符号化模型检测 CTL*

苏开乐[1),2)] 骆翔宇[1)] 吕关锋[3)]

[1)](中山大学计算机科学系 广州 510275)

[2)](河南科技大学电子信息工程学院 洛阳 471003)

[3)](北京工业大学计算机科学与技术学院 北京 100022)

摘 要 提出了一个关于时态逻辑 CTL* 的符号化模型检测算法. 该算法通过所谓的 tableau 构造方法来判定一个有限状态系统是否满足 CTL* 规范. 根据该理论,作者已实现了一个基于 OBDD 技术的 CTL* 符号化模型检测工具 MCTK,并完成了相当数量的实验. 到目前为止,已知有名的符号化模型检测工具,如 SMV 和 NuSMV 等,都只能对 CTL* 的子集逻辑(如 CTL,LTL)进行检测,而文中算法的结果是令人满意的,并且当规范不是特别复杂时,高效的 CTL* 符号化模型检测是可能的.

关键词 模型检测;时态逻辑;有序二值判定图(OBDD)

中图法分类号 TP301

Symbolic Model Checking for CTL*

SU Kai-Le[1),2)] LUO Xiang-Yu[1)] LU Guan-Feng[1)]

[1)](*Department of Computer Science, Sun Yat-Sen University, Guangzhou* 510275)

[2)](*Institute of Electronics and Information Engineering, Henan University of Science and Technology, Luoyang* 471003)

[3)](*College of Computer Science and Technology, Beijing University of Technology, Beijing* 100022)

Abstract This paper gives a symbolic model checking algorithm for the temporal logic CTL*. The algorithm determines whether a finite state system satisfies a formula in CTL* using the method of symbolic tableau construction. According to our algorithm, we had implemented a new CTL* symbolic model checker (MCTK) by means of OBDD and obtained some experimental results. Up to now, the well-known symbolic model checking tools (SMV, NuSMV etc.) have been implemented only for the sublogics of CTL*, such as CTL and LTL. The results that we have obtained in this paper are quite surprising and show that efficient CTL* model checking is possible when the specifications are not excessively complicated.

Keywords model checking; temporal logic; Ordered Binary Decision Diagrams (OBDD)

1 引 言

1.1 概述

模型检测是一种被广泛使用的验证有限状态系统满足规范的自动化技术. 在这里,形式化规范被描述为时态逻辑公式,如工具 SPIN[1] 及 FORSPEC[2] 使用的 LTL(线性时态逻辑)、SMV 中使用的 CTL(分支时态逻辑)[3]. 这些验证技术被称为时态逻辑模型检测,是由 Clarke 和 Emerson[4] 及 Queille 和

收稿日期:2004-08-29;修改稿收到日期:2005-07-14. 本课题得到国家自然科学基金(60496327,10410638,60473004)、广东省自然科学基金团队项目(04205407)与教育部留学回国人员科研启动基金以及上海市智能信息处理重点实验室(筹)开放课题资助.苏开乐,男,1964 年生,教授,博士生导师,研究兴趣包括模型检测、知识推理、非单调推理、多智能体系统、模态逻辑、时态逻辑、概率推理、安全协议验证和逻辑程序设计等. E-mail: issraymond@163.com. 骆翔宇,男,1974 年生,博士研究生,研究兴趣包括模型检测、时态逻辑、模态逻辑、知识推理、多智能体系统、安全协议验证和逻辑程序设计等. 吕关锋,男,1973 年出生,博士研究生,研究兴趣包括模型检测、知识推理、多智能体系统、模态逻辑、时态逻辑、概率推理、安全协议验证和逻辑程序设计等.

Sifakis[5]在 20 世纪 80 年代最先发展的. 在这种方法中，规范被描述成命题时态逻辑公式，系统(如电路设计、协议)被模型化为状态转换系统，使用高效的搜索算法来判定规范是否在系统中成立.

对于描述状态转换系统的性质来说，时态逻辑 CTL* 是一种表达力很强的规范描述逻辑[6]. 它用时态算子描述沿计算路径的状态变化，用路径算子解释时间分支性质. 很多系统性质可以用 CTL 或 LTL 表达，但有些性质只能用 CTL* 表达. 例如，CTL* 公式 $A(FGp) \vee AG(EFp)$，表示沿着所有的计算路径，要么命题 p 从某一状态开始一直成立下去；要么始终存在另一计算路径，使得命题 p 在这条路径的某一状态上成立. 可以看出 CTL 和 LTL 均不能表达该公式. 所以一个功能全面的模型检测工具应能检测 CTL* 公式.

大体上，有两种基本的模型检测方法. 第一种方法是构造出系统的所有可达状态(例如文献[4,7]对于 CTL 及文献[8,9]对于 LTL). 但这种方法面临着状态爆炸问题，特别当系统转换关系很复杂时. 使用偏序归约(partial order reduction)[10~12]可一定程度上减轻该问题. 另一种更好的方法就是所谓的符号化方法(symbolic method)[13]，该方法使得系统的规模得以大幅度地提高，这种提高得益于有序二值判定图(Ordered Binary Decision Diagrams，OBDDs)[14]的使用，OBDDs 是一种用来表示布尔函数的数据结构.

目前已有关于 CTL[3,13]及 LTL[15,16]的符号化模型检测方法. 这自然导致一个问题：是否同样技术可用于 CTL* 的模型检测？

Emerson 等在文献[6,17]里表明，CTL* 模型检测有 LTL 模型检测一样的复杂度. Emerson 和 Lei[6]已给出如何把 CTL* 模型检测问题转化为 LTL 模型检测问题的方法. 考虑到这一点，Clarke 等在文献[15]中给出了将 LTL 检测转化为 CTL 模型检测的方法，同时希望通过这两种方法的结合，来解决 CTL* 的符号化模型检测问题. 但是，从 CTL* 模型检测到 LTL 模型检测[6]不是基于 OBDDs 的，而 LTL 到 CTL 的转化是基于 OBDDs. 所以集成这两种方法是困难的.

在本文中，我们给出一个 CTL* 符号化模型检测算法. 该算法时间复杂度和已知的 CTL* 算法(非符号化)相一致. 同时我们指出如何构造出在实际应用中高效的 CTL* 模型检测工具. 对于 CTL 规范，我们的 CTL* 模型检测工具和 CTL 符号化模型检测工具在空间和时间的花费上是相同的. 另外，我们也验证了复杂的 CTL* (非 CTL 或 LTL)公式.

我们将在第 2 节给出 OBDDs 的扼要介绍；第 3 节给出时态逻辑 CTL* 的语法和语义；第 4 节给出我们的主要贡献，即 CTL* 的符号化模型检测算法；第 5 节给出算法的试验结果. 第 6 节总结全文.

1.2　相关工作

在过去的 15 年中，在 CTL* 的子逻辑(CTL，LTL)的符号化模型检测方面研究者们做了很多工作. 对于分支时态逻辑 CTL，Emerson 等基于 OBDDs 实现了 CTL 模型检测算法[4,7]，该算法能够验证超过 10^{20} 个状态[3,13]，甚至 10^{120} 个状态的系统[18,19].

对于线性时态逻辑 LTL 的符号化模型检测，Clarke，Grumberg 和 Hamaguchi[15]给出一种把 LTL 模型检测问题转化为带有公平性约束(fairness constraints)的 CTL 模型检测问题，从而可应用现有的 CTL 符号化模型检测工具(SMV)来对 LTL 公式进行检测. Kesten，Pnueli 和 Raviv[16]给出了一个一致且自包含的 LTL 符号化模型检测方法，不需要把 LTL 检测转化为 CTL 检测或自动机. 关于 CTL* 模型检测，研究者们也提出了一些算法. Bhat，Cleaveland 和 Grumberg 在文献[20]中给出了一个 on-the-fly 算法. Bernholtz，Vardi 和 Wolper 在文献[21]中提出了另一个 CTL* 模型检测算法. 但是，这些算法都不是符号化模型检测算法. 最近，Kesten 和 Pnueli 在文献[22]中给出了一个包含符号化模型检测和演绎验证技术的组合方法验证 CTL* 规范，其中的符号化模型检测算法与本文给出的比较相似，然而文献[22]中没有给出详细的证明和实验结果.

2　二值判定图(Binary Decision Diagrams，BDDs)

BDDs 是一种表示布尔函数的高效方法. 它首先是作为一种简单的形式，即 Binary Decision Trees 被提出的. 在 Binary Decision Trees 中每个非终结节点被布尔变量 $x,y,z,\cdots$ 标注(记为 $var(v)$，v 为节点)，终结节点用 1 或 0 标注，每个非终结节点 v 有两个后继节点，记为 $low(v)$(当 v 赋值 0 时)和 $high(v)$(当 v 赋值 1 时). BDDs 通常包含有冗余，所以不是一种高效简洁的布尔函数的表达方法，因为它有和真值表一样的大小. Bryant 在文献[14]

中阐明如何通过使用两个限制来得到 BDDs 的规范表达:首先,变量出现的顺序要保持一致;其次,不存在同构的子树和多余的节点.概括起来,有三种方法可得到 BDD 的简洁规范形式:

(1)删除多余终结节点:只留下一个(1 或 0,或两者)终结节点,删除其余的并使指向被删除终结节点的连接指向剩下的相应节点.

(2)删除多余的非终结节点:如果二个不同节点 n 和 m 是结构上相同的子 BDDs 的根,则删除其中一个,并且把指向它的所有连接指向另一个.

(3)如果二非终结节点 u 和 v 有 $var(u)=var(v)$, $low(u)=low(v)$ 并且 $high(u)=high(v)$,则删除 u 或 v,并且重定向所有指向它的连接指向另一节点.

重复上述方法,直到 BDD 的大小不再改变,则得到 BDD 的唯一规范表示(OBDD). Bryant 指出这可通过一从底向上的过程得到,并且是线性时间[14].另外,在 OBDD 中,变量赋值运算和逻辑运算都是线性时间复杂度.

3 计算树逻辑(Computation Tree Logic)

CTL*[4,17,23] 可以表示分支时间和线性时间性质,CTL* 公式由路径算子和时态算子组成.有两种路径算子,分别是 A(对于所有的路径)和 E(存在某些路径).时态算子用来描述沿着路径所具有的性质:X(下一状态),F(将来某些状态),G(总是)和 U(直到…为止).LTL 和 CTL 都包含于 CTL* 之内.

(1)如果 $p\in AP$,则 p 是一个状态公式.

(2)如果 f 和 g 是状态公式,则 $\neg f$ 和 $f\wedge g$ 是状态公式.

(3)如果 f 是一个路径公式,则 E(f)和 A(f)是状态公式.

路径公式的语法为:

(1)如果 f 是一状态公式,则 f 也是一路径公式.

(2)如果 f 和 g 是路径公式,则 $\neg f$, $f\wedge g$, Xf, Ff, Gf 及 fUg 是路径公式.

CTL* 的子逻辑 CTL 和 LTL 定义如下:

CTL:CTL(Branching Time Logic)[17] 公式是那些每个时态算子前必须紧跟有一路径算子的 CTL* 公式.

LTL:LTL(Linear Temporal Logic)[9] 公式是那些不含有路径算子 A,E 的 CTL* 公式.

一个符号化的 Kripke 结构[16] 是一个系统 $M=\langle \boldsymbol{x}, I, R\rangle$,其包含如下组成部分:

(1)$\boldsymbol{x}=\{x_1, x_2, \cdots, x_n\}$ 是一布尔变量的有限集.每个状态都被编码为对 $\boldsymbol{x}$ 的一个赋值,或者是 $\boldsymbol{x}$ 的一个子集(为了方便,我们有时候不区分一个集合和它的特征函数).所有状态组成的集合记为 S.

(2)I 是系统的初始状态集合.

(3)R 表示转换关系,其为一断言,把一个状态 $s\in S$ 和其后继状态 $s'\in S$ 联系起来.令 x' 为 x 的后继版本,则状态 s' 是状态 s 的后继当且仅当

$$\langle s, s'\rangle \models R(\boldsymbol{x}, \boldsymbol{x}'),$$

在这里 $\langle s, s'\rangle$ 是一联合解释,其中把 $x\in \boldsymbol{x}$ 解释为 $s[\boldsymbol{x}]$,把 x' 解释为 $s'[\boldsymbol{x}]$.

我们假设转换关系 R 是完全的:对于每个 $s\in S$ 有一个 $s'\in s$ 使得 $\langle s, s'\rangle \models R(\boldsymbol{x}, \boldsymbol{x}')$.有时候,我们把 $\langle s, s'\rangle \models R(\boldsymbol{x}, \boldsymbol{x}')$ 表示为 $\langle s, s'\rangle \in R$.

定义 1. 假设 $M=\langle \boldsymbol{x}, I, R\rangle$ 是一 Kripke 结构.

① M 的一条路径是一无限的状态序列 $\langle s_0, s_1, \cdots\rangle$,使得对于每个 $i\geq 0$,有 $\langle s_i, s_{i+1}\rangle \in R$ 和 $s_0 \models I$.

②如果 $r=\langle s_0, s_1, \cdots\rangle$ 是 M 的一条路径,则用 $r(i)$ 表示 s_i, r^i 表示 $\langle s_i, s_{i+1}, \cdots\rangle$.

定义 2. 给定一个 Kripke 结构 M, AP 是原子命题集,r 是一计算路径,φ 是 CTL* 公式,定义 φ 在 r 的第 i 步的可满足性(记为 $M, r, i\models\varphi$)如下:

$M, r, i\models p$, 如果 $p\in r(i)$.

$M, r, i\models \neg g_1$, 如果 $M, r, i \not\models g_1$.

$M, r, i\models g_1\wedge g_2$, 如果 $M, r, i\models g_1$ 并且 $M, r, i\models g_2$.

$M, r, i\models$ Eg_1, 如果存在一路径 r' 和自然数 j,使得 $r'(j)=r(i)$ 并且 $M, r', j\models g_1$.

$M, r, i\models$ Ag_1, 如果对于所有的路径 r',存在自然数 j,使得 $r'(j)=r(i)$,有 $M, r', j\models g_1$.

$M, r, i\models$ Xg_1, 如果 $M, r, (i+1)\models g_1$.

$M, r, i\models g_1$Ug_2, 如果存在 $k\geq i$,使得 $M, r, k\models g_2$ 且对于所有 $i\leq j<k$, $M, r, j\models g_1$.

$M, r, i\models$ Fg_1, 如果存在 $k\geq i$,使得 $M, r, k\models g_1$.

$M, r, i\models$ Gg_1, 如果对于所有 $k\geq i$, $M, r, k\models g_1$.

为方便起见,我们有时把 $M, r, i\models\varphi$ 记为 $M, r(i)\models\varphi$.

显然,算子 X,U 及 E 足以表示 CTL* 中所有路径及时态算子.

给定一个 Kripke 结构 M 和一用来表示规范的状态公式 f,则模型检测问题就是计算出 M 中那些满足公式 f 的所有状态,即集合:

$\{r(i) \mid r$ 是 M 中的路径并且 $M,r,i \models f\}$，这是一个 x 上的布尔公式.

4　关于 CTL* 的符号化模型检测

设 φ 是一可能包含时态算子的 CTL* 公式. 我们采用文献[9,15]中的所谓 tableau 构造的方法，对于一个公式 ψ，用 $\psi \in \varphi$ 表示 ψ 是 φ 的一个子(可能等于)公式. 公式 ψ 被称为主时态的如果它的主要算子是时态算子，即 ψ 是如下形式：$X\alpha$ 或 $\alpha U\beta$.

给定一个 Kripke 结构 $M=\langle \boldsymbol{x}, I, R\rangle$ 和一个公式 φ，我们定义另一个 Kripke 结构 $M_\varphi=\langle \boldsymbol{X}_\varphi, I_\varphi, R_\varphi\rangle$ 如下：

状态变量. M_φ 的状态变量 $\boldsymbol{X}_\varphi$ 包括 $\boldsymbol{x}$ 及附加的布尔变量集

$$\boldsymbol{x}_\varphi:\{x_\psi \mid \psi \text{ 是 } \varphi \text{ 的一个主时态子公式}\},$$

即 $\boldsymbol{X}_\varphi=\boldsymbol{x}\cup\boldsymbol{x}_\varphi$. 附加变量 x_ψ 在路径上某个状态为真当且仅当公式 ψ 在该状态成立.

方便起见，我们定义一函数 χ，它是把 φ 的每一个子公式映射到 $\boldsymbol{x}\cup\boldsymbol{x}_\varphi$ 上的一个布尔函数：

$$\chi(\psi)=\begin{cases}\psi, & \psi \text{ 是 } \boldsymbol{x} \text{ 中的一个变量.}\\ \neg\chi(\alpha), & \psi=\neg\alpha.\\ \chi(\alpha)\wedge\chi(\beta), & \psi=\alpha\wedge\beta.\\ x_\psi, & \psi \text{ 是主时态公式.}\end{cases}$$

令 x'_ψ 是 x_ψ 的后继版本(primed version). 对于 $\boldsymbol{x}\cup\boldsymbol{x}_\psi$ 上的公式 ψ，我们用 $\chi'(\psi)$ 来表示公式 $\psi\left(\frac{\boldsymbol{x}\cup\boldsymbol{X}_\varphi}{\boldsymbol{x}'\cup\boldsymbol{X}'_\varphi}\right)$，即 ψ 的后继版本.

初始条件. M_φ 的初始条件 I_φ 和 M 的初始条件 I 一样.

转换关系. M_φ 的转换关系 R_φ 为

$$R\wedge\bigwedge_{X\psi\in\varphi}(\boldsymbol{x}_{X\psi}\Leftrightarrow\chi'(\psi))\wedge\bigwedge_{\alpha U\beta\in\varphi}(x_{\alpha U\beta}\Leftrightarrow(\chi(\beta)\vee(\chi(\alpha)\wedge x'_{\alpha U\beta}))).$$

注 1. 我们关于公式 φ 的 tableau 构造与文献[15,16]中的有点不同. M_φ 的初始状态不一定要满足 φ，但是文献[15,16]中的 tableau 构造使得初始状态满足 φ.

方便起见，我们引进文献[24]中关于 CTL 的一些记号. 令 ψ 为 $\boldsymbol{x}\cup\boldsymbol{x}_\varphi$ 上的公式，EX 为一算子，使得：

$$EX\psi(\boldsymbol{x},\boldsymbol{x}_\varphi)=\exists(\boldsymbol{x}'\cup\boldsymbol{x}'_\varphi)(\psi(\boldsymbol{x}',\boldsymbol{x}'_\varphi)\wedge R_\varphi).$$

算子 EF 和 EU 分别定义为某个单调算子的最小不动点：

$$EFf=\mathrm{lfp}Z(f\vee EXZ)$$

和

$$EU(f,g)=\mathrm{lfp}Z(g\vee(f\wedge EXZ)).$$

令 $\mathcal{J}_\varphi$ 为公式 $\neg x_{\alpha U\beta}\vee\chi(\beta)$ 的集合，这里 $\alpha U\beta$ 是 φ 的一个子公式. 我们考察如下公平性约束(fairness constraints)：

C_φ：存在一条路径，使得 $\mathcal{J}_\varphi$ 中的每个公式在该路径上成立无限多次.

我们假设 $\mathcal{J}_\varphi$ 是一非空集，此假设不失一般性，因为我们可以把 true 加入 $\mathcal{J}_\varphi$.

约束条件 C_φ 可以被定义为一个最大不动点函数：

$$C_\varphi=\mathrm{gfp}Z\Big[\bigwedge_{J\in\mathcal{J}_\varphi}EX(EU(\mathrm{true},Z\wedge J))\Big].$$

我们称 M_φ 中的路径 r_φ 是公平的，如果 $\mathcal{J}_\varphi$ 中的每个 J 沿着路径 r_φ 被满足无限多次.

由文献[24]可知一状态满足约束条件 C_φ 当且仅当该状态在某条公平路径上，该路径使得每个 $J\in\mathcal{J}_\varphi$ 成立无限多次. 可得如下引理.

引理 1.　设 $M=\langle \boldsymbol{x}, I, R\rangle$ 为一 Kripke 结构，φ 为一可能含有时态算子的公式. $M_\varphi=\langle \boldsymbol{X}_\varphi, I_\varphi, R_\varphi\rangle$，$s_\varphi$ 是 M_φ 的一状态. 则 s_φ 满足 C_φ 当且仅当 s_φ 在 M_φ 中的公平路径上.

引理 2.　令 $M=\langle \boldsymbol{x}, I, R\rangle$ 是一 Kripke 结构，φ 为一可能包含时态算子的公式，$M_\varphi=\langle \boldsymbol{X}_\varphi, I_\varphi, R_\varphi\rangle$. 则对于 M 中的每条路径 r，存在 M_φ 中的一条公平路径 r_φ，使得对于每个自然数 u 和 φ 中的每个子公式 ψ，有

①$r(u)=r_\varphi(u)\cap\boldsymbol{x}$.

②$M,r(u)\models\psi$ 当且仅当 $M_\varphi,r_\varphi(u)\models\chi(\psi)$.

证明.　令 r 为 M 中的路径. 我们定义 M_φ 中相应的公平路径 r_φ 如下：对于每个状态 $r(u)$，令

$$r_\varphi(u)=r(u)\cup\{x_\alpha:\alpha \text{ 是 } \varphi \text{ 的主时态子公式并且 } M,r(u)\models\alpha\}.$$

则我们可立即得到 $r(u)=r_\varphi(u)\cap\boldsymbol{x}$，并且对于 φ 中的主时态子公式 ψ，有 $M,r(u)\models\psi$ 当且仅当 $M_\varphi,r_\varphi(u)\models\chi(\psi)$. 对于 φ 中别的子公式 ψ，我们可以通过归纳法证明. 显然，r_φ 是一公平路径.　证毕.

引理 3.　令 $M=\langle \boldsymbol{x}, I, R\rangle$ 是一 Kripke 结构，φ 是一可能包含时态算子的公式，$M_\varphi=\langle \boldsymbol{X}_\varphi, I_\varphi, R_\varphi\rangle$. 则对于 M_φ 中的任一公平路径 r_φ，存在 M 中的路径 r 使得对于每个自然数 u 和 φ 中的每个子公式 ψ 有

①$r(u)=r_\varphi(u)\cap\boldsymbol{x}$.

②$M,r(u)\models\psi$ 当且仅当 $M_\varphi,r_\varphi(u)\models\chi(\psi)$.

证明. 令 r_φ 为 M_φ 中的一公平路径. 我们定义 M 中的一路径 r 为:对于每个状态 $r_\varphi(u)$,

$$r(u) = r_\varphi(u) \cap \boldsymbol{x}.$$

给定 φ 的一个子公式 ψ,我们对 ψ 的结构使用归纳法证明:$M, r(u) \models \psi$ 当且仅当 $M_\varphi, r_\varphi(u) \models \chi(\psi)$.

①当 $\psi \in \boldsymbol{x}$. 根据事实 $r(u) = r_\varphi(u) \cap \boldsymbol{x}$,结论立即可得.

②当 $\psi = \neg\psi_1$ 或 $\psi = \psi_1 \wedge \psi_2$. 根据归纳假设,及 $\neg$ 和 $\wedge$ 的定义,易证结论.

③当 $\psi = \mathrm{X}\psi_1$. 根据 X 语义定义,我们有

$M, r(u) \models \psi$ 当且仅当 $M, r(u+1) \models \psi_1$,

由归纳假设可得

$M, r(u+1) \models \psi_1$ 当且仅当 $M_\varphi, r_\varphi(u+1) \models \chi(\psi_1)$,

再根据 X 的语义定义,有

$M_\varphi, r_\varphi(u+1) \models \chi(\psi_1)$ 当且仅当 $M_\varphi, r_\varphi(u) \models \mathrm{X}\chi(\psi_1)$,

由转换关系 R_φ 的定义,可得

$M_\varphi, r_\varphi(u) \models \mathrm{X}\chi(\psi_1)$ 当且仅当 $M_\varphi, r_\varphi(u) \models \chi(\psi)$.

④当 $\psi = \psi_1 \mathrm{U} \psi_2$.

($\Rightarrow$)假设 $M, r(u) \models \psi_1 \mathrm{U} \psi_2$. 则对于 $v \geq u$, $M, r(v) \models \psi_2$,并且对于所有的 $u \leq m < v$, $M, r(m) \models \psi_1$. 所以,根据归纳假设,有 $M_\varphi, r_\varphi(v) \models \chi(\psi_2)$,并且对于所有的 $u \leq m < v$, $M_\varphi, r_\varphi(m) \models \chi(\psi_1)$. 下一步我们对 m 使用递减归纳法证明:对于所有的 $u \leq m \leq v$,有 $M_\varphi, r_\varphi(m) \models \chi(\psi)$. 首先,根据转换关系 R_φ 的定义及 $M_\varphi, r_\varphi(v) \models \chi(\psi_2)$,可得 $M_\varphi, r_\varphi(v) \models \chi(\psi)$. 假设对于 $u < m \leq v$,有 $M_\varphi, r_\varphi(m) \models \chi(\psi)$. 则根据转换关系 R_φ 的定义及事实 $M_\varphi, r_\varphi(m-1) \models \chi(\psi_1)$,可得 $M_\varphi, r_\varphi(m-1) \models \chi(\psi)$. 这就完成了($\Rightarrow$)的证明.

($\Leftarrow$)假设 $M_\varphi, r_\varphi(u) \models \chi(\psi)$,但并非 $M, r(u) \models \psi_1 \mathrm{U} \psi_2$. 有两种情况:

(a) v 是一自然数,且 $v \geq u$ 使得 $M, r(v) \models \neg\psi_1$,并且对于所有的 $u \leq m \leq v$,有 $M, r(m) \models \neg\psi_2$.

(b)对于所有的 $m \geq u$,有 $M, r(m) \models \psi_1 \wedge \neg\psi_2$.

对于情况(a),根据归纳假设,对于 $v \geq u$,有 $M_\varphi, r_\varphi(v) \not\models \chi(\psi_1)$,并且对于所有的 $u \leq m \leq v$,有 $M_\varphi, r_\varphi(m) \not\models \chi(\psi_2)$. 现对 m 使用递减归纳法证明:对于所有的 $u \leq m \leq v$,有 $M_\varphi, r_\varphi(m) \not\models \chi(\psi)$. 首先,根据转换关系 R_φ 和 $M_\varphi, r_\varphi(v) \not\models \chi(\psi_1)$ 及 $M_\varphi, r_\varphi(v) \not\models \chi(\psi_2)$,有 $M_\varphi, r_\varphi(v) \not\models \chi(\psi)$. 假设对于所有 $u < m \leq v$,有 $M_\varphi, r_\varphi(m) \not\models \chi(\psi)$. 则根据转换关系 R_φ 和 $M_\varphi, r_\varphi(m-1) \not\models \chi(\psi_2)$ 及 $M_\varphi, r_\varphi(m) \not\models \chi(\psi)$,有 $M_\varphi, r_\varphi(m-1) \not\models \chi(\psi)$. 所以 $M_\varphi, r_\varphi(u) \not\models \chi(\psi)$,矛盾.

对于情况(b),根据归纳假设,对于所有的 $m \geq u$,有 $M_\varphi, r_\varphi(m) \models \chi(\psi_1)$ 并且 $M_\varphi, r_\varphi(m) \not\models \chi(\psi_2)$. 所以,再根据转换关系,可得:对于所有 $m \geq u$,有 $M_\varphi, r_\varphi(m) \models \chi(\psi)$ 当且仅当 $M_\varphi, r_\varphi(m+1) \models \chi(\psi)$. 所以,对于所有 $m \geq u$,有 $M_\varphi, r_\varphi(m) \models \chi(\psi)$ 和 $M_\varphi, r_\varphi(m) \not\models \chi(\psi_2)$. 这导致矛盾,因为 r_φ 的公平性保证有无限多个 m 使得 $r_\varphi(m)$ 满足 $\neg x_\psi \vee \psi_2$. 证毕.

下面的定理给出基于 OBDDs 的符号化验证 CTL* 公式的方法.

定理 1. 令 $M = \langle \boldsymbol{x}, I, R \rangle$ 为一 Kripke 结构,φ 为一可能包含时态算子的公式,$M_\varphi = \langle \boldsymbol{X}_\varphi, I_\varphi, R_\varphi \rangle$. 则对于 M 中的每条计算路径 r 及每个自然数 u,有

$$M, r(u) \models \mathrm{E}\varphi \Leftrightarrow \exists \boldsymbol{x}_\varphi (C_\varphi \wedge \chi(\varphi)).$$

证明. ($\Rightarrow$)假设 $M, r(u) \models \mathrm{E}\varphi$. 有一路径 r' 和一自然数 v 使得 $r'(v) = r(u)$ 并且 $M, r'(v) \models \varphi$. 根据引理 2,M_φ 中存在一公平路径 r_φ,使得 $M_\varphi, r_\varphi(v) \models \chi(\varphi)$. 并且,因为 r_φ 是一公平路径,则在其上的每个状态都必须满足 C_φ. 所以 $r_\varphi(v)$ 满足 $C_\varphi \wedge \chi(\varphi)$. 又因为 $r(u) = r'(v) = r_\varphi(v) \cap \boldsymbol{x}$,我们有 $r(u) \cup (r_\varphi(v) \cap \boldsymbol{x}_\varphi) = r_\varphi(v)$. 所以,可得 $r(u) \cup (r_\varphi(v) \cap \boldsymbol{x}_\varphi)$ 满足 $C_\varphi \wedge \chi(\varphi)$. 即 $M, r(u) \models \exists \boldsymbol{x}_\varphi (C_\varphi \wedge \chi(\varphi))$.

($\Leftarrow$)假设 $M, r(u) \models \exists \boldsymbol{x}_\varphi (C_\varphi \wedge \chi(\varphi))$. 则 $r(u)$ 满足 $\exists \boldsymbol{x}_\varphi (C_\varphi \wedge \chi(\varphi))$,并且存在 M_φ 中的一状态 s_φ 使得 $r(u) = s_\varphi \cap \boldsymbol{x}$,并且 s_φ 满足$((C_\varphi) \wedge \chi(\varphi))$. 根据事实 s_φ 满足 C_φ 以及引理 1,可得 s_φ 在 M_φ 中的某个公平路径 r_φ 上,并且存在一自然数 v 使得 $s_\varphi = r_\varphi(v)$. 根据引理 3,M 中存在一路径 r' 使得 $r'(v) = r_\varphi(v) \cap \boldsymbol{x}$,并且 $M, r', v \models \varphi$ 当且仅当 $r_\varphi(v)$ 满足 $\chi(\varphi)$. 又因为 $r_\varphi(v) = s_\varphi$ 且 $r(u) = s_\varphi \cap \boldsymbol{x}$,可得 $r'(v) = r(u)$. 另外,根据事实 $r_\varphi(v)$ 满足 $\chi(\varphi)$,可得 $M, r', v \models \varphi$. 所以 $M, r, u \models \mathrm{E}\varphi$. 证毕.

现在我们给出 CTL* 的符号化模型检测算法. 首先,我们给出 EX,EF 和 EU 的算法,它们分别记为 CheckEX,CheckEF 和 CheckEU. CheckEX 可根据下列方程计算:

$$\mathrm{CheckEX}(f(\boldsymbol{x}, \boldsymbol{x}_\varphi)) = \exists (\boldsymbol{x}' \cup \boldsymbol{x}'_\varphi)(f(\boldsymbol{x}', \boldsymbol{x}'_\varphi) \wedge R_\varphi).$$

令 BDD_And(f, g)和 BDD_Or(f, g)分别表示公式 f 和 g 的 BDDs 的并及或运算,则给出 CheckEU 的伪代码如下:

```
procedure CheckEU(f,g)
begin
  BDD a := False;
  BDD b := True;
  while a != b do
```

```
    begin
      b := a;
      a := CheckEX(a);
      a := BDD_And(f, a);
      a := BDD_Or(g, a);
    end;
    return(a);
  end;
```

CheckEF 则可根据下式得到：

$$\mathrm{CheckEF}(f) = \mathrm{CheckEU}(\mathrm{True}, f).$$

通过 CheckEX 和 CheckEU 及计算最大不动点的算法(参见文献[24])，可得到计算公平性约束 C_φ 的算法，下面给出其伪代码：

```
procedure GetConstraints()
/* 输入：J 是公平性公式集合 J_φ；输出：表示 C_φ 的 BDD */
begin
  BDD a := True;
  BDD b := False;
  BDD c;
  integer i;
  while a != b do
  begin
    b := a;
    a := True;
    for i := 1 to count of J do
    begin
      c := CheckEU(True, BDD_And(b, J[i]));
      a := BDD_And(a, CheckEX(c));
    end;
  end;
  return(a);
end;
```

定义 3.　令 g 和 g' 为 CTL* 公式，称 g' 为 g 的最大子公式，如果有以下条件成立：

①g' 是 g 的子公式；

②g' 的形式为 $\mathrm{E}g_1$ 或 g' 是一原子命题；

③g' 至少有一次不出现在 g 中的 E 的作用范围内.

例如，对于公式 $\mathrm{E}(f \vee \mathrm{E}(g \wedge \mathrm{EF}h))$，则 $\mathrm{EF}h$ 是 $\mathrm{E}(g \wedge \mathrm{EF}h)$ 的最大子公式，但不是 $\mathrm{E}(f \vee \mathrm{E}(g \wedge \mathrm{EF}h))$ 的最大子公式.

令 $M = \langle \boldsymbol{x}, I, R \rangle$ 为一 Kripke 结构，g 是一 CTL* 状态公式(如果 g 不是一状态公式，则验证它等于验证 $\mathrm{A}g$).

算法 1.　Check 函数.

输入：Kripke 结构 $M=\langle \boldsymbol{x}, I, R\rangle$ 和 CTL* 状态公式 g

输出：布尔函数 Check(g) 返回 M 中 g 在其上成立的所有状态的集合(由 BDD 表示)

1. 把 g 转化为一个除算子 E，X 和 U 外，不包含别的模态算子的 CTL* 公式，这一步可根据下列公式得到：

(a) $\mathrm{A}f \equiv \neg\mathrm{E}\neg f$.

(b) $\mathrm{F}f \equiv \mathrm{True}\mathrm{U}f$.

(c) $\mathrm{G}f \equiv \neg\mathrm{F}\neg f$.

2. 归纳地构造一个 BDD 来表示 g：

(a) 当 g 为原子命题，则返回表示 g 的 BDD.

(b) 当 $g=\neg g_1$，构造 g_1 的 BDD，并且返回该 BDD 的否定.

(c) 当 $g=g_1 \wedge g_2$，则构造出表示 g_1 的 BDD 和表示 g_2 的 BDD，然后返回它们的并.

(d) 当 $g=\mathrm{E}g_1$，这里 g_1 是一 LTL 公式，则返回 BDD：

$$\exists \boldsymbol{X}_{g_1}(C_{g_1} \wedge \chi(g_1)).$$

(e) 当 $g=\mathrm{E}g_1$，这里 g_1 不是 LTL 公式. 令 $f_1, f_2, \cdots, f_n$ 是 g_1 中最大子公式的序列，并且 $g_1=g_2(f_1, f_2, \cdots, f_n)$，则返回 BDD：

$$\mathrm{Check}(g_2(\mathrm{Check}(f_1), \mathrm{Check}(f_2), \cdots, \mathrm{Check}(f_n))).$$

注 2. 在算法 1 步 2 的(d)中，如果 g 是形式为 $\mathrm{EX}g_1$ 或 $\mathrm{E}(g_1 \mathrm{U} g_2)$ 的 CTL 公式，我们可简单地在 Kripke 结构 $\langle \boldsymbol{x}, I, R\rangle$ 上直接调用 CheckEX 或 CheckEU 来得到相应的 BDD；显然，这样使得效率更高，因为此时不需要构造 Kripke 结构 $M_g=\langle \boldsymbol{X}_g, I_g, R_g\rangle$. 我们把经过这种修改后得到的算法称为算法 2.

5　实验结果

我们已经在 NuSMV2.1.2[25] 的开放源码中实现了本文给出的 CTL* 的符号化模型检测算法. 其中使用的 OBDDs 是由美国科罗拉多大学 Somenzi 开发的 CUDD. 得到的模型检测工具我们称之为 MCTK. MCTK 向下兼容 NuSMV，在 NuSMV 的输入语言中扩展了对 CTL* 规范的支持. 这使得我们可以利用 MCTK 同时验证 CTL，LTL 和 CTL* 规范(NuSMV 仅支持 CTL 和 LTL).

我们在 NuSMV 软件包的一些实例中加入了若干 CTL* 规范并用 MCTK 加以验证，实验结果如表 1 所示. 其中第 4，6 列给出验证每个规范所用 BDD 的最大节点数. 为便于分析实验结果，我们设计一简单的实验用例，即文献[24]中给出的微波炉例子，如图 1 所示. 其中每个状态被原子命题标注，如果命题在该状态中为假，则用其否定标注；状态间的有向弧表示转换关系.

在该例子中有 4 个原子命题：*Start* 表示“微波

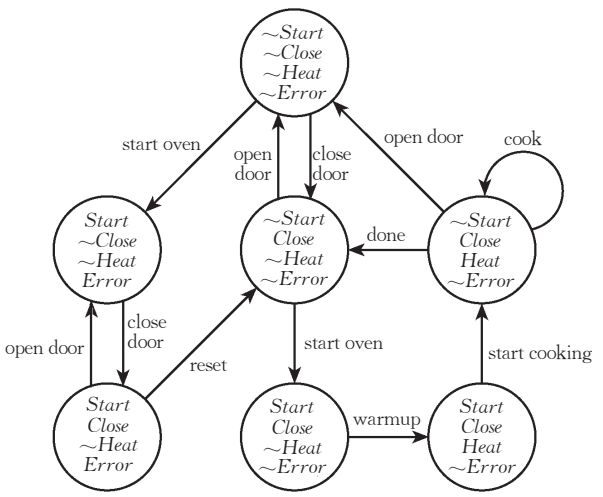

图 1 微波炉例子

炉开始加热";*Heat* 表示"微波炉加热";*Close* 表示"微波炉门被关闭";*Error* 表示"出现错误". 我们验证了该例中的一些 CTL* 规范,下面列出 3 个具有代表性的 CTL* 规范(实验结果列于表 1 中实例 oven 所处的 3 行).

1. X*Heat*. 即验证微波炉在下一状态将被加热. 等价于验证 AX*Heat*.

2. $\neg$*Heat*U*Close*. 即验证微波炉门没关加热是不可能的. 等价于验证 A($\neg$*Heat*U*Close*).

3. AG((*Close* $\wedge$ *Start*)$\rightarrow$A(G$\neg$*Heat* $\vee$ F$\neg$*Error*)). 验证:无论何时有错误出现,则微波炉要么不会加热,要么重启.

从表 1 可以看出,如果一个 CTL* 规范的整体或部分子公式是 CTL 公式,则用算法 2 计算时,所用 BDD 节点数和计算时间通常比算法 1 要少;否则算法 2 退化为算法 1,使得两算法的时空效率相同. 规范 1,2,4,5,9 的实验结果属于前一种情况,而规范 3,6,7,8,11 的属于后一种情况. 另外,用算法 2 验证规范 10 所需的时间比算法 1 少很多,但是所用 BDD 节点数比算法 1 的要多. 我们认为主要是由于 BDD 变量的排列顺序不够优化所导致的. 实际上,如果一个规范是 CTL 公式,用算法 2 得到的 BDD 和用文献[24]中的 CTL 算法得到的 BDD 是一样的. 我们通过验证表 1 中的实例证实了这一点.

表 1 实验结果

实例		CTL* 规范	结果	算法 1		算法 2	
				BDD 的节点数	时间(s)	BDD 的节点数	时间(s)
oven	1	X*Heat*	False	303	0.01	247	0.01
	2	$\neg$*Heat*U*Close*	True	403	0.01	253	0.01
	3	AG((*Close* $\wedge$ *Start*)$\rightarrow$A(G$\neg$*Heat* $\vee$ F$\neg$*Error*))	True	644	0.01	644	0.01
gigamax	4	$\neg$F(*p*0.*writable* $\wedge$ *p*1.*writable*)	True	16666	0.04	10571	0.03
	5	(G$\neg$(*p*0.*writable* $\wedge$ *p*1.*writable*)) $\wedge$ AGEF(*p*0.*readable* | *p*0.*writable*))	True	64072	0.13	24588	0.06
	6	GE(F*p*0.*readable* $\vee$ F*p*0.*writable*)	True	383479	0.80	383479	0.80
production-cell	7	AG((*s.FBM*=*on* $\wedge$ $\neg$*s.deliv*)$\rightarrow$A(F(*s.FBM*=*on* $\wedge$ *s.deliv*) $\wedge$ F(*s.botPos* $\wedge$ *s.minRot* $\wedge$ *s.TEM*=*idle* $\wedge$ *s.TRM*=*idle*)))	True	126287	0.46	126287	0.46
guidance	8	AGA($\neg$*cg.idle*$\rightarrow$F*cg.finished*)	True	329534	4.02	329534	4.02
	9	AG((*cg.idle* $\vee$ *cg.finished*)$\rightarrow$ EF($\neg$(*cg.idle* $\wedge$ *cg.finished*) $\vee$ EF*cg.finished*))	True	289370	4.32	256210	0.58
msi_wtran	10	AGEF*n*0.*c.invalid*	True	187081	85.32	195575	4.24
	11	AGE(F*n*0.*c.invalid* $\vee$ F*n*0.*c.shared*)	True	485774	393.05	485774	393.05

6 结 论

在本文中,我们给出了一个基于 OBDD 的 CTL* 符号化模型检测算法,实现了相应的符号化模型检测工具 MCTK,并验证了 NuSMV 软件包中的若干实例. 直到目前,已知的符号化模型检测工具,如 SMV 和 NuSMV 等,都只能对 CTL* 的子集(CTL,LTL 等)进行验证. 在本文中我们得到的结果表明,当系统规范不是非常复杂时,高效的 CTL* 模型检测是可能的. 关于将来的工作,我们将对该算法进行扩展,使其能够处理公平性约束(fairness constraints)和过去时态算子以及区间演算(duration calculus)[26].

参 考 文 献

1 Holzmann G. J.. The model checker SPIN. IEEE Transactions on Software Engineering, 1997, 23(5): 279~295

2 Vardi M. Y.. Branching vs. linear time: Final showdown. In: Proceedings of the 7th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, London, UK, 2001, 1~22

3 McMillan K. L.. Symbolic Model Checking. Boston: Kluwer Academic Publisher, 1993

4 Clarke E. M., Emerson E. A.. Design and synthesis of syn-

chronization skeletons using branching-time temporal logic. In: Proceedings of Logic of Programs, Workshop, Yorktown Heights, New York, 1981, 52～71

5 Queille J. P., Sifakis J.. Specification and verification of concurrent systems in CESAR. In: Proceedings of International Symposium on Programming, Torino, Italy, 1982, 337～351

6 Emerson E. A., Lei Chin-Laung. Modalities for model checking (extended abstract): Branching time strikes back. In: Proceedings of the 12th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages, New York, 1985, 84～96

7 Emerson E. A.. Branching time temporal logic and the design of correct concurrent programs[Ph. D. dissertation]. Harvard University, Cambridge, MA, 1981

8 Sistla A. P., Clarke E. M.. Complexity of propositional temporal logics. Journal of the ACM, 1986, 32(3): 733～749

9 Lichtenstein O., Pnueli A.. Checking that finite state concurrent programs satisfy their linear specification. In: Proceedings of the 12th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages, New York, 1985, 97～107

10 Godefroid P., Pirottin D.. Refining dependencies improves partial-order verification methods (extended abstract). In: Courcoubetis C. ed.. CAV, Lecture Notes in Computer Science 697. Springer, 1993, 438～449

11 Peled D.. Combining partial order reductions with on-the-fly model-checking. Formal Methods in System Design, 1996, 8(1): 39～64

12 Valmari A.. A stubborn attack on state explosion. Formal Methods in System Design, 1992, 1(4): 297～322

13 Burch J. R., Clarke E. M., McMillan K. L., Dill D. L., Hwang L. J.. Symbolic model checking: 10^{20} states and beyond. In: Proceedings of the 5th Annual IEEE Symposium on Logic in Computer Science, Washington, D. C., 1990, 1～33

14 Bryant R. E.. Symbolic Boolean manipulation with ordered binary-decision diagrams. ACM Computing Surveys, 1992, 24(3): 293～318

15 Clarke E. M., Grumberg O., Hamaguchi K.. Another look at ltl model checking. In: Proceedings of the 6th Conference on Computer Aided Verification, Stanford, California, USA, 1994, 415～427

16 Kesten Y., Pnueli A., Raviv L.. Algorithmic verification of linear temporal logic specifications. In: Larsen K. G., Skyum S., Winskel G. eds.. ICALP, Lecture Notes in Computer Science 1443. Springer, 1998, 1～16

17 Clarke E. M., Emerson E. A., Sistla A. P.. Automatic verification of finite state concurrent systems using temporal logic specifications: A practical approach. In: Proceedings of the 10th Annual ACM Symposium on Principles of Programming Language, Austin, Texas, 1983, 117～126

18 Burch J. R., Clarke E. M., Long D. E.. Symbolic model checking with partitioned transition relations. In: Proceedings of the International Conference on Very Large Scale Integration, Edinburgh, Scotland, 1991, 49～58

19 Burch J. R., Clarke E. M., Long D. E., MacMillan K. L., Dill D. L.. Symbolic model checking for sequential circuit verification. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 1994, 13(4): 401～424

20 Bhat G., Cleaveland R., Grumberg O.. Efficient on-the-fly model checking for CTL*. In: Proceedings of the 10th Annual IEEE Symposium on Logic in Computer Science, San Diego, California, USA, 1995, 388～397

21 Bernholtz O., Vardi M. Y., Wolper P.. An automata-theoretic approach to branching-time model checking (extended abstract). In: Dill D. L. ed.. CAV, Lecture Notes in Computer Science 818. Springer, 1994, 142～155

22 Kesten Y., Pnueli A.. A compositional approach to CTL* verification. Theoretical Computer Science, 2005, 331(2,3): 397～428

23 Emerson E. A., Halpern J. Y.. "sometimes" and "not never" revisited: On branching versus linear time temporal logic. Journal of the ACM, 1986, 33(1): 151～178

24 Grumberg O., Clarke E. M., Peled D. A.. Model Checking. Cambridge, MA: The MIT Press, 2000

25 Cimatti A., Clarke E. M., Giunchiglia E., Giunchiglia F., Pistore M., Roveri M., Sebastiani R., Tacchella A.. NuSMV Version 2: An OpenSource tool for symbolic model checking. In: Proceedings of the International Conference on Computer-Aided Verification (CAV 2002), Copenhagen, Denmark, 2002, 359～364

26 Zhou Chao-Chen, Hoare C. A. R., Ravn Anders P.. A calculus of durations. Information Processing Letters, 1991, 40(5): 269～276

SU Kai-Le, born in 1964, professor, Ph. D. supervisor. His research interests include model checking, reasoning about knowledge, non-monotonic reasoning, multi-agent system, modal logic, temporal logic, probability reasoning, verification of security protocol, logic programming.

LUO Xiang-Yu, born in 1974, Ph. D. candidate. His research interests include model checking, temporal logic, modal logic, reasoning about knowledge, multi-agent system, verification of security protocol, logic programming.

LU Guan-Feng, born in 1973, Ph. D. candidate. His research interests include model checking, reasoning about knowledge, multi-agent system, modal logic, temporal logic, probability reasoning, verification of security protocol, logic programming.

Background

In the past 15 years, there was many works on symbolic model checking for the sublogics of CTL*. For the branching-time temporal logic CTL, McMillan *et al*. realized the original CTL model checking algorithm of Clarke and Emerson with the OBDDs representation for state transition graphs, and thus it is possible to verify some examples that had more than 10^{20} states, even 10^{120} states by using various refinements of the OBDD-based techniques by other researchers.

For symbolic LTL model checking, Clarke, Grumberg and Hamaguchi gave a way to reduce LTL model checking to CTL model checking with fairness constraints, constructed asymbolic LTL model checker using the reduction, and also showed how SMV can be extended to permit LTL specifications. Kesten, Pnueli and Raviv gave an interesting framework for symbolic model checking specifications oflinear-time temporal logic LTL with full fairness into consideration, and it was self-contained exposition of full LTL symbolic model checking without resorting to reductions to either CTL or automata.

There was also some model checking algorithms for the full CTL*. Bhat, Cleave land and Grumberg presented an efficient on-the-fly algorithm for model checking CTL*. Bernholtz, Vardi and Wolper suggested another model-checking algorithm for the full CTL*, which allows the on-the-fly construction of the state space of a system. However, these algorithms are not symbolic ones for CTL* model checking. Recently, Kesten and Pnueli presented a compositional approach to the verification of CTL* properties over reactive systems. Both symbolic model-checking and deductive verification are considered. The model-checking part is similar to the symbolic model checking algorithm of this paper. But they didn't present the proofs and experimental results in detail.

This paper gives a symbolic model checking algorithm for CTL*. The time complexity of the algorithm matches that of the existing algorithms for model checking CTL*, or its sublogics CTL and LTL. Authors also describe how to construct a new CTL* symbolic model checker (MCTK) that appears to be efficient in practice. For the specifications which are CTL, the CTL* model checker get the same results as the CTL model checker.