not have these properties. On the other hand, it is not as complex as theorem proving which needs more special theory on formalized reasoning and more mathematical knowledge. However, there are some limitations on existing finite state verification approach. The properties being proved must be the sets of event sequences which can be accepted by finite state automata, or equivalently, these properties can be represented by regular expressions. As we all know that the set of languages accepted by the finite automata is the smallest set according to Chomsky's classification. In this paper, the limitation of using only finite automata is discussed, and as an attempt to break through this limitation, a new approach, finite path verification, is presented here. By this new approach, the properties being proved can be represented by automata of any kinds. The model describing system is restricted to be acyclic at a cost. An approach named finite path testing is also presented for the cyclic system model in this paper. The correctness of a program cannot be validated directly by this approach as by other common software testing. However, if the testing cannot be passed, counterexamples then can be found to reveal the faults in the program. The testing here is different from common testing for it is not to execute the actual program, but testing according to paths of the model.

Keywords Software verification, Software testing, Finite automaton, Finite state verification, Finite path verification

1 引言

程序有穷状态验证方法是介于程序验证和程序测试之间的一种方法,一方面它如同程序验证一样可以证明某程序具有用户定义的某些性质,或找出反例证明该程序不具有所要求的性质。另一方面它又不像程序验证那样复杂,要求验证人员具有较高的形式化推理的专业理论和数学水平^[1~5]。但是,有穷状态验证方法有很大的局限性,它要求所论证的性质是由有穷自动机所接受的事件序列的集合,或等价地说该性质能表示成为正则表达式。众所周知,有穷自动机所能接受的语言类,按 Chomsky 字的集合的分类是很小的类,比它大的类还有上下文无关语言、上下文有感语言和图灵机所接受递归可枚举集等。本文讨论了这种局限性。尝试突破程序有穷验证方法局限性,提出了一种新的方法,称为“有穷路径验证法”。在这种方法中,表示所论证的性质的自动机可以推广到使用任何一类自动机,如下推自动机、线性有界自动机甚至

图灵机等。作为代价,限制描写系统的模型是无环的。对于有环的 CFGs 描写系统的模型,本文提出了一种称之为“有穷路径测试”的方法。同一般的程序测试一样,用这种方法通过测试不能正面地验证程序的正确性,可是如果通不过测试,则能帮你发现反例,找出程序的错误。与一般的程序测试不同的是这里的测试是相对于性质和模型的路径,而不执行实际的程序。

本文首先简单介绍程序有穷状态验证方法和分析它的局限性,然后给出尝试突破程序有穷验证方法局限性的方法——有穷路径验证法,最后讨论有穷路径测试法。

2 程序有穷状态验证方法及其局限性分析

FLAVERS (Flow Analysis for VERification of Systems) 是美国麻省 Amherst 大学先进软件工程研究实验室研发的支持有穷状态验证方法的工具^[1,3]。它使用数据流分析技术验证并行系统的基于事件的行为性质。分析人员首先定

^{*} 本文受国家 863 计划“软件评测平台的研究与应用”课题(课题编号:2004AA115090)的资助。葛 玮 副教授,CCF 高级会员,研究领域:软件工程、软件测试;龚晓庆 博士,研究领域:软件工程;郝克刚 教授,博士生导师,研究领域:软件工程、软件理论、形式化方法。

义一组所论证的事件和以这些事件的出现顺序的形式规约并行系统的性质,然后使用 FLAVERS 自动创建系统的模型,明确地表示并行任务间的通讯、同步和事件的顺序,而不必穷

举整个状态空间。FLAVERS 的核心是一个多项式时间的流分析算法,确定系统是否满足给定的性质。如果不满足,则给出违反此性质的反例。

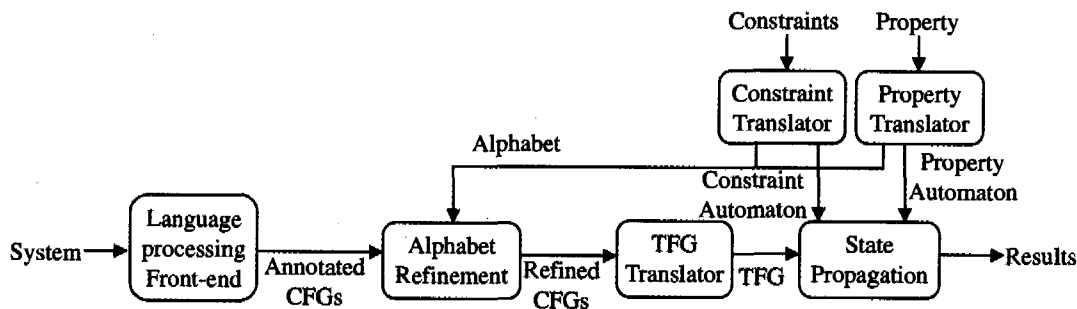


图1 FLAVERS 结构

FLAVERS 的工作过程如图1所示。分析人员用带量词(这里用的量词有全称量词 all 和非存在量词 no)的正规表达式 QRE(quantified regular expression)或有穷状态自动机 FSA (finite-state automaton)来表达性质和约束。所关心的事件是由性质和约束得到的,它决定了分析问题的字母表。为了论述系统所有可能的执行,FLAVERS 构建了一种能简洁地表示所有对应于系统执行中相关事件的序列的模型—追踪流图 TFG (trace-flow graph)。首先根据所分析的系统的每个并行任务的程序,建立标注的控制流图 CFG (control-flow graph),然后按照字母表将其精化为精化控制流图 RCFG (refined CFG)。在按字母表精化后,把这些 RCFGs 组合起来,添加上反映任务间的同步和通信的节点和边,以及并行动作可能顺序的有向边,最后经过优化构成 TFG(注:这里用添加有向边的方法来描述并行动作的顺序,并不一定好。可考虑采用并行字的办法^[6,7])。从理论上讲,同 TFG 的路径相联系的事件序列决定了性质自动机的状态转移序列。FLAVERS 采用的状态传播算法为 TFG 每个节点确定出所有可能到达此节点的路径所对应的性质自动机到达的状态。这种数据流分析算法避免了穷举所有可能的系统事件序列,而是把他们压缩为等价类。这种算法的时间复杂度是 TFG 节点个数的多项式函数。如果算法分析的结果是肯定的结论,即与 TFG 的终止节点对应的所有状态都是有穷状态自动机的终止状态,则可以得出结论:系统具有所要求的性质。如果算法分析的结果是非肯定的结论,即与 TFG 的终止节点对应的所有状态并非都是有穷状态自动机的终止状态,则提供了系统可能不具有所要求的性质的反例的信息,或者说明系统的模型不够精确。我们注意 TFG 只是一种近似,它包含了所有可执行的相关事件的序列,但是也可能包含某些实际上不可能出现的序列。如果是这种不可能出现的序列导致了有穷状态自动机的非终止状态,则不能断定系统不具有该性质。在这种情况下需要引入约束,删除掉这些实际上不可能出现的序列,最后再判断能否得出肯定的分析结果。约束由有穷自动机定义。约束自动机中有一个状态称为违背状态。凡是到达违背状态的事件序列都是实际上不可能出现的序列,应从分析中删除。约束有各种类型,如变量、任务、环境和界面等类型的约束。FLAVERS 对约束采取了一种灵活的策略,允许用户定义各种类型的约束,以提高模型的精度。FLAVERS 目前提供自动化支持的仅有变量和任务两种类型的约束。FLAVERS 的验证过程经常表现为不断引入约束,逐步提高模型精度的逐步逼近的过程。

从以上的介绍可以看出,有穷状态验证方法有很大的局限性。首先,它所验证的是系统是否满足定义的行为性质(Behavioral Properties),而不是断定系统是否满足定义的规约(Specification)。性质和规约有很大的区别,性质是系统的部分的、某一个方面的特性,而规约才是系统应满足的完备的特性。其次,它要求所论证的性质表现为程序执行中某些关心的事件出现序列的集合,而且限定是能由有穷自动机所接受的事件序列表示的集合,或等价地说该性质能表示成为正规表达式。FLAVERS 所支持的有穷状态验证方法之所以要做这样的限制,主要是为了提高分析算法的有效性和可行性。FLAVERS 采用的状态传播算法为 TFG 每个节点确定出所有可能到达此节点的路径所对应的性质有穷自动机到达的状态。这种数据流分析算法避免了穷举所有可能的系统事件序列,而是把他们压缩为等价类。这种算法的时间复杂度是 TFG 节点个数的多项式函数。众所周知,有穷自动机所能接受的语言类,按 Chomsky 字的集合的分类是很小的类,比它大的类还有下推自动机所接受的上下文无关语言、线性有界自动机所接受上下文有感语言和图灵机所接受递归可枚举集等。于是就产生了问题:能否找出新的验证方法,突破这种关于有穷自动机的限制?下面是我们所作的尝试。

3 程序有穷路径验证方法

我们先从一个例子说起。假定我们编了一个程序,在程序中用了一个栈,要分别使用 put 和 get 调用这个栈进行“进栈”和“退栈”的动作。为了验证程序的正确性,我们想验证 put 和 get 调用的序列是合适的,即验证程序是否满足如下性质:put 和 get 的调用严格配对。也就是说,要求栈在程序开始和终止时空,get 必须在栈不空时调用,而且 put 和 get 的调用次数相同。例如下述序列就满足此性质:

put get put put put get get put put get get get

熟悉自动机理论的读者肯定知道,满足这样性质的序列集合是不可能用有穷自动机接受的。它是由如下文法生成的语言:

$S \Rightarrow \epsilon \mid \text{put } S \text{ get} \mid S S$

例如生成上述序列的过程如下:

$S \Rightarrow S S$

$\Rightarrow \text{put } S \text{ get } S$

$\Rightarrow \text{put get } S$

$\Rightarrow \text{put get put } S \text{ get}$

$\Rightarrow \text{put get put } S S \text{ get}$

$\Rightarrow \text{put get put put S get S get}$
 $\Rightarrow \text{put get put put put S get get S get}$
 $\Rightarrow \text{put get put put put get get S get}$
 $\Rightarrow \text{put get put put put get get put S get get}$
 $\Rightarrow \text{put get put put put get get put put S get get get}$
 $\Rightarrow \text{put get put put put get get put put get get get}$

接受这种语言的至少是下推自动机,可见用有穷验证方法是验证不了这种性质的。为了突破程序有穷验证方法局限性,我们尝试提出了一种新的方法,称为“有穷路径验证法”。在这种方法中,表示所论证的性质的自动机可以推广到使用任何一类自动机,如下推自动机、线性有界自动机甚至图灵机等。例如上面的例子,我们可以构造一个下推自动机来接受 put 和 get 的调用严格配对的性质。

关于系统执行的模型,仍有有穷验证的方法一样采用追踪流图 TFG。不过为了验证的有效性,我们限定组成 TFG 的描述各个并行任务的 RCFG 作为有向图是无环的。

定义 3.1 我们称一个有向图是基本有向图,如果它仅有一个始点和一个终点,而且对图中任何节点,都有从始点到此节点的路径和从此节点到终点的路径。所谓始点是指无入边的节点,终点是指无出边的节点。本文所论的有向图都指基本有向图。

对于无环的基本有向图,所有路径都是无重复节点的路径。

定理 3.1 对于基本有向图,从始点到终点的所有路径都是无重复节点路径的必要充分条件是有向图是无环的。

证明:必要性。假定有向图是有环的,考虑如下路径。从始点到此环的某个节点,再在此环中作若干次循环,然后再到终点。显然此路径有重复节点。

充分性。假定从始点到终点的某路径有重复节点,取这两个节点间的子路径,在图中必形成图中的一个环。证毕。

我们所提出的有穷路径验证方法与有穷状态验证方法一样,把描述各个并行任务的 RCFGs 组合起来,添加上反映任务间的同步和通信的节点和边,以及并行动作可能顺序的有向边,最后经过优化构成 TFG。这里需要指出的是,这里的 RCFGs 和 TFG 都是基本有向图,而且这些 RCFGs 的无环性并不能保证由它们组合起来 TFG 是无环的。为描述并行动作可能形成的顺序而添加的有向边可能在 TFG 形成环,但是实际上由此而生成的有重复节点的路径,在并行系统地执行中实际上是不存在的。所以从理论上讲,反映系统中相关的事件的执行序列的是 TFG 的无重复节点的路径。另外,我们还可证明如下定理成立。

定理 3.2 对于基本有向图,从始点到终点的所有不同的无重复节点的路径的个数是有穷的。

证明:对于由有穷节点组成的有向图,它的任何无重复节点的路径的长度(指路径中节点的个数,下同)肯定是有穷的(小于有向图节点个数)。所以,从始点到终点的所有不同的无重复节点的路径的个数是有穷的。

算法 3.1(求基本有向图中从始点到终点的所有无重复节点路径的算法) 对基本有向图 T 中所有节点 N_i ,求出从始点到达此点 N_i 所有可能的无重复节点的路径集合 $L(N_i)$ 。对于终点 Z , $L(Z)$ 即是 T 中从始点到终点的所有不同的无重复节点的路径的集合。

(1)算法开始时,令 T 中所有节点 N_i 上的所求的路径集合为空集合 $L(N_i) = \{\}$ 。

(2)令始点 S 的路径集合 $L(S) = \{S\}$ 。

(3)对图中所有的由有向边连接的节点 Q 和 H ,其中 Q

是前节点, H 是后节点,做如下动作。如果有路径 $\alpha \in L(Q)$ 满足条件:

①节点 H 不在 α 中出现,而且

②路径 αH 不包含在 H 的路径集合 $L(H)$ 中,

则将路径 αH 加入到 $L(H)$ 中,即令 $L(H) = L(H) \cup \{\alpha H\}$

(4)重复动作 3,直至所有的由有向边连接的节点 Q 和 H , $L(Q)$ 中没有满足上述条件的路径。

不难证明,上述算法在有穷步后即可终止,算法结束后 $L(Z)$ 即是 T 中从始点到终点的所有不同的无重复节点的路径的集合。

现在我们对有穷路径验证方法给以完整地概括说明。

(1)使用任何一类自动机(不限定是有穷状态自动机)所接受的语言来定义系统的性质,并构造相应的性质自动机。

(2)使用有穷自动机定义系统的约束(开始验证时可能没有约束)。

(3)根据性质自动机和约束自动机涉及的事件字母表,用无环的 RCFG 描述各个并行任务,把 RCFGs 组合起来构成 TFG 为并行系统的执行序列建模。

(4)用算法 3.1 求出 TFG 从始点到终点的所有不同的无重复节点的路径集合(有穷集),并在算法的执行过程中删掉那些导致进入约束自动机违背状态的路径。

(5)对此路径集合中的所有路径,用性质自动机来判断。如果所有路径皆被性质自动机所接受,则通过验证,证明系统满足此性质。

(6)如果有路径不被性质自动机所接受,同有穷状态验证一样有两种可能:

①增加约束,转到 2,提高模型的近似度,删掉不现实的路径,继续验证。

②如果该路径是现实的,则提供信息帮助找到系统违背性质的反例。

此方法的复杂度与系统的结构和性质的复杂度有很大关系,可以简单分析如下。算法 3.1 的大步骤与边的个数有关,相当于 $O(n^2)$,其中 n 表示节点个数,但是所产生的路径的个数尽管是有穷的,可能会在某些复杂的情况下超过 n 的多项式函数。至于性质自动机的运行复杂度则取决于自动机本身的构造。

从上述论述可以看出,为了保证路径的有穷性,有穷路径验证方法作了 RCFGs 是无环有向图的限定。在实际的应用中,常常出现 RCFG 有环的情况。对于用有环的 RCFGs 描写的系统模型,本文提出了一种称之为“路径测试”的方法。

4 程序路径测试方法

对于 RCFG 有环的情况,反映系统的执行序列有可能是模型 TFG 中有重复节点的路径,而且这种路径的个数一般是无穷的。所以,用穷尽所有路径的方法进行验证是不可能的。不过我们可以像一般的程序测试方法那样,用路径是否被性质自动机所接受来测试所关心的性质是否对此路径成立。当然,用这种方法通过某些路径的测试不能正面地验证程序的正确性。可是如果对某路径通不过测试,则能帮你发现反例,找出程序的错误。与一般的程序测试不同的是这里的测试是相对于所关心性质和使用模型的路径,而不是使用测试用例执行实际的程序。所以本方法的关键,就如同一般的程序测试方法中生成测试用例那样,是如何产生用以进行测试的路径。

定义 4.1 对基本有向图的某节点 N ,我们称路径 α 为

节点 N 的小环路径,如果 α 以 N 为第一个和最后一个节点,而且除此以外无重复节点的路径。如果某节点 N 的小环路径的个数大于或等于 1,则称此节点为环节点。

算法 4.1 (求小环路径集合的算法) 假设有基本有向图 T 和它的某个节点 N 。对 T 中所有节点 N_i ,本算法求出从节点 N 到达此点 N_i 所有可能的无重复节点的路径集合 $X(N_i)$ (只有在 $N_i=N$ 的情况下,允许路径的首和尾是 N 是重复的)。对于节点 N , $X(N)$ 即是节点 N 的所有的小环路径的集合。如果 $X(N)$ 不空,则节点 N 是环节点。具体的算法如下:

(1) 算法开始时, T 中所有节点 N_i 上的所求的路径集合为空集合 $X(N_i)=\{\}$ 。

(2) 令节点 N 的路径集合 $L(N)=\{N\}$ 。注意这里放在 $L(N)$ 中的是长度为一的仅有一个节点 N 的路径,我们称其为平凡路径。

(3) 对图中所有的由有向边连接的节点 Q 和 H , 其中 Q 是前节点, H 是后节点, 如果有路径 $\alpha \in X(Q)$ 满足条件:

① 节点 H 不在 α 中出现, 或者 $H=N$ 仅作为 α 的第一个节点出现, 而且

② 路径 αH 不包含在 H 的路径集合 $X(H)$ 中,

则将路径 αH 加入到 $X(H)$ 中, 即令 $X(H)=X(H) \cup \{\alpha H\}$ 。

(4) 重复动作 3, 直至所有的由有向边连接的节点 Q 和 H , $X(Q)$ 中没有满足上述条件的路径。

(5) 从 $L(N)$ 中删除平凡路径。这里所谓的平凡路径是指算法第 2 步, 放在 $L(N)$ 中的长度为一的仅有一个节点 N 的路径。

定义 4.2 假设有基本有向图 T 的某路径 β , 如果 β 中有一个节点 N 是环节点, α 是节点 N 的某小环路径, 我们把由 β 将其中节点 N 替换为 α 生成的路径称为由 β 加环生成的路径。

定理 4.1 若 β 是基本有向图 T 的路径, 则由 β 加环生成的路径都是图 T 的路径。

证明: 按小环路径的定义, 小环路径是图 T 的路径, 所以将 β 中某环节点替换为小环路径后生成的路径仍然是图 T 的路径。

定理 4.2 基本有向图 T 的任何路径, 都可以由 T 的某无重复节点的路径, 经过有穷次 (注: 包括零次) 的加环操作而生成。

证明: 用归纳法, 施归纳于路径的节点数。对于长度为 1 的仅有 1 个节点的平凡路径, 定理显然成立。假设定理对于长度为 $n-1$ 的路径成立, 现在证明定理对于长度为 n 的路径也成立。令 $a=\langle a_1, a_2, \dots, a_{n-1}, a_n \rangle$ 是有向图 T 的任何一条长度为 n 的路径。按归纳假设, 路径 $\langle a_1, a_2, \dots, a_{n-1} \rangle$ 可以由 T 的某无重复节点的路径 $b=\langle b_1, \dots, b_k, \dots, b_m \rangle$, 经过有穷次的加环操作而生成。如果 a_n 不在 b 中出现, 则 a 可以由无重复节点的路径 $\langle b_1, \dots, b_k, \dots, b_m, a_n \rangle$, 经过有穷次的加环操作而生成, 定理得证。如果 a_n 在 b 中出现, 假设 $a_n=b_k$, 则 $\langle b_1, \dots, b_k, \dots, b_m, a_n \rangle$ 可以由无重复节点的路径 $\langle b_1, \dots, b_k \rangle$, 将节点 b_k 替换为小环路径 $\langle b_k, \dots, b_m, a_n \rangle$ 后生成, 从而 a 可以由无重复节点的路径 $\langle b_1, \dots, b_k \rangle$ 经过有穷次的加环操作而生成。定理证毕。

现在我们对路径测试方法给以完整地概括说明。

(1) 使用任何一类自动机 (不限定是有穷状态自动机) 所接受的语言来定义系统的性质, 并构造相应的性质自动机。

(2) 使用有穷自动机定义系统的约束 (开始验证时可能没

有约束)。

(3) 根据性质自动机和约束自动机涉及的事件字母表, 用 RCFG (可能有环) 描述各个并行任务, 把 RCFGs 组合起来构成 TFG 为并行系统的执行序列建模。

(4) 用算法 3.1 求出 TFG 从始点到终点的所有不同的无重复节点的路径集合, 并在算法的执行过程中删掉那些导致进入约束自动机违背状态的路径。

(5) 用算法 4.1 求出 TFG 中所有环节点的小环路径集合。

(6) 任意选择从始点到终点的无重复路径, 经任意次的加环操作, 生成任意的路径。

(7) 对这些路径, 分别用性质自动机来判断。如果这些路径被性质自动机所接受, 并不能证明系统满足此性质。此时可以中止测试, 宣称测试未找到错误。也可以转到 6, 再选新的路径继续测试。

(8) 如果有路径不被性质自动机所接受, 同有穷状态验证一样有两种可能:

① 增加约束, 转到 2, 提高模型的近似度, 删掉此不现实的路径。

② 如果该路径是现实的, 则提供信息帮助找到系统违背性质的反例。

从上面的分析可以看出, 本文提出的“路径测试”方法, 同一般的程序测试类似, 用这种方法通过某些路径的测试不能正面地验证程序的正确性, 可是如果通不过测试, 则能帮你发现反例, 找出程序的错误。与一般的程序测试方法不同的是这里的测试是相对于性质和模型的路径, 而不执行实际的程序。

结束语 本文讨论了程序的有穷状态验证方法的局限性, 并尝试突破性质自动机只能使用有穷自动机的限制, 提出了一种新的验证方法: 有穷路径验证法。在这种方法中, 所论证的性质表示可以推广到使用任何一类自动机。作为代价, 描写系统的模型限制是无环的, 而且在较复杂的情况下, 算法的复杂度会增加, 但是仍不失为一种有效的验证方法。对于有环的描写系统的模型, 本文提出了一种称之为“路径测试”的方法。与一般的程序测试方法不同的是这里的测试相对于性质和模型的路径, 而不执行实际的程序。

本文只是初步研究工作的报告, 对于该方法的进一步研究, 特别是有关理论方面、实际应用方面和支持工具的研发方面还将继续。

参考文献

- Clarke L A, Avrunin G S, Cobleigh J M, et al. FLAVERS: A Finite State Verification Approach for Software Systems. Clarke 教授 2004-10 在西北大学的演讲. <http://mainpage.nwu.edu.cn/hkg/docs/>
- Clarke L A. Finite State Verification: An Emerging Technology for Validating Software Systems. Clarke 教授 2003 在上海的演讲. <http://www.iturils.com/Expert/exp-Lori.asp>
- Cobleigh J M, Clarke L A, Osterweil L J. FLAVERS: A finite state verification technique for software systems. IBM Systems Journal, 2002, 41(1): 140~165
- Dwyer M B, Clarke L A. Data Flow Analysis for Verifying Properties of Concurrent Programs. In: Second ACM SIGSOFT Symposium on the Foundations of Software Engineering, New Orleans, LA, 1994. 62~75
- Dwyer M B, Clarke L A, Cobleigh J M, et al. Flow analysis for verifying properties of concurrent software systems. TR UM-CS-2004-006. University of Massachusetts, Department of Computer Science, Amherst, MA 01003, Feb. 2004
- Hao Kegang. Two Formal Models of Interactive Machine. In: Proceedings of The Third Asian Workshop on Foundations of Software, Xi'an, China, November 2004
- 郝克刚. 开放网-交互式并行系统的模型. 西北大学学报 (自然科学版), 1997(5)