# 一种改进的大规模CTL公式检测算法

奚 琪 王清贤 曾勇军 秦艳锋
(国家数字交换系统工程技术研究中心 郑州 450002)

**摘 要** 标记算法是模型检测用于验证计算树逻辑CTL公式的经典算法。针对标记算法检测大规模公式存在的效率问题,提出一种可用于验证大规模CTL公式的标记算法。算法通过公式预处理标识公式集中的公共子公式,在验证过程中绑定公共子公式与模型状态,避免公式的重复验证。实验结果表明,该算法有效提高了验证效率。

**关键词** 模型检测,标记算法,公共子公式,语法分析树
**中图法分类号** TP309 **文献标识码** A

## Improved Algorithm for Large-scale CTL Formulae Verification

XI Qi WANG Qing-xian ZENG Yong-jun QIN Yan-feng
(China National Digital Switching System Engineering and Technological Research Center, Zhengzhou 450002, China)

**Abstract** Labelling algorithm is a standard algorithm for model checking CTL formulas. This paper presented an algorithm to improve efficiency for large-scale CTL formulas verification. It identifies common subformulas from formulae set, binding program states with labels of common subformulas to avoid repeating checking them. The experimental results illustrate the advantages on the efficiency of the new algorithm.

**Keywords** Model checking, Labelling algorithm, Common subformulas, Parse tree

## 1 引言

随着计算机软硬件的规模与复杂性不断增大,提高系统正确性和可靠性的验证技术日益受到重视。在诸多的验证方法之中,模型检测以其自动化程度高、实用性强受到广泛关注。目前模型检测已在硬件设计、通信协议等领域得到广泛应用,并在软件安全领域也有很大进展。

Clarke[1]等提出利用计算树逻辑(Computation Tree Logic,CTL)描述并发系统的性质,设计实现了检验有穷状态系统是否满足给定CTL公式的标记算法[2],这为自动化验证并发系统性质奠定了基础。文献[3]提出的EMC算法能够提高检测模型的规模和验证公式的复杂性,但对于提高验证效率并没有太大的影响;文献[4,5]提出的ALMC算法通过对子公式进行预先评估的方式消除子公式中不必要的验证以提高验证效率,不足之处是预先评估的代价较大,且只考虑验证一个CTL公式的情况;此外,符号模型检测[6,7]、抽象解释[8,9]、偏序归约[10]等技术则为解决模型规模的"状态空间爆炸"问题提供了有效的方法。

上述技术和方法虽然在不同程度上对模型检测的效率和模型规模都有所提升,但大都只考虑验证系统单一属性,即一次只验证一个CTL公式是否满足模型。随着软件模型检测的大规模应用,需要验证多个属性的情况更加普遍。对于验证大规模CTL公式的情况,目前主要采用CTL公式逐一验证的方法,这种不考虑公式相似性的验证方法效率低下,无法满足现实的验证需求。

本文在标记算法的基础上提出了一种验证大规模CTL公式的模型检测算法LSLA(Large Scale Labelling Algorithm)。算法首先通过公式预处理标识出公式集中的所有公共子公式,对公式进行合并生成CTL公式集语法树。模型检测过程中通过绑定公共子公式及其在模型中可满足的状态,克服重复验证公共子公式的问题。实验说明该算法可有效提高大规模CTL公式的验证效率。

## 2 问题提出

现实中使用模型检测进行属性验证时,常会遇到针对同一个模型检验多个属性的情况。通常将模型表示为迁移系统$M$,其状态集合为$S$,将待验证的属性表示为一个CTL公式集合$C=\{\varphi_1,\varphi_2\cdots,\varphi_m\}$,$\varphi_i$为$C$中的一个CTL公式。通过模型检测工具检验$M$是否满足$\varphi_i$,即$M,s|=\varphi_i(\varphi_i\in C,s\in S)$。传统的标记算法验证大规模CTL公式时并未考虑各个公式之间是否具有相同的子公式,而是采用依次验证公式集中公式的方式。

标记算法是验证CTL公式的典型算法,其主要思想是选定一个适当的连接词集合$\{\neg,\wedge,\bot,AF,EU,EX\}$,将CTL公式$\varphi$转换为由该连接词集合组合成的等价形式,然后从$\varphi$中最小的子公式开始,用$\varphi$的被满足的子公式来标记$M$的状

到稿日期:2012-12-18 返修日期:2013-03-12 本文受国家863计划(2012AA012902)资助。

奚 琪(1978—),男,博士生,讲师,CCF会员,主要研究方向为网络信息安全,E-mail:keikey43@163.com;王清贤(1960—),男,教授,博士生导师,主要研究方向为网络安全、计算复杂性理论;曾勇军(1973—),男,硕士,讲师,主要研究方向为网络信息安全;秦艳锋(1978—),男,博士生,讲师,主要研究方向为网络信息安全。

态，并由里向外逐步扩展 $\varphi$，最后通过查看标记为 $\varphi$ 的状态中是否包含模型的初始状态来确定模型是否满足 $\varphi$。

为了证明标记算法在检测多个 CTL 公式时存在的效率问题，我们给出以下定义及定理。

**定义 1** CTL 公式的语法分析树（简称语法树）是根据 CTL 文法构造的二叉树，它的叶子节点为公式中的原子公式，其它节点为公式中的连接词。

**定义 2** 一个 CTL 公式 $\varphi$ 的子公式 $\psi$ 是这样的公式，其语法分析树是 $\varphi$ 的语法分析树的子树，记为 $\psi \sqsubset \varphi$。

**定义 3** 设 CTL 公式 $\varphi_1$，$\varphi_2$ 和 $\psi$，若 $\psi \sqsubset \varphi_1$ 且 $\psi \sqsubset \varphi_2$，则称 $\psi$ 为公式 $\varphi_1$ 与 $\varphi_2$ 的公共子公式。

**定义 4** $\varphi_1$，$\varphi_2$，$\varphi$ 为 CTL 公式，若三者满足 $\varphi=\begin{cases} op_u(\varphi_1), op_u \text{ 为一元连接词} \\ \varphi_1 op_b \varphi_2, op_b \text{ 为二元连接词} \end{cases}$，则称 $\varphi_1$，$\varphi_2$ 为 $\varphi$ 的直接子公式。

例 1 假设有如下两个 CTL 公式：

$$A[AX\neg pUE[EX(p \wedge q)U\neg p]] \tag{1}$$

$$EF[(AX\neg p) \wedge EX(p \wedge q)] \tag{2}$$

式(1)与式(2)的公共子公式构成的集合为 $\{p, q, \neg p, p \wedge q, EX(p \wedge q), AX\neg p\}$。式(1)中 $p$ 为 $\neg p$，$p \wedge q$ 为 $EX(p \wedge q)$ 的直接子公式。

**定理 1** 设 CTL 公式 $\psi$ 和 $\varphi$，若 $\psi$ 为 $\varphi$ 的子公式，即 $\psi \sqsubset \varphi$，则标记算法验证 $M,s|=\varphi$ 的过程中一定先验证 $M,s|=\psi$。

证明：因为 CTL 公式 $\psi$ 和 $\varphi$ 满足关系 $\psi \sqsubset \varphi$，则可写为 $\varphi=\theta_1 op_1(\theta_2 op_2 \cdots \theta_i op_i(\psi)\cdots)$，其中 $\theta_k$，$op_k$ $(0<k\leqslant i)$ 分别表示 CTL 公式和 CTL 连接词。令公式 $\delta=\theta_i op_i(\psi)$，则 $\delta \sqsubset \varphi$ 且 $\psi$ 为 $\delta$ 的直接子公式。

首先证明验证 $M,s|=\delta$ 的过程必须先验证 $M,s|=\psi$。令 $SAT(\psi)$ 为求解 $M,s|=\psi$ 的函数算法，其返回值为满足公式 $\psi$ 的集合，标记算法要求公式的验证是由其直接子公式扩展进行验证，则 $\delta$ 与 $\psi$ 一定存在如下关系中的一种[2]，左侧表示 $\delta$ 与 $\psi$ 存在的关系类型，右侧为函数推导过程：

(1) $\delta=\neg\psi$ $\quad SAT(\delta)=S-SAT(\psi)$

(2) $\delta=\psi \wedge \psi_1$ $\quad SAT(\delta)=SAT(\psi) \cap SAT(\psi_1)$

(3) $\delta=\psi \vee \psi_1$ $\quad SAT(\delta)=SAT(\psi) \cup SAT(\psi_1)$

(4) $\delta=\psi \rightarrow \psi_1$ $\quad SAT(\delta)=SAT(\neg\psi \vee \psi_1)$
$=SAT(\neg\psi) \cup SAT(\psi_1)$
$=(S-SAT(\psi)) \cup SAT(\psi_1)$

(5) $\delta=AX\psi$ $\quad SAT(\delta)=SAT(\neg EX\neg\psi)$
$=\neg SAT_{EX}(\neg\psi)$
$=S-SAT_{EX}(S-SAT(\psi))$

(6) $\delta=EX\psi$ $\quad SAT(\delta)=SAT_{EX}(\psi)$

(7) $\delta=A[\psi U\psi_1]$ $\quad SAT(\delta)=SAT(\neg(E[\neg\psi_1 U(\neg\psi \wedge \neg\psi_1)] \vee EG\neg\psi_1))$
$=(S-SAT_{EU}(\neg\psi_1, (\neg\psi \wedge \neg\psi_1))) \vee (S-SAT_{AF}(\psi_1))$

(8) $\delta=E[\psi U\psi_1]$ $\quad SAT(\delta)=SAT_{EU}(\psi, \psi_1)$

(9) $\delta=EF\psi$ $\quad SAT(\delta)=SAT_{EU}(T, \psi)$

(10) $\delta=EG\psi$ $\quad SAT(\delta)=SAT(\neg AF\neg\psi)$
$=S-SAT_{AF}(\neg\psi)$

(11) $\delta=AF\psi$ $\quad SAT(\delta)=SAT_{AF}(\psi)$

(12) $\delta=AG\psi$ $\quad SAT(\delta)=SAT(\neg EF\neg\psi)$
$=S-SAT(EF\neg\psi)$
$=S-SAT_{EU}(T, \neg\psi)$

由式(1)－式(4)直接可知，求解 $SAT(\delta)$ 必须先求解 $SAT(\psi)$，根据 CTL 公式的等价性，将式(5)－式(12)转换为由连接词 $\{\neg, \wedge, T, AF, EU, EX\}$ 构成的等价公式，由 $SAT_{EU}(\psi_1, \psi)$，$SAT_{EX}(\psi)$ 及 $SAT_{AF}(\psi)$ 的推导算法[2]可知，其求解过程中必须先求解 $SAT(\psi)$。由此可知，当 $\psi$ 为公式 $\delta$ 的直接子公式时，验证 $M,s|=\delta$ 的过程中必须验证 $M,s|=\psi$。

由于标记算法是由直接子公式依次向外扩展验证，则当第 $i-1$ 次扩展后，$\varphi$ 的所有子公式都已验证完毕。由此可证，标记算法验证 $M,s|=\varphi$ 的过程中一定先验证 $M,s|=\psi$。

**引理 1** 设 CTL 公式 $\varphi_1$ 和 $\varphi_2$，若 $\varphi_1 \cap \varphi_2=\{\psi_i\}$，$\psi_i$ 为 $\varphi_1$ 与 $\varphi_2$ 的一个公共子公式，则通过标记算法分别验证 $M,s|=\varphi_1$ 和 $M,s|=\varphi_2$ 的过程中重复验证 $M,s|=\psi_i$。

证明：由于 $\psi_i$ 为 $\varphi_1$ 的子公式，由定理 1 知验证 $M,s|=\varphi_1$ 必先验证 $M,s|=\psi_i$，同理验证 $M,s|=\varphi_1$ 必验证 $M,s|=\psi_i$，显然 $M,s|=\psi_i$ 被重复验证两次，引理得证。

**定理 2** 设 CTL 公式 $\varphi=\psi_1$ op $\psi_2$，其中 op 为 CTL 的二元连接词，若 $\delta$ 为 $\psi_1$ 和 $\psi_2$ 的公共子公式，则验证 $M,s|=\varphi$ 的过程中必然重复验证 $M,s|=\delta$。

证明：因为 CTL 公式 $\varphi=\psi_1$ op $\psi_2$，所以 $\psi_1 \sqsubset \varphi$，$\psi_2 \sqsubset \varphi$。由定理 1 可知，$M,s|=\varphi$ 的验证过程中需要先验证 $M,s|=\psi_1$ 和 $M,s|=\psi_2$。又因为 $\delta$ 为 $\psi_1$ 和 $\psi_2$ 的公共子公式，则 $\delta \sqsubset \psi_1$ 且 $\delta \sqsubset \psi_2$。由定理 1 知，验证 $M,s|=\psi_1$ 和 $M,s|=\psi_2$ 的过程中均需验证 $M,s|=\delta$，定理得证。

引理 1 和定理 2 说明，无论公共子公式是出现在 CTL 公式集中的不同公式，还是属于 CTL 公式的不同子公式，对同一个模型进行 CTL 公式的验证都会造成其公共子公式的重复验证。

## 3 LSLA 检测算法

### 3.1 公式处理

为了避免检测过程对公式集中出现的公共子公式重复验证，需要准确标识公式集中的所有公共子公式。

用语法树结构表示 CTL 公式有利于标记算法由子公式向外逐步扩展的匹配过程，但是 CTL 公式的标准语法树既没有体现二元连接词 AU、EU 等所连接的两个子公式的时序关系，也未能明确标识公共子公式，需要对其改进。扩展语法树通过合并公共子树削减了标准语法树中的重复节点，同时用左右孩子节点表示两个子公式的时序关系。扩展语法树生成算法以标准语法树为输入，通过后序遍历算法对 CTL 公式语法树各节点访问，同时由叶节点向根节点方向生成新的语法树，若当前访问的节点及其所有孩子节点已在新语法树中存在，则直接在语法树中标记，不再重复生成，如此可保证公共子公式只在树中出现一次。

设 CTL 公式的语法分析树为 $T\langle root_T\rangle$，则扩展语法树 $S\langle root_S\rangle$ 的生成过程如下：

(1)新语法树 $S$ 初始化为空，$root_S$=NULL；

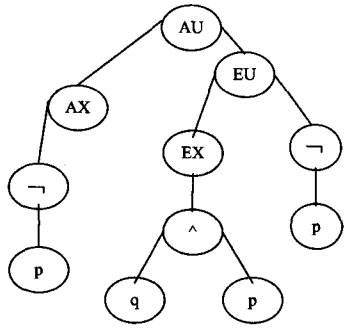
(2)后序遍历 $T$,取出一个节点 $N$,当节点 $N$ 为 NULL 时,转(5),否则转(3);

(3)若 $N$ 是原子公式且未在 $S$ 中存在,则新生成一个节点 $M$,若已存在则标识为公共子公式并返回(2);否则说明 $N$ 为连接词,转(4);

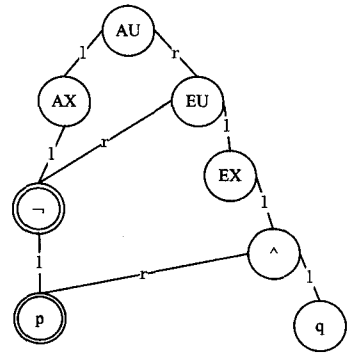
(4)$N$ 为中间节点,若 $N$ 在 $S$ 中存在,且以 $N$ 为根的子树在 $S$ 中匹配成功,则 $N$ 被标记为公共子公式,否则,新生成节点 $M$,并依据其在 $T$ 中信息找到其在 $S$ 中的孩子,按时序关系标识;返回(2);

(5)$root_S=M$,返回 $root_S$ 为树 $S$ 的根节点。

图 1(a)、(b)显示了式(1)经由扩展树生成算法生成的前后的两种语法树的对比。



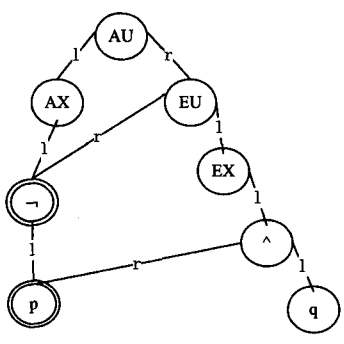

(a) 式(1)的标准语法树



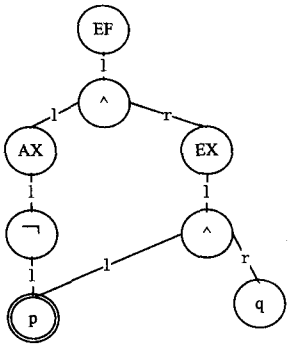

(b) 式(1)的扩展语法树

图 1

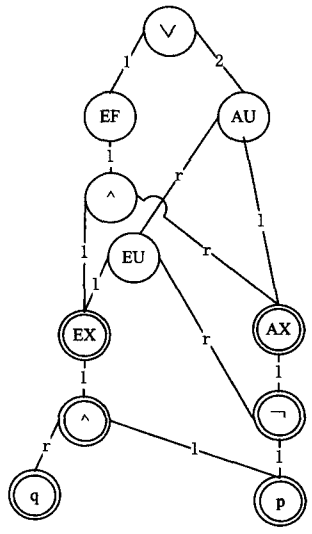
公共子公式不仅存在于公式内部,还存在于各公式之间。为了明确标识各公式之间的公共子公式,我们引入了新的根节点 root,将 CTL 公式集中的各公式语法树进行了合并。为了保证验证时各 CTL 公式的相对独立性,根节点设置为连接词"∨",其孩子为各个公式改进语法树的根节点,合并过程仍采用语法树生成算法。图 2 为式(1)与式(2)合并后最终生成的语法树结构。




(a) 式(1)的语法树




(b) 式(2)的语法树




(c) 合并后的语法树

图 2

### 3.2 LSLA 检测算法

LSLA 检测算法将生成后的 CTL 公式集语法树与模型 $M$ 作为输入,对于语法树中的每一个 CTL 公式 $\varphi_i$,通过 LSLA 检测算法输出 $M,s|=\varphi_i$ 的结果是或否。与标记算法不同的是,LSLA 算法在进行子公式满足性验证时先确定该子公式是否为公共子公式,若是而且已在模型中验证过,则不再匹配每一个模型状态,直接返回满足该子公式的状态,否则按标记算法处理。为了节省内存空间,算法在每一个公式检测完毕后,将模型中所有非公共子公式的标记清除。

设 $\varphi$ 是 CTL 公式 $\phi$ 的子公式,$\psi$ 为 $\varphi$ 的直接子公式,假设满足 $\psi$ 的所有直接子公式状态都已经被标记,则 LSLA 算法检测 $\varphi$ 的可满足性过程描述如下:

(1)检查 $\varphi$ 是否已被标记为公共子公式,若是转(2),否则转(3);

(2)检查 $\varphi$ 是否已经进行过 $M,s|=\varphi_i$ 的检测,若是转(5),否则转(3);

(3)根据连接词类型和已模型中标记 $\psi$ 的状态,用标记算法找出满足 $\varphi$ 的状态,转(4);

(4)若 $\varphi$ 是公共子公式,在 $M$ 中绑定公共子公式的状态,同时在语法树中将 $\varphi$ 标记为已被检测,转(5);

(5)若 $\phi==\varphi$,清除所有 $M$ 中非绑定状态的标记,否则,直接返回标记 $\varphi$ 的状态。

算法实现代码如下:

**算法 1** LSLA_检测算法

输入:Kripke * module, Tree * ctl_tree, Formular * f;//模型 M,CTL 公式语法树(即公式集语法树的一棵直接子树),及待验证的子公式 f

输出:Kripke * module;//标记 f 与状态后的模型

(1)global Formular * g []=ExtractFromTree(ctl_tree);//从语法树生成待验证的子公式集合 g,子公式 f∈g

(2)Kripke * LSLA_Aogrithm(Kripke * module, Formular * f, Tree * ctl_tree)

(3){

```
(4)  Formular *f1,*f2;//f的直接子公式f1,f2
(5)  int ctype; //f中f1,f2之间的连接词
(6)  if(Is_Comsubformula(f)){//若f为公共子公式
(7)    if(Is_Checked(ctltree,f))//若f已在树中标记
(8)        goto:exit;
(9)     else{
(10)        MarkChecked(ctl_tree,f);//标记f被检测
(11)        goto:labeled;}
(12)   }
(13)  labeled://标记算法
(14)    GetfromFormular(ctype,f,f1,f2);//从f中得到直接子公式
        等信息
(15)    Labeled(module,ctype,f1,f2);//标准的Label算法
(16)exit:
(17)    if(Is_CompleteCTLForm(f,ctl_tree))//若f为CTL公式
(18)      CleanLabel(module);//从模型中清除非公共子公式标记
(19)  return module;
(20)}
```

第(1)行代码中全局变量 $g$ 是由函数 ExtractFromTree 采用后序遍历树的方式生成的 CTL 子公式集合；LSLA 算法的实现过程由第(2)行开始，其中第(6)－(12)行为当公式 $f$ 为公共子公式时的操作，函数 MarkChecked 负责标记语法树中已检测的公共子公式；行(13)－(15)完成标记算法，其中函数 GetfromFormular 得到公式 $f$ 的直接子公式 $f1$，$f2$ 及连接词 ctype，函数 Labeled 为 Label 标记算法；(17)、(18)行实现当 CTL 公式检测完后对模型中非公共子公式标记的清除。

设公式集中有 $n$ 个公式，$\sum_{i=1}^{n} f_i$ 为公式集中所有公式的连接词总和，公式集中存在的公共子公式为 $k$ 个，每个公共子公式出现的次数分别为 $m_j(1\leqslant j\leqslant k)$，且具有连接词个数为 $f_j$ $(1\leqslant j\leqslant k)$ 个，$V$ 为模型中状态个数，$E$ 为模型中迁移个数。采用标记算法计算大规模公式的满足性问题时间复杂度为 $O(\sum_{i=1}^{n} f\cdot V\cdot(V+E))$，而采用 LSLA 算法的时间复杂度为 $O((\sum_{i=1}^{n} f-\sum_{j=1}^{k} f_j\cdot m_j)\cdot V\cdot(V+E))$，由此可见，公共子公式的个数 $k$ 越多，出现的频率越高，则节省匹配的时间就越多。

## 4 模拟实验

为了比较本文提出的 LSLA 检测算法和传统的标记算法的性能，我们利用 VC6.0 工具分别设计实现了传统的标记算法[5]和 LSLA 检测算法。验证模型 $M=(S,\rightarrow,L)$ 来自于静态分析工具 IDA Pro 对二进制程序分析生成的控制流图，其中程序的基本块构成了模型的状态集 $S=\{blk_1,blk_2,\cdots blk_n\}$，标记集由块中包含的系统调用名称构成，表示为 $L=\{emt,syscall_1,syscall_2,\cdots,syscall_m\}$(emt 表示块中无系统调用)，迁移关系则由块间的控制流确定。表示验证属性的 CTL 公式由与安全相关的系统函数调用的时序关系生成，如表 1 所列。第 1 组验证包含公式 1－3，该组公式不存在公共子公式；第 2 组验证由公式 4、5 构成，其中公式 4 重复 4 次；第 3 组验证包括公式 1－10，其公共子公式占所有子公式的 17.8%。验证的结果如表 2 所列。

表 1　实验采用的公式集

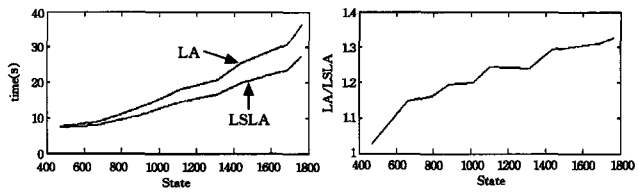
| 序号 | CTL 公式 |
|---|---|
| 1 | EF((callset=GetModuleFileName)& EF(callset =WrtieFile)) |
| 2 | EF(((callset = FindFileFirst) \| (callset = FindFileFirstEx)) & EF (callset=Find NextFile)) |
| 3 | EF(callset=GetSystemDirectory−>(EF(callset=CopyFile) \| EF (callset=MoveFile))) |
| 4 | EF(callset=GetTempDirectory−>EF(callset=MoveFile)) |
| 5 | EF(callset=RegQueryValueEx−>EF(callset=RegSetValueEx)) |
| 6 | EF((callset = FindNextFile) &EF((callset = CreateFileMapping) &EF(WriteFile))) |
| 7 | EF(((callset = GetSystemDirectory \| callset = GetTempDirectory)) &EF(callset=WriteFile)) |
| 8 | EF((callset=fopen)−>EF(callset=send)) |
| 9 | EF((callset= bind) −>(EF(callset = listen) −>EF(callset = send))) |
| 10 | EF((callset=EnumProcesses)−>EF(callset=OpenProcess)) |

表 2　不同 CTL 公式集的 LSLA 算法测试结果

| 程序 | 状态数 | 迁移数 | CTL 公式数 | 公共子公式占有量 | 公式处理算法(s) | 标记算法 | LSLA 算法 | LSLA/LA 时间比(%) |
|---|---|---|---|---|---|---|---|---|
| Ex1.exe | 472 | 2153 | 3 | 0 | 0.257 | 1.536 | 1.653 | 124.34 |
| Ex1.exe | 472 | 2153 | 5 | 80% | 0.571 | 3.172 | 1.795 | 74.59 |
| Ex1.exe | 472 | 2153 | 10 | 17.8% | 0.937 | 7.493 | 6.348 | 97.22 |

表 2 是对同一模型对象 Ex1.exe，不同数量的 CTL 公式集检测的结果。可以看出，当模型相同时，若 CTL 公式中的公共子公式占有量不高，大规模标记算法所消耗的时间与标记算法所消耗时间相差不大，甚至检测效率还要稍低，这主要是由于处理 CTL 公式集的时间消耗较高。当公共子公式占用量很高时，检测效率明显提高，符合公共子公式越多，匹配时间消耗越少的预期。应当注意的是，随着 CTL 公式数量的增加，LSLA 算法中公式预处理消耗的时间相应增长，这成为影响 LSLA 效率的重要因素。

图 3 显示了在相同模型和 CTL 公式集的条件下 LA 与 LSLA 的验证效率对比，其中 10 个程序模型来自于 Windows 系统 System 目录的可执行文件(size<100k byte)，CTL 公式集与表 1 中相同。从图 3(a)中可以看出，程序的状态数量越多，LSLA 算法验证时间就越短，其优势就越明显。图 3(b)显示出 LA 与 LSLA 对不同程序检测消耗的时间比，越陡峭表示消耗时间差越大。可以看到，两者处理第一个模型的时间比较大，经分析发现主要原因在于 LSLA 对 CTL 公式集的预处理过程只在第一个模型匹配时进行。




(a) 程序状态与检测时间对应图　　(b) LA/LSLA 消耗时间比

图 3　LA 与 LSLA 算法效率比较

**结束语**　本文提出了一种适用于验证大规模 CTL 公式的标记算法 LSLA，其基本思想是通过对 CTL 公式的处理，标记所有公共子公式，验证过程中对模型中满足公共子公式的状态进行绑定，减少对公共子公式的重复验证从而提高对

CTL 公式集的检测效率。实验结果验证了 LSLA 算法有效地提高了检测 CTL 公式集的效率。未来的工作重点研究将该思想移植到基于 BDD 或 SAT 机制的符号模型检测工具中，以期在提高规模的同时提升检测模型的效率。

## 参 考 文 献


[1] Clarke E, Grumberg O, Peled D. Model Checking (Second Printing)[M]. London: MIT Press, 2000: 27-30

[2] Michael H, Maark R. Logic In Computer Science: Modelling and Resoning about System (Second Edition) [M]. London: Combridge University Press, 2004: 221-229

[3] Clarke E, Emerson E, Sistla A. Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifcations [C]// ACM Transactions on Programming Languages and Systems. 1986, 8: 244-263

[4] Vergauwen B, Lewi J. A Linear Local Model Checking Algorithm for CTL[C]// Proceedings of 4th International Conference on Concurrency Theory, 1993. Hidesheim, Germany, Springer-Verlag, 1993: 447-461

[5] Heljanko K. Implementing a CTL Model checker[C]// Workshop Concurrency: Specification & Programming. Humboldt-University zu Berlin, Institut fur Informatik, 1996: 75-84

[6] Mumme M, Ciardo G. A fully symbolic bisimulation algorithm [C]// Giorgio Delzanno and Igor Potapov. RP, volume 6945 of Lecture Notes in Computer Science, Springer, 2011: 218-230

[7] Lefticaru R, Ipate F. Automated Model Design using Genetic Algorithms and Model Checking[C]// 2009 Fourth Balkan Conference in Informatics (BCI'09). Thessaloniki, IEEE Computer Society, 2009: 79-84

[8] 李梦君，李舟军，陈火旺. 基于抽象解释理论的程序验证技术[J]. 软件学报，2008，19(1)：17-26

[9] Cousot P. Formal Verification by Abstract Interpretation[C]// NASA Formal Methods 2012, 4th International Symposium. NFM, Norfolk, VA, USA, 2012: 3-7

[10] Meulen J V, Pecheur C. Milestones: A model checker combining symbolic model checking and partial order reduction [C]// NASA Formal Methods 2011, volume 6617 of LNCS. Springer, 2011: 525-531


(上接第 107 页)

毫秒，BGP 的优势并没有太大意义。与 Centaur 相比，ZCR 提升了约 75%网络事件的收敛速度，这是由于 ZCR 将部分网络事件的规模控制在区域内部，不会导致全网范围内的重新收敛。但是与 Centaur 相比 ZCR 仍然会出现部分长时间收敛事件，这是因为 CIMR 在区域间仍然采用 BGP 的原理，在某些连接失效时收敛过程仍然会近似于 BGP。

### 3.2 实验 2-单个失效事件导致的网络负载

本实验验证了 ZCR 的负载。实验采用如下方式：分别在相同的网络拓扑下部署 BGP、Centaur 以及 ZCR 3 种网络协议；将网络中的某个连接移开，等待网络收敛；再将该连接移回，直到网络稳定，统计在此过程中总共产生的网络消息数目。图 8 中给出了实验结果的 CDF。

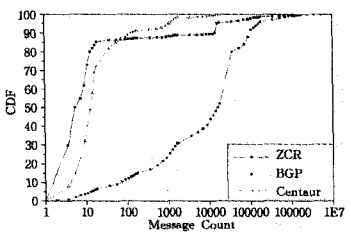



图 8 网络负载

从图 8 可以看到，ZCR 与 BGP 相比极大地减少了消息数目，我们分析这是因为采用了区域的方式，降低了某些事件的全局影响，减少了更新消息数量，并且在区域内部使用 Centaur 协议，同样起到降低网络负载的效果。与 Centaur 相比，ZCR 中更多的网络事件只在极少量的网络消息数目后就收敛了，我们认为这是由于 ZCR 采用了基于区域的方式，将部分事件的影响限制在区域内，降低了网络负载。然而，ZCR 中也有部分事件的负载比 Centaur 大，这是因为在 ZCR 中如果事件导致了区域间的收敛，收敛过程与 BGP 类似，将会带来较大的网络负载。

**结束语** 为了降低 BGP 协议的全局敏感性，同时保留对策略的支持，本文提出了一种基于区域的可控路由模型 ZCR。ZCR 提出对网络进行区域划分，在区域内使用支持策略的链路状态路由协议 Centaur，区域间使用扩展的路径向量协议 Q-BGP。ZCR 提供了每条连接的控制手段，并且能够将部分网络事件的影响控制在区域内，降低了路由的全局敏感性，提高了网络的收敛速度。此外，ZCR 中实现了 Q-BGP 与 BGP 的兼容，使其能够在网络中递增部署，逐步构建一个可控的路由环境，为构建可控的网络打下基础。

在未来的研究中，我们拟研究在 ZCR 下如何通过各区域间的协作构建可行的网络控制措施来提高网络的控制效率。

## 参 考 文 献


[1] John J P, Katz-Bassett E, Krishnamurthy A, et al. Consensus Routing: The Internet as a Distribute System [C]// Proceedings of the fifth USENIX Network Symposium on Design and Implementation(NSDI'08). 2008: 351-364

[2] Subramanian L, Caesar M, Ee C T, et al. HLP: A next generation inter-domain routing protocol [C] // Proc. ACM SIGCOMM. 2011: 13-24

[3] Zhang Xin, Perrig A, Zhang Hui. Centaur: A Hybrid Approach for Reliable Policy-Based Routing[C]// the 29th International Conference on Distributed Computing Systems (ICDCS). June 2010: 22-26

[4] Pei D, et al. BGP-RCN: Improving BGP Convergence through Root Cause Notification[R]. TR-030047. UCLA, 2012

[5] Karlin J, et al. Nation-State Routing: Censorship, Wiretapping, and BGP[J]. Computing Research Repository-CORR, 2009, 903

[6] SSF-Scalable Simulation Framework[OL]. http://www.ssfnet.org, 2010-04-23

[7] Medina A, Lakhina A, Matta I, et al. BRITE: An approach to universal topology generation[C]// MASCOTS. 2011

[8] 吴大鹏，杨正川，刘乔寿，等. 带有相遇预测的自适应路由机制[J]. 重庆邮电大学学报：自然科学版，2013，25(3)：368-371