软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software, 2022,33(8):2930–2946 [doi: 10.13328/j.cnki.jos.006606]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

基于时态测试器的实时分支时态逻辑模型检测*

骆翔宇[1,2], 黄欣玥[1], 古天龙[2,3], 苏开乐[4], 陈祖希[1], 郑黎晓[1]

[1](华侨大学 计算机科学与技术学院, 福建 厦门 361021)

[2](广西可信软件重点实验室(桂林电子科技大学), 广西 桂林 541004)

[3](暨南大学 信息科学技术学院/网络空间安全学院, 广东 广州 510632)

[4](南京信息工程大学 人工智能学院, 江苏 南京 210044)

通信作者: 郑黎晓, E-mail: zheng_lixiao@163.com

摘 要: 基于自动机理论的模型检测技术在形式化验证领域处于核心地位, 然而传统自动机在时态算子上不具备可组合性, 导致各种时态逻辑的模型检测算法不能有机整合. 为了实现集成限界时态算子的实时分支时态逻辑RTCTL*的高效模型检测, 提出一种 RTCTL*正时态测试器构造方法以及相关符号化模型检测算法, 既证明了所提出的 RTCTL*正时态测试器构造方法是完备的, 也证明了该算法时间复杂度与被验证系统呈线性关系, 与公式长度呈指数关系. 基于 JavaBDD 软件包成功开发了该算法的模型检测工具 MCTK 2.0.0. 完成了 MCTK 与著名的符号化模型检测工具 nuXmv 之间的实验对比分析工作, 结果表明: MCTK 虽然在内存消耗上要多于 nuXmv, 但是 MCTK 的时间复杂度双指数级小于 nuXmv, 使得利用 MCTK 验证大规模系统的实时时态性质成为可能.

关键词: 符号化模型检测; 公平离散系统; 正时态测试器; 实时分支时态逻辑; 二元决策图

中图法分类号: TP311

中文引用格式: 骆翔宇, 黄欣玥, 古天龙, 苏开乐, 陈祖希, 郑黎晓. 基于时态测试器的实时分支时态逻辑模型检测. 软件学报, 2022, 33(8): 2930–2946. http://www.jos.org.cn/1000-9825/6606.htm

英文引用格式: Luo XY, Huang XY, Gu TL, Su KL, Chen ZX, Zheng LX. Model Checking for Real-time Branching-time Temporal Logic Based on Temporal Testers. Ruan Jian Xue Bao/Journal of Software, 2022, 33(8): 2930–2946 (in Chinese). http://www.jos.org.cn/1000-9825/6606.htm

Model Checking for Real-time Branching-time Temporal Logic Based on Temporal Testers

LUO Xiang-Yu[1,2], HUANG Xin-Yue[1], GU Tian-Long[2,3], SU Kai-Le[4], CHEN Zu-Xi[1], ZHENG Li-Xiao[1]

[1](College of Computer Science and Technology, Huaqiao University, Xiamen 361021, China)

[2](Guangxi Key Laboratory of Trusted Software (Guilin University of Electronic Technology), Guilin 541004, China)

[3](College of Information Science and Technology/College of Cyber Security, Jinan University, Guangzhou 510632, China)

[4](School of Artificial Intelligence, Nanjing University of Information Science & Technology, Nanjing 210044, China)

Abstract: Model checking techniques based on automata theory play a central role in formal verification. Nevertheless, classical automata are not composable for temporal operators, such that the model checking algorithms for various temporal logics cannot be organically integrated. To realize efficient model checking for real-time branching-time temporal logic RTCTL* integrating bounded temporal operators, a construction method is proposed for RTCTL* positive temporal testers, and the RTCTL* symbolic model checking algorithm is further proposed based on positive temporal testers. It is proved that the proposed construction method for RTCTL* positive temporal testers is complete. It is also proved that the time complexity of the proposed algorithm is linear in the size of the verified system and exponential in the length of the given formula. Based on the JavaBDD software package, the model checking tool MCTK 2.0.0 is

* 基金项目: 国家自然科学基金(U1711263, 1966009, 62006057, 61170028); 福建省自然科学基金(2021J01316, 2021J01320, 2015J01255); 广西可信软件重点实验室研究课题(kx201323)

本文由“形式化方法与应用”专题特约编辑陈立前副教授、孙猛教授推荐.

收稿时间: 2021-09-06; 修改时间: 2021-10-14; 采用时间: 2022-01-10; jos 在线出版时间: 2022-01-28

successfully developed for the proposed algorithm. The experimental data and the analysis results show that although MCTK consumes more memory than the famous symbolic model checker nuXmv, the time complexity of MCTK is doubly-exponentially less than nuXmv, which makes it possible to verify real-time temporal properties of large-scale systems by MCTK.

Key words: symbolic model checking; just discrete system; positive temporal tester; real-time branching-time temporal logic; binary decision diagram

反应式系统的形式化验证是计算机科学领域的主要技术挑战, 而基于自动机理论的模型检测技术则在形式化验证领域处于中心地位[1]. 例如, 对于一个 LTL 线性时态逻辑公式φ在有限状态系统 S 上是否满足这样一个模型检测, 传统的方法是先后构建一个刻画 S 的自动机 A_S 和一个接受所有违反公式φ语义的状态序列的ω-自动机$A_{\neg\varphi}$, 然后检查乘积自动机$A_S\times A_{\neg\varphi}$是否接受空语言: 如果不接受空语言, 则乘积自动机的一条接受状态序列即为违反公式φ的一个反例; 否则, 系统 S 满足公式φ. 此类方法多数采用ω-自动机 $A_{\neg\varphi}$这类接受$\mathcal{L}(\neg\varphi)$语言的接受器[2].

然而, 随着模型检测技术的持续进步, 自动机这类语言接受器也逐渐呈现出一些不足: 首先, 现有的与新提出的时态逻辑及其模型检测技术需要根据应用需求进行灵活而有机地整合, 形成满足应用需求的、有更精密复杂的时态表达力的新时态逻辑及其模型检测技术. 比如, 符合 IEEE 工业级标准的性质规约语言 PSL[3]和 SystemVerilog 断言 SVA[4]即为这类时态逻辑的典型代表. PSL 和 SVA 的时态部分是通过在线性时态逻辑 LTL 中扩展正则表达式和后缀蕴涵算子得到的, 使其表达力从无星号ω正则语言扩展到ω正则语言[2]. PSL 还支持基于 CTL 的分支时态逻辑. 对于 PSL 和 SVA 这类由多种基础子逻辑相互组合构成的时态逻辑, 其中各个基础子逻辑相应的模型检测算法必须是可组合的, 才能自底向上组合构造出这类复杂时态逻辑的模型检测算法. 而自动机这类语言接受器并不能相互组合. 比如给定两个 LTL 公式 f 和 g 及其自动机 A_f 和 A_g, 将这两个自动机进行简单组合并不能构造出复合时态公式(如公式 fUg, 表示 f 成立直至 g 成立)的自动机[5].

除了提升表达力外, 时态逻辑的简洁性和易用性对于用户(尤其是工业领域的用户)应用来说也是很重要的. 再以 PSL 逻辑为例, PSL 为时态逻辑用户的频繁使用模式提供句法支持, 其中一个重要的句法支持是计数[2], 即在 LTL 时态算子上附加离散区间, 表示对相应时态算子的离散时间约束. 比如, 限界 Until 算子 $fU_{[a,b]}g$ 的含义是: g 在第 a 迁移步至第 b 迁移步之间成立, 在此之前 f 一直成立. 限界 Until 算子 $fU_{[a,b]}g$ 可变换为等价的不带区间的 LTL 公式, 因此这种扩展了时间区间的 LTL(称为 RTLTL[6])与 LTL 的时态表达力相同, 但是 RTLTL 的简洁性指数级优于 LTL, 使得 RTLTL 以及基于 RTLTL 的 PSL 有很好的易用性. 同样地, 复合的限界时态公式(如 $fU_{[a,b]}g$)的自动机亦不能由子式 f 和 g 的自动机简单地组合而成.

时态测试器(temporal tester, 简称测试器)是一种特殊的自动机, 它不仅能像传统自动机那样评估一个完整的(无限)输入序列是否在其刻画的语言中, 也能评估该输入序列的任意后缀是否在该语言中. 文献[7]提出了 LTL 的时态测试器, 在一个 LTL 公式φ的语法树中自底向上为每一主时态子式ψ构造一个子测试器(可视为非确定变换器), 其中包含代表该子式的新建布尔输出变量 x_ψ. 该子测试器持续观察输入状态序列任意位置起始的后缀序列, 并输出 x_ψ=1 当且仅当该后缀序列满足子式ψ. 通过同步并行组合所有子式的子测试器构成原式φ的时态测试器 T_φ, 并将φ中的主时态子式替换为其对应的布尔输出变量, 得到不含时态算子的命题公式 $\chi(\varphi)$, 则系统 D 是否满足原式φ的模型检测问题可转变为同步并行组合系统 $D\|T_\varphi$是否满足$\chi(\varphi)$的模型检测问题. 显然, 这类时态测试器自然是可以自底向上进行组合的. 时态测试器另一先天优势是可以被符号化(基于二元决策图 BDD[8])表示, 而测试器之间的同步并行组合也能够通过 BDD 简单高效地实现.

此后, 随着符号化模型检测技术[1]在工业界的成功应用, Pnueli 等人进一步提出了 PSL 和 MITL[9]时态逻辑的时态测试器构造方法和相关模型检测算法[5], 其中, 实时时态逻辑 MITL 与 RTLTL 和 PSL 类似, 在 LTL 时态算子中引入了有理数(包含整数)区间. 经我们分析, MITL 时态测试器的构造复杂度很高.对于 $\varphi=fU_{[a,b]}g$(假设子式 f 和 g 为命题, $[a,b]$为整数区间), 虽然其测试器的输出变量仍然是一个布尔变量, 但是实际上总共需要新建 3 个布尔变量、1 个域为$[0,b-a]$的整型变量以及 $2k$ 个域为$[0,a]$的整型变量, 其中, $k<a$ 且是不确定的. 因此, φ测试器不仅构造复杂度很高, 而且也不可实现. 另外, PSL 时态测试器并没有考虑区间约束.

由于 LTL 测试器为每一时态算子新建一个布尔变量, 使得基于测试器的 LTL 模型检测算法时间复杂度与公式长度呈指数关系, 已达到 LTL 模型检测的时间复杂度下界. 因此时至今日, 大多数基于 BDD 的符号化模型检测工具(如 NuSMV 2.6.0[10], nuXmv 2.0.0[11], MCMAS 1.3.0[12]和我们的 MCTK 1.0.0[13])的 LTL 模型检测算法仍然是基于 A. Pnueli 的 LTL 测试器实现的. 虽然 NuSMV 和 nuXmv 支持 PSL 的全部语法,但是其模型检测算法只能验证与 RTLTL 或 RTCTL[1]等价的 PSL 子集. 而且奇怪的是, NuSMV 2.6.0 的 RTLTL 不支持关键的限界时态算子 $U_{[a,b]}$, 因此简洁性较低; 而 nuXmv 2.0.0 则弥补了这一不足. 另外, MCMAS 和我们的 MCTK 1.0.0 不仅支持 LTL, 而且还支持时态表达力更强的 CTL*[2], 但是 MCMAS 和 MCTK 1.0.0 的 CTL*仍然没有扩展支持区间.

根据上述分析, 我们自然地期望模型检测工具不仅能够通过有更强表达力的传统时态逻辑 CTL*“定性地”分析系统, 而且能够在时态算子上引入区间约束来“定量地”分析系统. 从应用的角度看, 对于在分布式和实时环境下的关键系统(比如网络通信协议和嵌入式实时控制系统等)来说, 其时间攸关的实时性质必须用兼具定性和定量时态表达能力的时态逻辑进行刻画和验证. 为此, 我们在文献[6]中提出一种实时分支时态逻辑 RTCTL*及其模型检测算法, 通过在 CTL*的将来和过去时态算子上附加离散区间, 使得 RTCTL*的时态表达力强于 LTL, CTL, RTLTL 和 RTCTL. 这意味着 RTCTL*的时态表达力强于目前许多基于 LTL 和 CTL 的实时时态逻辑和相关工具(基于 LTL 的实时时态逻辑有 MITL, MTL[14]和 TPTL[15]等; 基于 CTL 的工具有 Uppaal[16], HyTech[17], Kronos[18]和 FSMT-MC[19]等).

我们在文献[6]中还开发了将 RTLTL 时态测试器转化为 NuSMV 输入语言的翻译方法, 可基于 NuSMV 实现 RTLTL 的模型检测和反例生成, 其中, 限界 Until 算子 $U_{[0,b]}$和限界 Since 过去算子 $S_{[0,b]}$模型检测算法的时间复杂度指数级小于 NuSMV 和 nuXmv 的相应算法. 但是该翻译方法对于区间下界 $a>0$ 的时态算子仍然需要将公式展开为区间下界为 0 的等价公式, 其中包含 a 个新引入的下一步时态算子, 致使相应模型检测算法与 NuSMV 和 nuXmv 的一致, 其时间复杂度退化为与 a 呈指数关系.

由上述模型检测工具可知, 目前已有的 RTLTL 和 RTCTL*模型检测算法实现的性能仍然非常低. 总体来说, 这些算法和工具不具备大规模工业应用的潜力. 为了显著提升 RTCTL*模型检测性能, 本文提出并实现一种基于正时态测试器的 RTCTL*符号化模型检测算法, 实验比较结果也证实了该算法的高效性. 本文的主要贡献如下:

(1) 提出了 RTCTL*正时态测试器及其构造方法, 给出了 RTCTL*正时态测试器构造的完备性证明;
(2) 提出了基于正时态测试器的 RTCTL*符号化模型检测算法;
(3) 给出了基于RTCTL*正时态测试器的构造复杂度和相应模型检测算法的时间复杂度证明, 结果表明: 该算法时间复杂度与被验证系统呈线性关系, 与公式长度呈指数关系;
(4) 实现了基于正时态测试器的 RTCTL*符号化模型检测工具 MCTK 2.0.0;
(5) 通过案例进行 nuXmv 与 MCTK 2.0.0 的实验比较和分析, 结果表明: 虽然基于 Java 开发的 MCTK 2.0.0 在内存消耗上要多于基于 C 和 C++开发的 nuXmv, 但是 MCTK 2.0.0 的时间复杂度双指数级小于 nuXmv. 因此, 本文的 RTCTL*高效模型检测算法具备了工业化应用的潜能.

本文第 1 节介绍公平离散系统模型和 RTCTL*时态逻辑. 第 2 节提出 RTCTL*正时态测试器定义及其构造方法. 第 3 节提出基于 RTCTL*正时态测试器的符号化模型检测算法. 第 4 节给出 RTCTL*正时态测试器构造的完备性证明. 第 5 节给出基于 RTCTL*正时态测试器的模型检测算法时间复杂度证明. 第 6 节介绍 MCTK 2.0.0 的工具实现, 及其与 nuXmv 的实验比较和分析. 最后, 在第 7 节给出本文结论和展望.

1 预备知识

1.1 公平离散系统

首先定义公平离散系统形式模型, 其中支持弱公平性约束.

定义 1(**公平离散系统**[2]). 公平离散系统(JDS)是一个四元组 $D=(V,\theta,R,J)$:

- V 是有限论域上变量的有限集合$\{v_1,\ldots,v_n\}$. V 中变量的一个赋值表示系统中的一个状态, 因此 V 上的所有赋值构成系统的可能状态集合 S. 我们用 $s[v]$表示状态 s 中变量 v 的值;
- θ是 V 上的初始条件断言, 满足θ的状态为系统的初始状态;
- $R(V,V')$是 $V\cup V'$上的一个断言, 用于刻画状态迁移关系, 其中, V'是 V 的副本, V 和 V'分别用于刻画当前状态和直接后继状态. 状态 s'是状态 s 的一个直接后继当且仅当 $R(s,s')$为真, 记为$(s,s')\in R$;
- $J=\{j_1,\ldots,j_m\}$是弱公平性约束集合, 其中元素为 V 上的断言.

对于一个公平离散系统 D, 设有一个状态序列 $r:s_0,s_1,s_2,s_3,\ldots$表示一条路径, 其中每一对相邻状态满足$(s_i,s_{i+1})\in R$. 我们称路径 r 是公平的当且仅当对于任意 $j\in J$, r 中包含无穷多个满足 j 的状态. 我们用 $r(i)$表示状态 s_i, 用 r^i 表示 r 的起始于 $r(i)$的后缀路径. 如果 $r(0)$是 D 的一个初始状态, 则路径 r 也称为 D 的一次运行. 如果运行 r 是公平的, 则 r 也称为 D 的一次计算. 我们用$\Pi(D)$表示系统 D 的所有计算的集合. 我们定义两个 JDS D_1 和 D_2 的同步并行组合系统 $D_1\|D_2=(V_1\cup V_2,\theta_1\wedge\theta_2,R_1\wedge R_2,J_1\cup J_2)$.

1.2 RTCTL*语法

实时分支时态逻辑RTCTL*可同时在计算树和线性计算路径上解释公式语义, 是在计算树逻辑CTL*基础上扩展离散实时时态算子得到的. 给定有限域上的类型变量集合 V(包含有限整数类型和有限枚举类型), 我们定义 L 为 V 上的一阶断言语言, 其中包含有限整数类型变量上的常规解释符号(如整数的算术和关系运算符以及枚举类型的集合运算符). RTCTL*建立在 L 之上, 包含布尔运算符、(实时)时态算子以及路径量词.

RTCTL*包含两种类型的公式: 状态公式和路径公式. 状态公式是在计算树上解释的, 而路径公式则是在线性计算路径上解释的. 设φ为状态公式, ψ为路径公式, 则 RTCTL*为由如下语法规则定义的状态公式集合:

$$\varphi ::= p\mid\neg\varphi\mid\varphi\wedge\varphi\mid\varphi\vee\varphi\mid E\psi\mid A\psi$$

$$\psi ::= \varphi\mid\psi\wedge\psi\mid\psi\vee\psi\mid X\psi\mid\psi U\psi\mid\psi R\psi\mid\psi U_{[a,b]}\psi\mid\psi R_{[a,b]}\psi$$

其中, p 是一个 V 上的断言, E 和 A 分别是存在路径量词和全称路径量词, X,U 和 R 是 LTL 时态算子. $U_{[a,b]}$和 $R_{[a,b]}$是带闭区间的实时时态算子, 其中, a 和 b 为非负整数且 $0\leqslant a\leqslant b$. 假设 TRUE 和 FALSE 分别用符号$\top$和$\bot$表示, f 和 g 为两个 RTCTL*路径公式, 则其他扩展的时态算子 F(最终)和 G(始终)可由如下等式定义: $Ef=\top Uf$, $Gf=\neg F\neg f$, $F_{[a,b]}f=\top U_{[a,b]}f$, $G_{[a,b]}f=\neg F_{[a,b]}\neg f$, $fU_{[a,\infty)}g=fU_{[a,a]}(fUg)$和 $fR_{[a,\infty)}g=fR_{[a,a]}(fRg)$等. 下文统一用$\varphi$表示状态公式, 用$\psi$, f 或 g 表示路径公式. 需要注意的是, 任意状态公式同时也是路径公式. 因此, 如果下文出现的路径公式ψ, f 或 g 满足状态公式的语法定义, 则它们同时也是状态公式.

我们用 $\psi_1\in\psi$表示 ψ_1 是 ψ的一个子式. 如果一个公式的主算子是时态算子, 则该公式称为主时态(principally temporal)公式. 对于ψ中的一个以路径量词为主算子的子式ψ_1, 如果 ψ_1 不是其他同类型(以路径量词为主算子)子式的真子式, 则称 ψ_1 是 ψ的一个最大状态子式. 为了便于描述 RTCTL*模型检测算法, 我们定义函数 $vars(\psi)$表示 ψ所依赖的变量集合.

1.3 RTCTL*语义

给定一个 JDS $D=(V,\theta,R,J)$, 一条公平路径 $r=s_0,s_1,s_2,\ldots$, 一个 RTCTL*公式φ, 假设路径公式 $f,g\in\varphi$, 则在公式结构上归纳定义φ在 D 的后缀路径 $r^i(i\geqslant 0)$上的可满足关系(记为$(D,r,i)\models\varphi$)如下.

- $(D,r,i)\models p$ iff p 是 V 上的一个断言, 并且 p 在状态 $r(i)$上为真.
- $(D,r,i)\models\neg f$ iff $(D,r,i)\not\models f$.
- $(D,r,i)\models f\wedge g$ iff $(D,r,i)\models f$ 并且$(D,r,i)\models g$.
- $(D,r,i)\models f\vee g$ iff $(D,r,i)\models f$ 或者$(D,r,i)\models g$.
- $(D,r,i)\models Xf$ iff $(D,r,i+1)\models f$.
- $(D,r,i)\models fUg$ iff 存在 $i'\geqslant i$, 使得$(D,r,i')\models g$, 并且对所有 $i''\in[i,i'-1]$有$(D,r,i'')\models f$.
- $(D,r,i)\models fRg$ iff 对所有 $i'\geqslant i$, 如果对所有 $i''\in[i,i'-1]$有$(D,r,i'')\models\neg f$, 则$(D,r,i')\models g$.
- $(D,r,i)\models fU_{[a,b]}g$ iff 存在 $i'\in[i+a,i+b]$使得$(D,r,i')\models g$, 并且对所有 $i''\in[i,i'-1]$有$(D,r,i'')\models f$.

- $(D,r,i)\models fR_{[a,b]}g$ iff 对所有 $i'\in[i+a,i+b]$, 如果对所有 $i''\in[i,i'-1]$有$(D,r,i'')\models\neg f$, 则$(D,r,i')\models g$.
- $(D,r,i)\models Ef$ iff 存在 $r'\in\Pi(D)$和某一 $i'\geqslant 0$, 使得 $r'(i')=r(i)$并且$(D,r',i')\models f$.
- $(D,r,i)\models Af$ iff 所有 $r'\in\Pi(D)$和 $i'\geqslant 0$, 有 $r'(i')=r(i)$蕴涵$(D,r',i')\models f$.

注意: 如果φ是一个 RTCTL*状态公式, 则$(D,r,i)\models\varphi$也可以写为$(D,r(i))\models\varphi$, 即φ在 D 的状态 $r(i)$上为真. JDS D 满足 RTCTL*状态公式φ(记为 $D\models\varphi$)当且仅当对所有满足初始条件θ的状态 s 有$(D,s)\models\varphi$.

2 RTCTL*正时态测试器及其构造方法

2.1 RTCTL*正时态测试器

对一个主时态公式ψ, 其全时态测试器(full temporal tester)是一个新的 JDS, 其中包含为该公式创建的布尔输出变量 x_ψ, 通过其迁移确保对任意公平路径的任意后缀路径 r^i, 有 $(T_\psi^{full},r,i)\models\psi$ 当且仅当 $(T_\psi^{full},r,i)\models x_\psi$. 因此, 对于复合 RTCTL*公式$\varphi$, 可通过将其中的主时态子式替换为相应布尔输出变量的方式, 将 $D\models\varphi$模型检测问题转化为 $D\,\|T_\varphi^{full}\models\varphi'$ 模型检测问题, 其中, φ'为将φ中的主时态子式替换为相应布尔输出变量的状态公式. 引言中提到的文献[5−7,10−13]采用的都是全时态测试器.

实际上, 全时态测试器同时包含正时态测试器(x_ψ蕴涵ψ)和负时态测试器($\neg x_\psi$蕴涵$\neg\psi$). 由于 LTL 语义简单, 因此构造 LTL 全时态测试器是容易的. 但是对于 RTCTL*, 构造其全时态测试器是相对困难的. 原因在于 RTCTL*包含区间, 简单的布尔输出变量无法表示一个限界的主时态公式. 另一方面, 对于模型检测来说, 如果一个公式中没有时态算子被否定词约束(称为否定范式), 则只需构造该公式的正时态测试器就足够了. 因此, 我们将扩展定义正时态测试器, 将布尔输出变量扩展为输出断言.

对于一个 RTCTL*路径公式ψ, 我们首先定义函数 $nnf(\psi)$, 将ψ转换为它的否定范式. 函数 nnf 遵循以下转换规则: $\neg\neg f\equiv f$, $\neg(f\wedge g)\equiv\neg f\vee\neg g$, $\neg Xf\equiv X\neg f$, $\neg(fRg)\equiv\neg fU\neg g$, $\neg(fR_{[a,b]}g)\equiv\neg fU_{[a,b]}\neg g$ 和$\neg(fR_{[a,\infty)}g)\equiv\neg fU_{[a,\infty)}\neg g$. 设$\psi$是否定范式, 我们定义如下函数$\chi(\psi)$为公式$\psi$的输出断言:

$$\chi(\psi)=\begin{cases}\psi, & \psi\text{是断言或断言的否定式}\\ \chi(f)\wedge\chi(g), & \psi=f\wedge g\\ \chi(f)\vee\chi(g), & \psi=f\vee g\\ x_\psi,\psi=Xf, & fUg\text{或}fRg\\ x_\psi\wedge l_\psi=a\wedge w_\psi=b-a, & \psi=fU_{[a,b]}g\text{或}fR_{[a,b]}g\text{并且}0<a<b\\ x_\psi\wedge l_\psi=a, & \psi=fU_{[a,a]}g,\ fU_{[0,a]}g,\ fR_{[a,a]}g,\ fR_{[0,a]}g\text{并且}a>0\\ \chi(g), & \psi=fU_{[0,0]}g\text{或}fR_{[0,0]}g\\ [\![\psi]\!], & \psi=Ef\text{或}Af\end{cases}\tag{1}$$

其中的 x_ψ, l_ψ, w_ψ是为对应公式新建的变量, x_ψ是布尔输出变量, $l_\psi\in[0,a]$是表示区间下界的整型变量, $w_\psi\in[0,b-a]$是表示区间宽度的整型变量. 例如, $\psi=fU_{[a,b]}g(0<a<b)$对应的输出断言为 $x_\psi\wedge l_\psi=a\wedge w_\psi=b-a$. 另外, 对于路径量词约束的子式$\psi$, 表示满足该式的状态集合.

定义 2(RTCTL*正时态测试器(positive temporal tester)). 设ψ是一个 RTCTL*路径公式的否定范式, ψ的正时态测试器 T_ψ(简称测试器)满足如下条件.

- 可靠性: 对于 T_ψ中任意公平路径ρ和任意位置 $i\geqslant 0$, 如果$(T_\psi,\rho(i))\models\chi(\psi)$, 则$(T_\psi,\rho,i)\models\psi$;
- 完备性: 若 $D=(V,\theta,R,J)$是一个 JDS, 对于 D 上的任意公平路径 r 和任意位置 $i\geqslant 0$, T_ψ中必有一条公平路径ρ, 其上所有状态与 r 上对应位置的状态有相同的 $vars(\psi)$变量解释, 使得$(T_\psi,\rho(i))\models\chi(\psi)$当且仅当$(D,r,i)\models\psi$.

2.2 LTL正时态测试器构造

RTCTL*包含 LTL, 因此我们首先构造 LTL 的正时态测试器如下.

2.2.1 *Xf* 正时态测试器构造

设$\psi=Xf$, ψ对应的正时态测试器$T_\psi=(V_\psi,\theta_\psi,R_\psi,J_\psi)||T_f$, 其中: 变量集合$V_\psi:=\{x_\psi\}\cup vars(f)$, x_ψ为测试器的输出变量; 初始条件$\theta_\psi:=\top$; 迁移关系 $R_\psi:=x_\psi\to\chi'(f)$, 其中, $\chi'(f)$为将$\chi(f)$所依赖变量替换为相应副本后得到的输出断言; 公平性约束集合$J_\psi:=\varnothing$.

2.2.2 *fUg* 正时态测试器构造

设$\psi=fUg$, 根据扩展式 $fUg=g\vee(f\wedge X(fUg))$构造$\varphi$的正时态测试器 $T_\psi=(V_\psi,\theta_\psi,R_\psi,J_\psi)||T_f||T_g$, 其中: 变量集合$V_\psi:=\{x_\psi\}\cup vars(\chi(f))\cup vars(\chi(g))$, x_ψ为测试器的输出变量; 初始条件$\theta_\psi:=x_\psi\to(\chi(f)\vee\chi(g))$; 迁移关系 $R_\psi:=x_\psi\to(\chi(g)\vee(\chi(f)\wedge x'_\psi))$, 其中, x'_ψ 为 x_ψ的副本; 公平性约束集合 $J_\psi:=\{x_\psi\to\chi(g)\}$. 由于迁移关系 R_ψ不能确保$\chi(g)$在将来的某一状态中成立, 因此通过增加公平性约束$x_\psi\to\chi(g)$, 可排除那些使得x_ψ永远成立而$\chi(g)$永远不成立的非公平路径. 因此, 保留下来的路径都是满足 *fUg* 语义的公平路径.

2.2.3 *fRg* 正时态测试器构造

设$\psi=fRUg$, 根据扩展式 $fRg=\neg(\neg fU\neg g)=g\wedge(f\vee X(fRg))$构造$\psi$的正时态测试器 $T_\psi=(V_\psi,\theta_\psi,R_\psi,J_\psi)||T_f||T_g$, 其中: 变量集合$V_\psi:=\{x_\psi\}\cup vars(f)\cup vars(g)$, x_ψ为测试器的输出变量; 初始条件$\theta_\psi:=\top$; 迁移关系$R_\psi:=x_\psi\to(\chi(g)\wedge(\chi(g)\vee x'_\psi))$; 公平性约束集合$J_\psi:=\varnothing$.

2.3 限界Until时态算子的正时态测试器构造

本节根据如下扩展式构造限界 Until 时态算子的正时态测试器:

$$fU_{[a,b]}g=\begin{cases}f\wedge X(fU_{[a-1,b-1]}g), & \text{如果}0<a\leqslant b\\ g\vee(f\wedge X(fU_{[0,b-1]}g)), & \text{如果}0=a<b\\ g, & \text{如果}0=a=b\end{cases}\tag{2}$$

2.3.1 $fU_{[a,b]}g$ 正时态测试器构造

设$\psi=fU_{[a,b]}g$ 并且 $0<a<b$, ψ的正时态测试器 $T_\psi=(V_\psi,\theta_\psi,R_\psi,J_\psi)||T_f||T_g$ 定义如下.

- $V_\psi:=\{x_\psi,l_\psi,w_\psi\}\cup vars(f)\cup vars(g)$, 其中, x_ψ为测试器布尔输出变量, $l_\psi\in[0,a]$是表示区间下界的整型变量, $w_\psi\in[0,b-a]$是表示区间宽度的整型变量;
- $\theta_\psi:=\top$;
- R_ψ由以下公式的合取得到:
  - $(x_\psi\wedge l_\psi>0\wedge w_\psi>0)\to(\chi(f)\wedge x'_\psi\wedge l'_\psi=l_\psi-1\wedge w'_\psi=w_\psi)$;
  - $(x_\psi\wedge l_\psi=0\wedge w_\psi>0)\to(\chi(g)\vee(\chi(f)\wedge x'_\psi\wedge l'_\psi=0\wedge w'_\psi=w_\psi-1))$;
  - $(x_\psi\wedge l_\psi=0\wedge w_\psi=0)\to\chi(g)$;
- $J_\psi:=\varnothing$.

2.3.2 $fU_{[a,a]}g$ 正时态测试器构造

设$\psi=fU_{[a,a]}g$ 并且 $a>0$, ψ的正时态测试器 $T_\psi=(V_\psi,\theta_\psi,R_\psi,J_\psi)||T_f||T_g$ 定义如下.

- $V_\psi:=\{x_\psi,l_\psi\}\cup vars(f)\cup vars(g)$, 其中, x_ψ为测试器布尔输出变量, 整型变量 $l_\psi\in[0,a]$表示区间下界;
- $\theta_\psi:=\top$;
- R_ψ由以下公式的合取得到:
  - $(x_\psi\wedge l_\psi>0)\to(\chi(f)\wedge x'_\psi\wedge l'_\psi=l_\psi-1)$;
  - $(x_\psi\wedge l_\psi=0)\to\chi(g)$.
- $J_\psi:=\varnothing$.

2.3.3 $fU_{[0,a]}g$ 正时态测试器构造

设$\psi=fU_{[0,a]}g$ 并且 $a>0$, ψ的正时态测试器 $T_\psi=(V_\psi,\theta_\psi,R_\psi,J_\psi)||T_f||T_g$ 定义如下.

- $V_\psi:=\{x_\psi,l_\psi\}\cup vars(f)\cup vars(g)$, 其中, x_ψ为测试器布尔输出变量, 整型变量 $l_\psi\in[0,a]$表示区间宽度;
- $\theta_\psi:=\top$;

- R_ψ由以下公式的合取得到:
  - $(x_\psi \wedge l_\psi > 0) \to (\chi(g) \vee (\chi(f) \wedge x'_\psi \wedge l'_\psi = l_\psi - 1))$;
  - $(x_\psi \wedge l_\psi = 0) \to \chi(g)$.
- $J_\psi := \varnothing$.

2.4 限界Release时态算子的正时态测试器构造

由于 $fR_{[a,b]}g = \neg(\neg fU_{[a,b]}\neg g)$, 通过扩展公式(2), 可推导出如下扩展公式(3):

$$fR_{[a,b]}g = \begin{cases} f \vee X(fR_{[a-1,b-1]}g), & \text{如果} 0 < a \leqslant b \\ g \wedge (f \vee X(fU_{[0,b-1]}g)), & \text{如果} 0 = a < b \\ g, & \text{如果} 0 = a = b \end{cases} \tag{3}$$

本节根据扩展公式(3)构造限界 Release 时态算子的正时态测试器.

2.4.1 $fR_{[a,b]}g$ 正时态测试器构造

设 $\psi = fR_{[a,b]}g$ 并且 $0<a<b$, ψ的正时态测试器 $T_\psi = (V_\psi, \theta_\psi, R_\psi, J_\psi) \| T_f \| T_g$ 定义如下.

- $V_\psi := \{x_\psi, l_\psi, w_\psi\} \cup vars(f) \cup vars(g)$, 其中, x_ψ为测试器布尔输出变量, $l_\psi \in [0,a]$是表示区间下界的整型变量, $w_\psi \in [0, b-a]$是表示区间宽度的整型变量;
- $\theta_\psi := \top$;
- R_ψ由以下公式的合取得到:
  - $(x_\psi \wedge l_\psi > 0 \wedge w_\psi > 0) \to (\chi(f) \vee (x'_\psi \wedge l'_\psi = l_\psi - 1 \wedge w'_\psi = w_\psi))$;
  - $(x_\psi \wedge l_\psi = 0 \wedge w_\psi > 0) \to (\chi(g) \wedge (\chi(f) \vee (x'_\psi \wedge l'_\psi = 0 \wedge w'_\psi = w_\psi - 1)))$;
  - $(x_\psi \wedge l_\psi = 0 \wedge w_\psi = 0) \to \chi(g)$;
- $J_\psi := \varnothing$.

2.4.2 $fR_{[a,a]}g$ 正时态测试器构造

设 $\psi = fR_{[a,a]}g$ 并且 $a>0$, ψ的正时态测试器 $T_\psi = (V_\psi, \theta_\psi, R_\psi, J_\psi) \| T_f \| T_g$ 定义如下.

- $V_\psi := \{x_\psi, l_\psi\} \cup vars(f) \cup vars(g)$, 其中, x_ψ为测试器布尔输出变量, 整型变量 $l_\psi \in [0,a]$表示区间下界;
- $\theta_\psi := \top$;
- R_ψ由以下公式的合取得到:
  - $(x_\psi \wedge l_\psi > 0) \to (\chi(f) \vee (x'_\psi \wedge l'_\psi = l_\psi - 1))$;
  - $(x_\psi \wedge l_\psi = 0) \to \chi(g)$.
- $J_\psi := \varnothing$.

2.4.3 $fR_{[0,a]}g$ 正时态测试器构造

设 $\psi = fR_{[0,a]}g$ 并且 $a>0$, ψ的正时态测试器 $T_\psi = (V_\psi, \theta_\psi, R_\psi, J_\psi) \| T_f \| T_g$ 定义如下.

- $V_\psi := \{x_\psi, l_\psi\} \cup vars(f) \cup vars(g)$, 其中, x_ψ为测试器布尔输出变量, 整型变量 $l_\psi \in [0,a]$表示区间宽度;
- $\theta_\psi := \top$;
- R_ψ由以下公式的合取得到:
  - $(x_\psi \wedge l_\psi > 0) \to (\chi(g) \wedge (\chi(f) \vee (x'_\psi \wedge l'_\psi = l_\psi - 1)))$;
  - $(x_\psi \wedge l_\psi = 0) \to \chi(g)$.
- $J_\psi := \varnothing$.

2.5 任意RTCTL*公式的正时态测试器组合构造

2.5.1 冗余子式的消除算法

对于有复杂结构的路径公式 ψ, 如果在公式语法树上采用自底向上逐步构造的方法构造该公式的正时态测试器 T_ψ和输出断言 $\chi(\psi)$, 则会对构造过程中遇到的每一主时态子式构造相应的子测试器. 我们定义 X_ψ为在 $DAG(\psi)$上构建 T_ψ时创建的新变量的集合. 若该公式包含多个相同的主时态子式, 则会导致最终构造的测试

器中包含多个完全相同且冗余的子测试器, 使得 X_ψ包含许多冗余的新建变量, 导致测试器状态空间迅速膨胀.

为消除冗余的子测试器, 我们设计了算法 1, 将ψ的语法树转化为有向无环图, 其中不包含任何冗余的终端节点和中间节点. 然后, 根据该图构造ψ的正时态测试器 T_ψ. 首先创建一个堆栈 stack, 用于存储ψ子式语法树的根节点. 然后设计一个函数 $cache(n_\psi)$, 将表示公式ψ的根节点 n_ψ存储到 stack 中, 同时确保 stack 中的节点所表示的子式互不相同, 其流程如下: 如果 stack 中存在节点 n, 其表示的子式与 n_ψ所表示的ψ子式相同, 则将所有指向 n_ψ的语法树有向边重定向到 n, 并删除 n_ψ及其射出边; 否则, 将 n_ψ加入 stack. 最后, 算法 1 形式描述如下.

算法 1. $DAG(n_\psi)$: G_ψ.

输入: n_ψ是公式ψ的语法树的根节点;

输出: G_ψ为将ψ语法树中所有冗余终端节点和中间节点删除后得到的有向无环图.

1. **if** n_ψ是一个断言 **then** $cache(n_\psi)$;
2. **else if** n_ψ是一元算子(路径量词 E 或 A, 时态算子 X 和否定词$\neg$) **then**
3. $DAG(left(n_\psi))$; //$left(n_\psi)$为 n_ψ唯一的后继节点,其表示的子式已不含冗余节点
4. $cache(n_\psi)$;
5. **else** //n_ψ是二元算子(时态算子 U, R, $U_{[a,b]}$或 $R_{[a,b]}$)
6. $DAG(left(n_\psi))$; //$left(n_\psi)$为 n_ψ的左后继节点, 其表示的子式已不含冗余节点
7. $DAG(right(n_\psi))$; //$right(n_\psi)$为 n_ψ的右后继节点, 其表示的子式已不含冗余节点
8. $cache(n_\psi)$;

给定一个 RTCTL*状态公式的否定范式ψ, 假设ψ语法树的根节点是 n_ψ, 则 $DAG(n_\psi)$返回ψ语法树对应的有向无环图 G_ψ, 其中已消除所有冗余终端节点和中间节点.

2.5.2 正时态测试器的构造示例

通过算法 1 $DAG(n_\psi)$获得 RTCTL*否定范式ψ语法树对应的有向无环图 G_ψ后, 假设 L 是 G_ψ的拓扑逆序列, 则通过从头到尾依次遍历 L 节点的方式可构造正时态测试器 T_ψ, 其具体实现算法如第 3 节的算法 2 所示. 本节通过如下示例, 直观解释正时态测试器的构造过程.

给定路径公式$\psi=(fR_{[a,b]}g)\wedge(fU_{[c,d]}(fR_{[a,b]}g))$, 其中, f 和 g 是两个断言, 并且 $a<b$ 和 $c<d$, 则正时态测试器 T_ψ的构造过程如下: 首先, 通过算法 1 $DAG(n_\psi)$将ψ语法树(图 1 左子图)转化为ψ的有向无环图(图 1 右子图), 该图的拓扑逆序列 $L=\{f,g,R_{[a,b]},U_{[c,d]},\wedge\}$(其中的元素为该有向无环图节点); 然后, 对 L 的第 1 个时态算子节点 $R_{[a,b]}$表示的子式 $fR_{[a,b]}g$ 构造其测试器 $T_1=(V_1,\theta_1,R_1,J_1)$和输出断言$\chi(fR_{[a,b]}g)=(x_1\wedge l_1=a\wedge w_1=b-a)$, 其中, $V_1:=\{x_1,l_1,w_1\}\cup vars(f)\cup vars(g)$, 其他组件请参见第 2.4.1 节. 进一步地, 将 L 的第 2 个时态算子节点 $U_{[c,d]}$的右孩子替换为$\chi(fR_{[a,b]}g)$, 为其表示的子式 $fU_{[c,d]}(fR_{[a,b]}g)$构造相应的输出断言$\chi(fU_{[c,d]}\chi(fR_{[a,b]}g))=(x_2\wedge l_2=c\wedge w_1=d-c)$和测试器 $T_2=(V_2,\theta_2,R_2,J_2)\|T_1$, 其中,

- $V_2:=\{x_2,l_2,w_2\}\cup\{x_1,l_1,w_1\}\cup vars(f)\cup vars(g)$, 其中, x_2 为测试器 T_2 布尔输出变量, $l_2\in[0,c]$是表示区间下界的整型变量, $w_2\in[0,d-c]$是表示区间宽度的整型变量;
- $\theta_2:=\top$;
- R_2 由以下公式的合取得到:
  - $(x_2\wedge l_2>0\wedge w_2>0)\rightarrow(f\wedge x_2'\wedge l_2'=l_2-1\wedge w_2'=w_\varphi)$;
  - $(x_2\wedge l_2=0\wedge w_2>0)\rightarrow((x_1\wedge l_1=a\wedge w_1=b-a)\vee(f\wedge x_2'\wedge l_2'=0\wedge w_2'=w_2-1))$;
  - $(x_2\wedge l_2=0\wedge w_2=0)\rightarrow(x_1\wedge l_1=a\wedge w_1=b-a)$;
- $J_2:=\varnothing$.

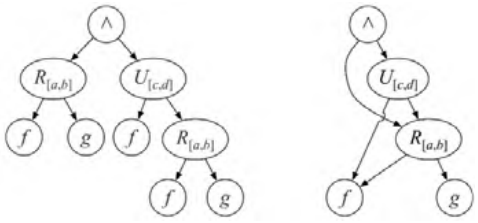


图 1 $\psi=(fR_{[a,b]}g)\wedge(fU_{[c,d]}(fR_{[a,b]}g))$的语法树(左子图)和 $DAG(n_\psi)$(右子图)

从图 1 可知: 基于算法 $DAG(n_\psi)$的测试器构造方法创建了新变量 $X_\psi=\{x_1,l_1,w_1,x_2,l_2,w_2\}$, 避免为 2 个时态算子 $R_{[a,b]}$中的一个构造相应的子测试器, 因此减少了创建 1 个布尔输出变量以及 2 个域分别为[0,a]和[0,$b-a$]的整型变量. 最终为公式ψ构造的正时态测试器是 T_2(其中包含 T_1), 并且输出断言:

$$\chi(\psi)=(x_1\wedge l_1=a\wedge w_1=b-a)\wedge(x_2\wedge l_2=c\wedge w_1=d-c).$$

3 基于 RTCTL*正时态测试器的符号化模型检测算法

如上节所述, 对于任意 RTCTL*公式ψ, 为了实现在有向无环图 $DAG(n_\psi)$中自底向上地组合构造ψ的输出断言$\chi(\psi)$和正时态测试器 T_ψ, 算法 2 按照 $DAG(n_\psi)$的拓扑逆序列, 依次构造每一节点所表示子式对应的输出断言和正时态测试器, 这些输出断言和正时态测试器将存储到相应的节点中, 因此该序列最后一个节点对应的输出断言和正时态测试器即为公式ψ的输出断言$\chi(\psi)$和正时态测试器 T_ψ.

算法 2 形式描述如下.

算法 2. ***sat***(***D***, ψ): ***L***.

输入: $D=(V,\theta,R,J)$是一个 JDS, ψ是一个 RTCTL*否定范式;

输出: L 为拓扑逆序列 $DAG(n_\psi)$, 其中所有节点都包含该节点对应子式的输出断言和正时态测试器.

1. 初始构造 $DAG(n_\psi)$的拓扑逆序列 L. 对任意元素$(n,o,t)\in L$, n 是一个ψ语法树节点. 设以 n 为根的子图表示子式ψ, 则 o 为输出断言$\chi(\psi)$的 BDD 表示, t 为正时态测试器 T_ψ. L 所有元素的 o 和 t 初始均为空值;
2. **for** i:=0 **to** $i<length(L)$ **do**
3. 令 n 为 L 的第 i 个元素中的节点, ψ为以 n 为根的子图所表示的子式;
4. **if** n 是断言或断言的否定 **then**
5. o:=ψ的 BDD 表示;
6. t:=(∅,⊤,⊤,∅); //“空”测试器
7. **else if** n 是连接词∧或∨ **then** //$\psi=f\wedge g$ 或 $f\vee g$
8. o_l:=$left(n).o$; o_{lr}:=$right(n).o$; //$left(n)$/$right(n)$表示 n 的左/右后继节点所对应的 L 元素
9. o:=$o_l no_r$ 的 BDD 表示;
10. t:=$left(n).t||right(n).t$;
11. **else if** n 是路径量词 E **then** //$\psi=Ef$, 其中, f 由以 $left(n)$为根的子图表示
12. t:=$left(n).t$;
13. o:=$fair(D||t)\wedge left(n).o$; //$o$ 即为满足 Ef 的状态集合$[\![Ef]\!]$
14. **else if** n 是路径量词 A **then** //$\psi=Af$,其中, f 由以 $left(n)$为根的子图表示
15. 令 $L_{\neg f}$为 $DAG(nnf(\neg f))$的拓扑逆序列;
16. $sat(D,L_{\neg f})$;
17. t:=$last(L').t$;
18. o:=$\neg(fair(D||t)\wedge last(L_{\neg f}).o)$; //$o$ 即为满足 Af 的状态集合$[\![Af]\!]$

19. **else if** n 是时态算子 X **then** //$\psi=Xf$
20. $o:=\chi(\psi)$的 BDD 表示;
21. $t:=(V_\psi,\theta_\psi,R_\psi,J_\psi)||left(n).t$; //$V_\psi$, θ_ψ, R_ψ, J_ψ的定义如第 2.2.1 节所示
22. **else** n 是二元时态算子 **then** //$\psi=fUg$, fRg, $fU_{[a,b]}g$, $fU_{[0,a]}g$, $fU_{[a,a]}g$, $fR_{[a,b]}g$, $fR_{[0,a]}g$ 或 $fR_{[a,a]}g$.
23. $o:=\chi(\psi)$的 BDD 表示;
24. $t:=(V_\psi,\theta_\psi,R_\psi,J_\psi)||left(n).t||right(n).t$;//$V_\psi$, θ_ψ, R_ψ, J_ψ的定义如第 2.2.2 节–第 2.4.3 节所示
25. 将 L 的第 i 个元素更新为(n,o,t);
26. **return** L;

算法 2 第 13 行、第 18 行中的 $fair(D||t)$表示 JDSD 与正时态测试器 t 的同步并行组合系统的公平状态集合(BDD 表示). 对于 JDS $D=(V,\theta,R,J)$, D 的公平状态集合 $fair(D)=\nu Z.\wedge_{j\in J}EX(E(\top U(Z\wedge j)))$, 其中, $\nu Z.\tau(Z)$表示谓词变换函数$\tau(Z)$的最大不动点; $EX(Z)=\exists V'(R(V,V')\wedge Z(V'))$, $E(Z_1UZ_2)=\mu Z.(Z_2\vee(Z_1\wedge EX(Z)))$, 其中, $\mu Z.\tau(Z)$表示谓词变换函数$\tau(Z)$的最小不动点.

算法 3 $check(D,\psi)$为基于正时态测试器的 RTCTL*符号化模型检测主算法, 用于判定公平离散系统 D 是否满足 RTCTL*公式 ψ.

算法 3. $\boldsymbol{check(D,\psi)}$.

输入: D 是一个 JDS, ψ是一个 RTCTL*公式;

输出: 验证结果.

1. **if** ψ不是状态公式 **then** $\bar{\psi}:=nnf(E\neg\psi)$; **else** $\bar{\psi}:=nnf(\neg\psi)$;
2. $L:=sat(D,\bar{\psi})$;
3. 令$(n,o,t):=last(L)$为 L 最后一个元素, 则以 n 为根的子图即表示公式 $\bar{\psi}$, o 是输出断言 $\chi(\bar{\psi})$ 的 BDD 表示, t 是正时态测试器 $T_{\bar{\psi}}$;
4. **if** $(D.\theta\wedge o\wedge fair(D||t))=\perp$ **then** 输出"$D\models\psi$"; **else** 输出"$D\not\models\psi$";

4 RTCTL*正时态测试器构造的正确性

对任意 RTCTL*否定范式 ψ, 算法 2 构造的 T_ψ正确性可在定义 1 的可靠性和完备性两方面进行证明.

定理 1(RTCTL*正时态测试器可靠性). 设ψ是一个 RTCTL*公式, 对于正时态测试器 T_ψ的任意公平路径ρ和任意位置 $i\geqslant 0$, 如果$(T_\psi,\rho(i))\models\chi(\psi)$, 则$(T_\psi,\rho,i)\models\psi$.

证明: 如第 2 节所示, 除 X 时态算子外, 其余时态算子的正时态测试器都是严格根据其对应扩展式的语义进行构造的. 这些扩展式的语义正确性可直观获知, 因此该定理的 T_ψ可靠性易于在公式ψ的语法树上通过归纳法证明, 其证明过程平凡而繁琐, 篇幅所限, 证明过程从略. □

下面我们对最复杂的限界主时态公式的测试器完备性进行证明.

推论 1($\boldsymbol{fU_{[a,b]}g}$ 正时态测试器完备性). 设$\psi=fU_{[a,b]}g$, 其中, f 和 g 是断言, 且 $0<a<b$, 则对于某一 JDS D 上的任意公平路径 r 和任意位置 $i\geqslant 0$, T_ψ中必有一条公平路径ρ, 其上所有状态与 r 上对应位置的状态有相同的 $vars(\psi)$变量解释, 使得$(T_\psi,\rho(i))\models\chi(\psi)$当且仅当$(D,r,i)\models\psi$.

证明: 我们分别就如下情况(1)和情况(2)进行证明.

(1) 当$(D,r,i)\models\psi$时, 可以根据以下步骤为测试器 T_ψ寻找满足$(T_\psi,\rho(i))\models\chi(\psi)$的路径$\rho$:

首先, 令ρ为 r 在 $vars(\psi)$上的投影, 即对于任意 $i\geqslant 0$, 有$\rho(i)=r(i)\cap vars(\psi)$. 显然, ρ与 r 在 $vars(\psi)$上有相同解释. 由预设条件$(D,r,i)\models\psi$可知: 存在 $i'\in[i+a,i+b]$, 使得$(D,r,i')\models g$ 且对任意 $j\in[i,i')$有$(D,r,j)\models f$. 下面根据该语义在ρ上附加 X_ψ的赋值, 使得$(T_\psi,\rho(i))\models\chi(\psi)$. 已知$\chi(f)=f$ 与$\chi(g)=g$.

已知$\chi(\psi)=x_\psi\wedge l_\psi=a\wedge w_\psi=b-a$. 先为$\rho(i)$附加赋值$\chi(\psi)$, 再根据迁移关系可附加如下赋值: 对所有 $j\in[i,i+a]$, 为所有$\rho(j)$附加赋值 $x_\psi\wedge l_\psi=a-j+i\wedge w_\psi=b-a$; 对于所有 $j\in[i+a,i')$, 为$\rho(j)$附加赋值 $x_\psi\wedge l_\psi=0\wedge w_\psi=b-a-j+i+a$. 至此, 我们找到 T_ψ的一条公平路径ρ, 其上所有状态与 r 上对应位置的状态有相同的 $vars(\psi)$变量解释, 并且有:

$(T_\psi,\rho(i))\models\chi(\psi)$.

(2) 当$(D,r,i)\not\models\psi$时, 意味着$(D,r,i)\models\neg(fU_{[a,b]}g)$, 有如下两种情况: 情况(2.1)和情况(2.2).

(2.1) 对任意 $i'\in[i+a,i+b]$, 有$(D,r,i')\models\neg g$. 对于这种情况, 我们构造路径ρ如下: 首先, 令ρ为 r 在 $vars(\psi)$上的投影, 即对于任意 $i\geqslant 0$, 有$\rho(i)=r(i)\cap vars(\psi)$. 显然, ρ与 r 在 $vars(\psi)$上有相同解释.

然后, 尝试为$\rho(i)$附加赋值$\chi(\psi)=x_\psi\wedge l_\psi=a\wedge w_\psi=b-a$. 根据迁移关系和$\chi(f)=f$ 与$\chi(g)=g$, 我们可构造如下理想情况: 对所有 $j\in[i,i+b]$, 为$\rho(j)$附加 $x_\psi\wedge f$. 根据前提条件有: 对任意 $i'\in[i+a,i+b]$, 有$\rho(i')$满足$\neg g$. 在此基础上, 根据迁移关系可得$\rho(i+b)$满足 $x_\psi\wedge l_\psi=0\wedge w_\psi=0$. 由迁移关系$(x_\psi\wedge l_\psi=0\wedge w_\psi=0)\rightarrow\chi(g)$可知, $\rho(i+b)$必须满足$\chi(g)$(也就是 g). 然而根据前提条件可知, $\rho(i+b)$满足$\neg g$. 矛盾. 因此, 尝试为$\rho(i)$附加赋值$\chi(\psi)$是不可行的, 也就是说, $(T_\psi,\rho(i))\models\neg\chi(\psi)$.

(2.2) 对于满足$(D,r,i')\models g$ 的任意 $i'\in[i+a,i+b]$, 均存在$j\in[i,i')$使得$(D,r,j)\models\neg f$. 对于这种情况, 我们构造路径ρ如下: 首先, 令ρ为 r 在 $vars(\psi)$上的投影, 即对于任意 $i\geqslant 0$, 有$\rho(i)=r(i)\cap vars(\psi)$. 显然, ρ与 r 在 $vars(\psi)$上有相同解释. 然后, 假设存在 $i'\in[i+a,i+b]$使得$(D,r,i')\models g$, 进一步分为如下两种情况:

a) 存在 $j\in[i,i+a)$, 使得$(D,r,j)\models\neg f$; 且对任意 $k\in[i,j)$, 有$(D,r,k)\models f$. 尝试为$\rho(i)$附加赋值$\chi(\psi)=x_\psi\wedge l_\psi=a\wedge w_\psi=b-a$. 根据迁移关系可知, $\rho(j)$满足 $x_\psi\wedge l_\psi>0\wedge w_\psi>0$; 且由前提可知, $\rho(j)$不满足 f. 这不满足迁移关系 $(x_\psi\wedge l_\psi>0\wedge w_\psi>0)\rightarrow(\chi(f)\wedge x'_\psi\wedge l'_\psi=l_\psi-1\wedge w'_\psi=w_\psi)$, 意味着从$\rho(i)$不能到达$\rho(j)$. 因此, 为$\rho(i)$附加赋值$\chi(\psi)$是不可行的, 也就是说, $(T_\psi,\rho(i))\models\neg\chi(\psi)$;

b) 存在 $j\in[i+a,i')$, 使得$(D,r,j)\models\neg f$; 且对任意 $k\in[i+a,j)$, 有$(D,r,k)\models f$; 且对任意 $k'\in[i,j]$, 有$(D,r,k')\models\neg g$. 尝试为$\rho(i)$附加赋值$\chi(\psi)=x_\psi\wedge l_\psi=a\wedge w_\psi=b-a$. 根据迁移关系可知: $\rho(j)$满足 $x_\psi\wedge l_\psi=0\wedge w_\psi>0$, 且$\rho(j)$不满足 f和 g. 这不满足迁移关系 $(x_\psi\wedge l_\psi=0\wedge w_\psi>0)\rightarrow(\chi(g)\vee(\chi(f)\wedge x'_\psi\wedge l'_\psi=0\wedge w'_\psi=w_\psi-1))$, 意味着从$\rho(i)$不能到达$\rho(j)$. 因此, 为$\rho(i)$附加赋值$\chi(\psi)$是不可行的, 也就是说, $(T_\psi,\rho(i))\models\neg\chi(\psi)$.

证毕. □

推论 2($fR_{[a,b]}g$ 正时态测试器完备性). 设$\psi=fR_{[a,b]}g$, 其中, f 和 g 是断言, 且 $0<a<b$, 则对于某一 JDS D 上的任意公平路径 r 和任意位置 $i\geqslant 0$, T_ψ中必有一条公平路径ρ, 其上所有状态与 r 上对应位置的状态有相同的 $vars(\psi)$变量解释, 使得$(T_\psi,\rho(i))\models\chi(\psi)$当且仅当$(D,r,i)\models\psi$.

证明: 令ρ为 r 在 $vars(\psi)$上的投影, 即对于任意 $i\geqslant 0$, 有$\rho(i)=r(i)\cap vars(\psi)$. 显然, ρ与 r 在 $vars(\psi)$上有相同解释. 分如下情况(1)和情况(2)进行证明.

(1) 当$(D,r,i)\models\psi$时, 在ρ上附加 X_ψ赋值作为测试器 T_ψ的一条路径, 使得$(T_\psi,\rho(i))\models\chi(\psi)$. 分如下两种情况: 情况(1.1)和情况(1.2).

(1.1) 对所有 $i'\in[i+a,i+b]$, 有$(D,r,i')\models g$, 即$(T_\psi,\rho(i'))\models g$.

首先, 在$\rho(i)$附加赋值$\chi(\psi)$, 其中, $\chi(\psi)=x_\psi\wedge l_\psi=a\wedge w_\psi=b-a$. 极端情况下, 对所有 $j\in[i,i+a)$, 有$(T_\psi,\rho(j))\models\neg f$. 则根据 T_ψ迁移关系(第 1 行)和$\chi(f)=f$, $\chi(g)=g$, 有$\rho(i+a)$被附加赋值$(x_\psi\wedge l_\psi=0\wedge w_\psi=b-a)$. 即使对所有 $j\in[i+a,i+b)$有$(T_\psi,\rho(j))\models\neg f$, 根据(1.1)前提条件和迁移关系可知: 对所有 $j\in[i+a,i+b]$有$\rho(j)$被附加赋值$(x_\psi\wedge l_\psi=0\wedge w_\psi=b-j+i)$. 因此, 在$\rho(i)$附加赋值$\chi(\psi)$是可行的, 即$(T_\psi,\rho(i))\models\chi(\psi)$.

(1.2) 存在 $i'\in[i+a,i+b]$, 使得$(D,r,i')\models\neg g$; 并且存在 $j\in[i,i')$, 使得$(D,r,j)\models f$.

首先, 在$\rho(i)$附加赋值$\chi(\psi)$, 其中, $\chi(\psi)=x_\psi\wedge l_\psi=a\wedge w_\psi=b-a$. 根据 T_ψ迁移关系(第 1 行)有: 对所有 $k\in[i,j]$, 有$\rho(k)$被附加赋值 $x_\psi\wedge l_\psi=i+a-k$, 其中, $l_\psi>0$. 因此, 在$\rho(i)$附加赋值$\chi(\psi)$是可行的, 即$(T_\psi,\rho(i))\models\chi(\psi)$.

(2) 当$(D,r,i)\models\neg\psi$时, 在ρ上尝试附加 X_ψ赋值作为测试器 T_ψ的一条路径, 使得$(T_\psi,\rho(i))\models\chi(\psi)$.

由前提可知: ρ中存在 $i'\in[i+a,i+b]$, 使得$(T_\psi,\rho(i'))\models\neg g$; 并且对所有 $j\in[i,i')$, 有$(T_\psi,\rho(j))\models\neg f$. 我们尝试在$\rho(i)$附加赋值$\chi(\psi)=x_\psi\wedge l_\psi=a\wedge w_\psi=b-a$. 根据$\chi(f)=f$和$\chi(g)=g$ 以及测试器 T_ψ迁移关系(第 1 行), 必然对所有 $j\in[i+1,i+a]$, 在$\rho(j)$上附加 $x_\psi\wedge l_\psi=a-j+i\wedge w_\psi=b-a$. 因此, $\rho(i+a)$附加了 $l_\psi=0$. 由前提可知: 对所有$j\in[i+a,i')$, 有$(T_\psi,\rho(j))\models\neg f$. 根据 T_ψ迁移关系(第 2 行), 必然有: 对所有 $k\in[i+a,i')$, 在$\rho(k)$上附加 $g\wedge x_\psi\wedge l_\psi=0\wedge w_\psi=i+b-k$; 以及在$\rho(i')$上

附加 $x_\psi \wedge l_\psi=0 \wedge w_\psi=i+b-i'$(其中, $w_\psi \geqslant 0$). 再根据 T_ψ迁移关系(第 2,3 行), 必然在$\rho(i')$上附加 g, 这与前提相矛盾. 因此, 在路径ρ^j 的首个状态$\rho(i)$上附加$\chi(\psi)=x_\psi \wedge l_\psi=a \wedge w_\psi=b-a$ 是不可行的, 即$(T_\psi,\rho(i)) \not\models \chi(\psi)$. □

定理 2(RTCTL*正时态测试器完备性). 设ψ是一个 RTCTL*公式, 则对于某一 JDS D 上的任意公平路径 r 和任意位置 $i \geqslant 0$, T_ψ中必有一条公平路径ρ, 其上所有状态与 r 上对应位置的状态有相同的 $vars(\psi)$变量解释, 使得$(T_\psi,\rho(i)) \models \chi(\psi)$当且仅当$(D,r,i) \models \psi$.

证明: 分如下情况(1)–情况(5)进行证明.

(1) 当$\psi=fU_{[a,a]}g$ 或 $fU_{[0,a]}g$ 时, 其测试器是 $fU_{[a,b]}g$ 的优化版本, 其中: $fU_{[a,a]}g$ 中的区间宽度为 0, 不需创建宽度变量 w_ψ; $fU_{[0,a]}g$ 中的区间下界为 0, 不需创建下界变量 l_ψ. 因此, 推论 1 中 $fU_{[a,b]}g$ 测试器的完备性证明经适当简化即可作为 $fU_{[a,a]}g$ 和 $fU_{[0,a]}g$ 正时态测试器的完备性证明. 同理, 推论 2 中 $fUR_{[a,b]}g$ 测试器的完备性证明经适当简化即可作为 $fR_{[a,a]}g$ 和 $fR_{[0,a]}g$ 正时态测试器的完备性证明;

(2) 当φ是断言或断言的否定时, ψ中不含任何时态算子, 因此$\chi(\psi)=\psi$并且 $T_\psi=(\varnothing,\top,\top,\varnothing)$, 可将 D 的路径 r 在 $vars(\psi)$上的投影作为 T_ψ的一条路径ρ. 显然, $(T_\psi,\rho(i)) \models \chi(\psi)$当且仅当$(D,r,i) \models \psi$;

(3) 当$\psi=f \wedge g$ 或 $f \vee g$ 时, 有$\chi(\psi)=\chi(f) \wedge \chi(g)$或$\chi(\psi)=\chi(f) \vee \chi(g)$以及 $T_\psi=T_f \| T_g$. 因此, 由归纳假设的 T_f和 T_g 完备性易证 T_ψ的完备性;

(4) 当$\psi=Ef$ 时, 有 $T_\psi=T_f$和$\chi(\psi)=[\![Ef]\!]$. $[\![Ef]\!]$是由算法 2 第 13 行计算得到的 D 中满足 Ef 的状态集合, 由此易证 T_ψ的完备性;

(5) 当$\psi=Af$ 时, 有 $T_\psi=T_{nnf(\neg f)}$和$\chi(\psi)=[\![Af]\!]$. $[\![Af]\!]$是由算法 2 第 18 行计算得到的 D 中满足 Af 的状态集合, 由此易证 T_ψ的完备性.

证毕. □

5 基于 RTCTL*正时态测试器的模型检测算法时间复杂度

设ψ是一个 RTCTL*否定范式, 我们定义其长度(记为$|\psi|$)为以最简洁形式写下的字串长度. $|\psi|$定义如下:

$$|\psi|=\begin{cases}1, & \psi \text{是一个断言}\\ |f|+1, & \psi=\neg f, Ef, Af \text{或} Xf\\ |f|+|g|+1, & \psi=f \wedge g, f \vee g, fUg \text{或} fRg\\ |f|+|g|+1+\lceil \log_2 a \rceil+\lceil \log_2 b \rceil, & \psi=fU_{[a,b]}g \text{或} fR_{[a,b]}g \text{并且} 0<a<b\\ |f|+|g|+2+\lceil \log_2 a \rceil, & \psi=fU_{[a,a]}g, fU_{[0,a]}g, fR_{[a,a]}g, fR_{[0,a]}g \text{并且} a>0\\ |g|, & \psi=fU_{[0,0]}g \text{或} fR_{[0,0]}g\end{cases}$$

其中, 符号 $\log_2 a$ 表示 $\log_2 a$ 的上整数. 由$|\psi|$的定义可知: 断言的长度为 1, 非 0 正整数变量的长度为其二进制编码变量的个数, 布尔连接词、路径量词和时态算子的长度为 1. 对于$|\psi|$定义的第 5 行, 区间$[a,a]$和$[0,a]$可分别简写为"$=a$"和"$\leqslant a$", 其中的符号"="和"$\leqslant$"的长度分别为 1. 对于$|\psi|$定义的第 6 行, 公式ψ可化简为其子式 g,因此在此情况下, $|\psi|=|g|$. 例如, 假设 f 和 g 为断言, 则:

$$|fU_{[300,1000]}g|=|f|+|g|+|U|+|\log_2 300|+|\log_2 1000|=1+1+1+9+10=22.$$

正时态测试器的构造复杂度分析如下.

定理 3(RTCTL*正时态测试器构造复杂度). 对任意 RTCTL*公式ψ, 存在有 $O(|\psi|)$个布尔变量的正时态测试器 T_ψ.

证明: 最坏情况下, 公式ψ语法树是一棵满二叉树, 其中, 终端结点是布尔变量, 非终端结点是 $U_{[a,b]}$或 $R_{[a,b]}(0<a<b)$, 并且 $DAG(n_\psi)$不能删除ψ语法树的任何结点, 即ψ中任意子式互不相同.

假设$\mathcal{B}(T_\psi)$为 T_ψ中的变量二进制编码集合, 根据正时态测试器的构造过程, 需要为每一 $U_{[a,b]}$或 $R_{[a,b]}$算子创建 1 个布尔输出变量 x_ψ、2 个整型变量 $l_\psi \in [0,a]$和 $w_\psi \in [0,b-a]$, 因此有:

$$\mathcal{B}(T_\psi)=(2^{h-1}-1)\times(1+\log_2(a+1)+\log_2(b-a+1))+2^{h-1}.$$

其中, h 是ψ语法树高度. 当 $a=b/2$ 时, $\mathcal{B}(T_\psi)$成为极大值:

$$(2^{h-1}-1)\times(1+2\log_2(b/2+1))+2^{h-1}=(2^{h-1}-1)\times(2\log_2(b+2)-1)+2^{h-1}.$$

注意: 为便于计算, 在 a 足够大时, 可删除上整数符号. 我们有公式长度:

$$|\psi|=(2^{h-1}-1)\times(1+\log_2 a+\log_2 b)+2^{h-1}=2^{h-1}\times(2+\log_2 a+\log_2 b)-(1+\log_2 a+\log_2 b).$$

当 $a=b/2$ 并且足够大时, $|\psi|=2^{h-1}\times(1+2\log_2 b)-2\log_2 b$. 先求极限 $\lim\limits_{h\to\infty,a=b/2}\dfrac{\mathcal{B}(T_\psi)}{|\psi|}=\dfrac{2\log_2(b+2)}{1+2\log_2 b}$, 然后求极限 $\lim\limits_{b\to\infty}\dfrac{2\log_2(b+2)}{1+2\log_2 b}=1$, 因此在最坏情况下, T_ψ有$|\psi|$个布尔变量, 即 T_ψ有 $O(|\psi|)$个布尔变量. □

我们知道, JDS D 的一个可能状态是其中变量的二进制编码的一个赋值, 因此, D 的可能状态空间 $|D|=O(2^{\mathcal{B}(D)})$. 设$\psi$是一个 RTCTL*公式, 并且$\psi_1=nnf(\neg\psi)$, 算法 3 将 $D\models\psi$模型检测问题转化为在 $D\|T_{\psi_1}$ 的可能状态空间中搜索满足输出断言$\chi(\psi_1)$的状态集合的问题.

我们有 $|D\|T_{\psi_1}|=|D|\times|T_{\psi_1}|=|D|\times O(2^{\mathcal{B}(T_{\psi_1})})=|D|\times O(2^{|\psi_1|})$, 得到如下定理 4.

定理 4. 给定 JDS D 和 RTCTL*公式ψ, 基于 RTCTL*正时态测试器的模型检测算法时间复杂度为 $|D|\times O(2^{|\psi|})$.

6 工具实现与比较实验

6.1 模型检测工具MCTK 2.0.0

我们基于 Pnueli 等人的 JTLV 验证算法平台(1.4.0 版)开发了一个高效的 RTCTL*符号化模型检测工具 MCTK 2.0.0(以下简称 MCTK, 开源下载地址: https://gitlab.com/hovertiger/mctk3). 我们不仅在 MCTK 中利用 JavaBDD 实现了本文提出的 RTCTL*符号化模型检测算法, 而且开发了 RTCTL*树状反例的图形化生成工具. 反例生成不是本文的主题, 相关工作将另文介绍. MCTK 兼容支持 NuSMV 和 nuXmv 所采用的的 SMV 输入语言, 支持模块化、分层次定义 JDS. 图 2 是 MCTK 的主界面, 其中, 右下角的窗口是一个 RTCTL*公式的反例生成界面.

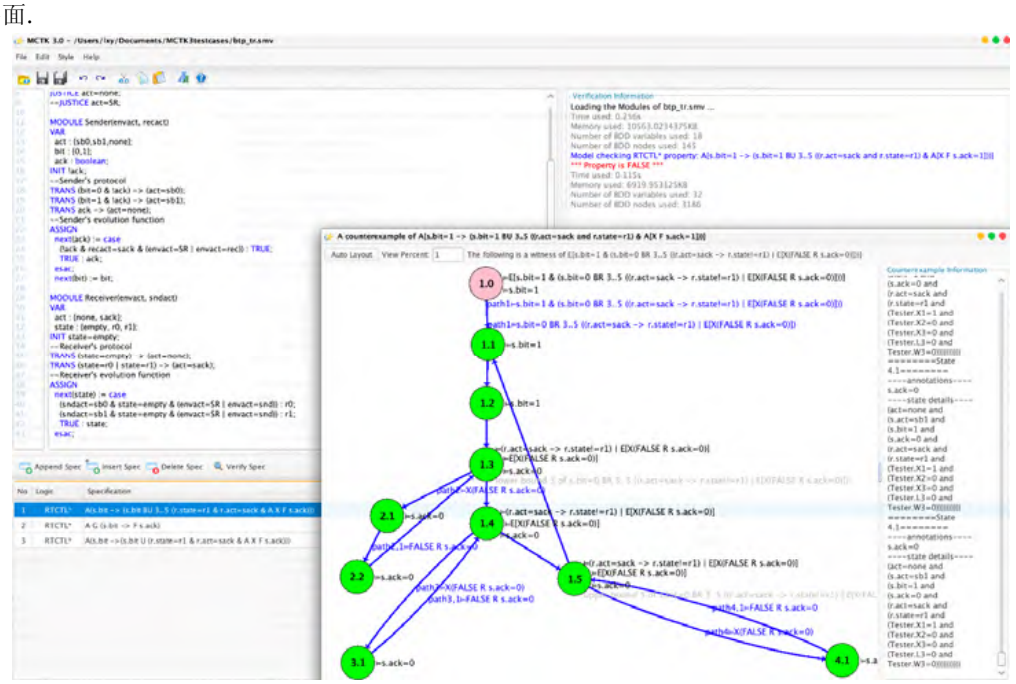


图 2 MCTK 2.0.0 主界面

6.1.1 nuXmv 与 MCTK 的 RTLTL 验证性能比较

NuSMV 及其升级版本 nuXmv 是两个著名的符号化模型检测工具, 共同支持基于 BDD 的 RTLTL 符号化模型检测以及基于 SAT 的 RTLTL 限界模型检测, 两者在学术界和工业界已得到广泛研究和应用, 有重要的研究和应用价值. 由于 NuSMV 和 nuXmv 均不支持 CTL*和 RTCTL*, 并且 nuXmv 中基于 BDD 的 RTLTL 模型检测算法完全继承自 NuSMV, 因此下面的实验将比较 nuXmv 与 MCTK 的 RTLTL 验证性能. 我们分别在以下的实验 1 和实验 2 中针对比特传输协议模型和 NuSMV 的基准测试案例 syncarb5 进行 nuXmv 与 MCTK 的性能比较实验.

6.1.2 实验 1: nuXmv 与 MCTK 对比特传输协议的验证性能比较

比特传输协议(BTP)的 JDS 模型由图 3 的 SMV 输入语言描述, 其中有 Sender *s* 和 Receiver *r* 两个进程, 进程 *s* 随机选择 1 个比特(0 或 1)后, 持续地向进程 *r* 发送这个比特, 直至收到来自进程 *r* 的确认消息(*ack*=TRUE)为止. 而进程 *r* 收到这个比特后, 将持续发送确认消息(*r.ack=sack*). 然而, 两进程之间的双工通信是不可靠的(从一方到另一方的比特传输可能会失败). 我们通过 JUSTICE 关键词定义弱公平约束条件, 动态调整双工通道的失效模式, 比如图 3 中的两个弱公平约束条件 *act=snd* 和 *act=none* 确保了从进程 *s* 到进程 *r* 的通信是有效的, 但是从进程 *r* 到进程 *s* 的通信是可能失效的.

```
1  MODULE main
2  VAR act: {snd,rec,SR,none};
3     s: Sender(act,r.act);
4     r: Receiver(act,s.act);
5  JUSTICE act=snd;
6  JUSTICE act=none;
7
8  MODULE Sender(envact,recact)
9  VAR act: {sb0,sb1,none};
10    bit: {0,1};
11    ack: boolean;
12 INIT !ack;
13 --Sender's protocol
14 TRANS (bit=0 & !ack)→(act=sb0);
15 TRANS (bit=1 & !ack)→(act=sb1);
16 TRANS ack→(act=none);
17 --Sender's evolution function
18 ASSIGN
19 next(ack):=case
20    (!ack& recact=sack &
21       (envact=SR|envact=rec)): TRUE;
22    TRUE: ack;
23 esac;
24 next(bit):=bit;
25
26 MODULE Receiver(envact,sndact)
27 VAR act: {none,sack};
28    state: {empty,r0,r1};
29 INIT state=empty;
30 --Receiver's protocol
31 TRANS (state=empty)→(act=none);
32 TRANS (state=r0|state=r1)→(act=sack);
33 --Receiver's evolution function
34 ASSIGN
35 next(state):=case
36    (sndact=sb0 & state=empty &
37       (envact=SR|envact=snd)): r0;
38    (sndact=sb1 & state=empty &
39       (envact=SR|envact=snd)): r1;
40    TRUE: state;
41 esac;
42
```

图 3 比特传输协议的 SMV 输入语言

图 2 中的图形化树状反例是在检测 RTCTL*公式 $A(s_1\rightarrow(s_1U_{[3,5]}(r_1\wedge sa\wedge AXF\ ra)))$为假后生成的, 其中, 符号 s_1 表示进程 *s* 发送比特 1, 符号 r_1 表示进程 *r* 接受到比特 1, 符号 *sa* 表示进程 *r* 发送确认消息, 符号 *ra* 表示进程 *s* 收到进程 *r* 的确认消息. 由于 RTLTL 不能表达上述 RTCTL*公式, 为了与 nuXmv 进行比较实验, 我们将上述公式的路径量词 *A* 删除, 变成如下形式的 RTLTL 公式:

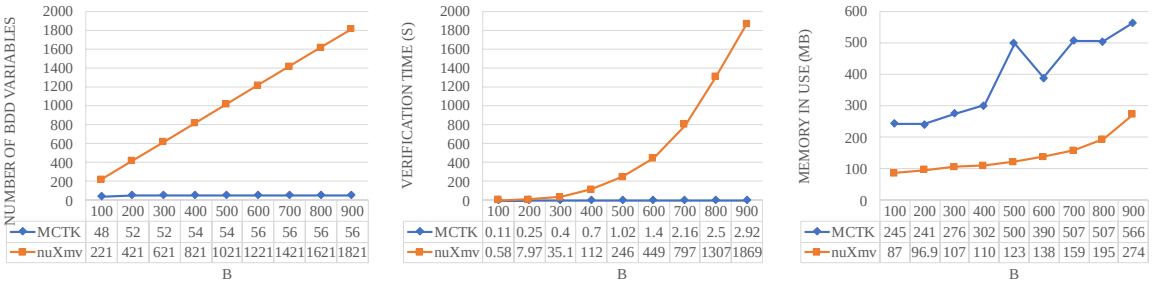
$$\psi_1:=s_1\rightarrow(s_1U_{[a,b]}(r_1\wedge sa\wedge XF\ ra)) \tag{4}$$

我们在一台配置了 Intel i7 2.90GHz CPU、16GB 内存和 Ubuntu Desktop 20.04 LTS 操作系统的电脑上, 分别用 MCTK 和 nuXmv 检测了若干形如 ψ_1 的 RTLTL 公式, 其中, $a=b-50$, 并且 *b* 以 100 为单位从 100 步进递增到 900, 检测结果均为假. 检测数据如图 4 所示, 其中, 图 4(a)~图 4(c)的横坐标是 ψ_1 的区间上界 *b*, 横坐标下的表格是图中各点的纵坐标具体数值.

(1) nuXmv 和 MCTK 的测试器构造复杂度与时间消耗比较

首先, nuXmv 需要创建 19 个 BDD 变量建模 BTP 协议. 对于公式(2) ψ_1, nuXmv 需要为时态算子 *X*, *F* 和 $U_{[a,b]}$创建输出变量. 与 MCTK 类似, nuXmv 为无区间的 LTL 时态算子 *X* 和 *F* 分别创建 2 个 BDD 变量(其中包含表示下一状态的变量副本). 从图 4(a)的数据可以推断出, nuXmv 为限界时态算子 $U_{[a,b]}$创建了 $2(b-1)$个 BDD

变量(其中一半是变量副本). 假设 nuXmv 为公式ψ构建的测试器为T_ψ^N, 则在不计入变量副本的最坏情况下, 有$\mathcal{B}(T_\psi^N)=(2^{h-1}-1)\times(b-1)+2^{h-1}$, 其中, h是ψ语法树高度. 因此, nuXmv 验证该例所创建的 BDD 布尔变量数$\mathcal{B}(T_\psi^N)$与区间上界 b 呈线性关系. 这与图 4(a)关于 nuXmv 的数据相一致.



(a) BDD 变量数 (b) 验证时间(s) (c) 内存用量(MB)

图 4

然后, 假设 MCTK 为公式ψ构建的测试器为 T_ψ. 由定理 4 的证明和 a=b−50 可知:

$$\mathcal{B}(T_\psi)=(2^{h-1}-1)\times(1+\log_2(b-49)+\log_2 51)+2^{h-1}.$$

因此, MCTK 验证该例所创建的 BDD 变量数与区间上界 b 呈对数关系. 这与图 4(a)关于 MCTK 的数据相一致. 注意, MCTK 需要为 BTP 协议模型创建 18 个 BDD 变量.

上述分析表明: MCTK 创建的 BDD 变量数指数级小于 nuXmv, 而测试器的状态空间规模与测试器布尔变量数也呈指数关系. 因此, MCTK 模型检测 RTLTL 的时间复杂度双指数级小于 nuXmv. 图 4(b)的具体数据也符合这一结论. 事实上, 我们也用 MCTK 和 nuXmv 验证了 b=1000 的公式ψ_1, MCTK 只用了 4.342 s 即输出验证结果和反例, 而 nuXmv 在 4h 后仍未得出验证结果. 由此可见: nuXmv 的 RTLTL 验证性能完全不能与 MCTK 相比, 不能满足实用化的需求.

(2) nuXmv 和 MCTK 的内存消耗比较

图 4(c)说明 MCTK 的内存消耗通常比 nuXmv 更大, 但是增长率相近. 我们认为: 这并不是算法策略本身造成的, 而是因为 MCTK 所依托的 JTLV 平台采用的 BDD 软件包是早年(2010 年)发布的 JavaBDD 2.0, 并且 Java 的内存垃圾管理技术通常会推迟内存释放操作, 这会导致 MCTK 内存消耗处于较高水平. 而 nuXmv 及其采用的 BDD 软件包 CUDD 是采用 C 或 C++开发的, 使得开发者可及时手工释放无用内存. 因此, MCTK 中基于 Java 和 JavaBDD 实现的符号化算法在内存性能上要低于 nuXmv 中基于 C 和 C++实现的算法.

总的来说, 虽然基于 Java 开发的 MCTK 在内存消耗上要多于基于 C 和 C++开发的 nuXmv, 但是 MCTK 的时间复杂度双指数级小于 nuXmv. 因此, MCTK 高效的 RTLTL 模型检测算法具备了工业化应用的潜能. 下面进一步针对 NuSMV 基准测试案例 syncarb5 进行比较实验.

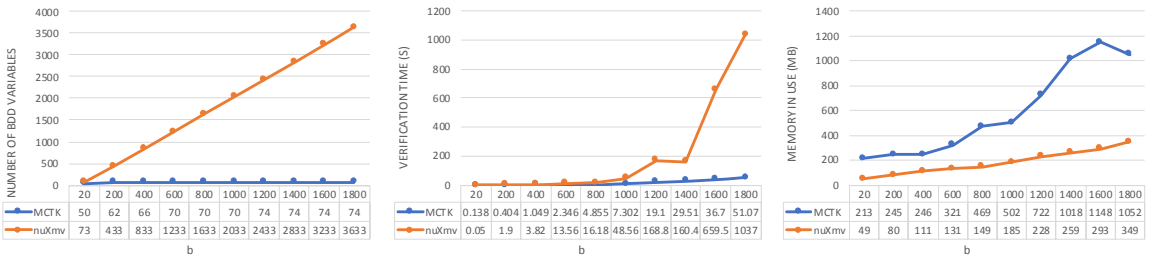
6.1.3 实验 2: nuXmv 与 MCTK 对基准测试案例 syncarb5 的 RTLTL 验证性能比较

NuSMV 软件包中的基准测试案例 syncarb5 是一个同步总线控制器电路模型, 其中包含 5 个控制器单元 e1~e5, 具体模型见文献[20]. 本文将对如下形式的 RTLTL 公式进行验证:

$$\psi_2:=G(\text{e5.Request}\rightarrow F_{[a,b]}(\neg\text{e5.Request}\vee\text{e5.ack-out})) \quad (5)$$

ψ_2的含义是, 任何时候, 只要控制器单元 e5 收到请求信号(e5.Request=⊤), 则在未来 a 至 b 个周期之间, 必然存在一个周期, 满足: 如果 e5 在此周期收到请求信号, 则 e5 将同时发出应答信号(e5.ack-out=⊤).

我们在一台配置了 Intel i7 3.40 GHz CPU、16 GB 内存和 Ubuntu Desktop 20.04 LTS 操作系统的电脑上, 分别用 nuXmv 和 MCTK 检测了若干形如ψ_2的 RTLTL 公式, 其中, b=2a. 初始情况下, b=20, 并且 b 以 200 为单位从 200 步进递增到 1 800, 检测结果均为真. 检测数据如图 5 所示, 其中, 图 5(a)–图 5(c)的横坐标是ψ_2的区间上界 b, 横坐标下的表格是图中各点的纵坐标具体数值. 从图 5 可以得出类似实验 1 的结论, 即: MCTK 的时间复杂度双指数级小于 nuXmv, 但是 MCTK 内存消耗高于 nuXmv.



(a) BDD 变量数 (b) 验证时间(s) (c) 内存用量(MB)

图 5

7 结论与未来工作

本文为了显著提升 RTCTL*模型检测的性能，提出一种简洁高效的 RTCTL*正时态测试器构造方法以及相关符号化模型检测算法. 基于 JavaBDD 软件包成功开发了该算法的模型检测工具 MCTK 2.0.0. 证明了所提出的 RTCTL*正时态测试器构造方法是完备的. 也证明了该算法时间复杂度与被验证系统呈线性关系，与公式长度呈指数关系. 与著名的符号化模型检测工具 nuXmv 相比，虽然基于 Java 开发的 MCTK 2.0.0 在内存消耗上要多于基于 C 和 C++开发的 nuXmv，但是实验结果和复杂度分析均表明, MCTK 2.0.0 的时间复杂度双指数级小于 nuXmv. 而且,RTLTL 也可视为 PSL 的基础时态逻辑之一，因此本文的 RTLTL 高效模型检测算法可作为 PSL 模型检测的基础算法. 综上，我们强烈建议包括 nuXmv 在内的符号化模型检测工具集成本文算法.

本文算法的另一优势是易于集成到其他工具，因为对于 RTLTL 这类不包含路径量词的纯线性时态逻辑，其时态测试器完全可以直接在工具输入语言中描述. 只要一个现有的模型检测工具(如 NuSMV 和 nuXmv 等)支持模块化描述子 JDS，并支持多个 JDS 之间的同步并行组合，则可根据正时态测试器的构造方法开发相应的语言翻译程序，将一个 RTLTL 公式 ψ的正时态测试器 T_ψ翻译为该工具输入语言代码并加入被验证系统模型中，同时将被验证公式 ψ替换为相应的输出断言 $\chi(\psi)$，即可在现有工具中实现 RTLTL 的高效验证，还能利用工具现成的反例生成算法直接生成 RTLTL 反例，反例中附带的测试器新建变量也将有助于用户深入理解生成的反例. 未来我们将基于上述思路实现基于 nuXmv 的 RTLTL 模型检测，从而有望充分利用 nuXmv 内部的所有优化算法，进一步提高验证效率.

时态测试器的另一重要应用是演绎验证(deductive verification)[21]. 基于所提出的正时态测试器的可组合性，未来我们可以构建一个 RTCTL*的演绎证明系统，使得在一个 JDS D 上的 RTCTL*公式 ψ证明可规约为在同步并行组合系统 $D||T_\psi$上的输出断言 $\chi(\psi)$的证明. 这样一个 RTCTL*演绎证明系统，将能够用于验证无穷状态实时系统的实时时态性质.

References:

[1] Clarke EM, Grumberg O, Kroening D, *et al*. Model Checking. 2nd ed., Cambridge: MIT Press, 2018.

[2] Clarke EM, Henzinger TA, Veith H, *et al*. Handbook of Model Checking. Cham: Springer, 2018. [doi: 10.1007/978-3-319-10575-8]

[3] IEEE Computer Society. IEEE Standard for Property Specification Language (PSL). IEEE Std 1850-2010 (Revision of IEEE Std 1850-2005), 2010. 1–182. [doi: 10.1109/IEEESTD.2010.5446004]

[4] IEEE Computer Society and the IEEE Standards Association Corporate Advisory Group. IEEE Standard for SystemVerilog—Unified Hardware Design, Specification, and Verification Language. IEEE Std 1800-2017 (Revision of IEEE Std 1800-2012), 2018. 1–1315. [doi: 10.1109/IEEESTD.2018.8299595]

[5] Pnueli A, Zaks A. On the merits of temporal testers. In: Grumberg O, Veith H, eds. Proc. of the 25 Years of Model Checking. Berlin, Heidelberg: Springer, 2008. 172–195. [doi: 10.1007/978-3-540-69850-0_11]

[6] Luo XY, Wu LJ, Chen QL, *et al*. Symbolic model checking for discrete real-time systems. Science China Information Science, 2018, 61(5): 052106:1–052106:23. [doi: 10.1007/s11432-017-9152-x]

[7] Kesten Y, Pnueli A, Raviv L. Algorithmic verification of linear temporal logic specifications. In: Larsen KG, Skyum S, Winskel G, eds. Proc. of the 25th Int'l Colloquium on Automata, Languages and Programming. Berlin, Heidelberg: Springer, 1998. 1–16. [doi: 10.5555/646252.756784]

[8] Bryant RE.Graph-Based algorithms for boolean function manipulation. IEEE Trans. on Computers, 1986, 35(8): 677–691. [doi: 10.1109/TC.1986.1676819]

[9] Maler O, Nickovic D, Pnueli A. From MITL to timed automata. In: Asarin E, BouyerP, eds. Proc. of the FORMATS 2006. Berlin, Heidelberg: Springer, 2006. 274–289. [doi: 10.1007/11867340_20]

[10] Cimatti A, Clarke E, Giunchiglia E, *et al*. Nusmv 2: An opensource tool for symbolic model checking. In: Brinksma E, Larsen KG, eds. Proc. of the CAV 2002. Berlin, Heidelberg: Springer, 2002. 359–364. [doi: 10.1007/3-540-45657-0_29]

[11] Cavada R, Cimatti A, Dorigatti M, *et al*. The nuXmv symbolic model checker. In: Biere A, Bloem R, eds. Proc. of the CAV 2014. Berlin, Heidelberg: Springer, 2014. 334–342. [doi: 10.1007/978-3-319-08867-9_22]

[12] Lomuscio A, Qu HY, Raimondi F. MCMAS: An open-source model checker for the verification of multi-agent systems. Int'l Journal on Software Tools for Technology Transfer, 2017, 19(1): 9–30. [doi: 10.1007/s10009-015-0378-x]

[13] Su KL, Sattar A, Luo XY. Model checking temporal logics of knowledge via OBDDs. The Computer Journal, 2007, 50(4): 403–420. [doi: 10.1093/comjnl/bxm009]

[14] Alur R, Henzinger TA. Real-time logics: Complexity and expressiveness. Information and Computation, 1993, 104(1): 35–77. [doi: 10.1006/inco.1993.1025]

[15] Alur R, Henzinger TA. A really temporal logic. Journal of the ACM, 1994, 41(1): 181–203. [doi: 10.1145/174644.174651]

[16] Larsen KG, Pettersson P, Wang Y. UPPAAL in a nutshell. Int'l Journal on Software Tools for Technology Transfer, 1997, 1(1-2): 134–152. [doi: 10.1007/s100090050010]

[17] Henzinger TA, Ho PH, Wong-Toi H. HYTECH: A model checker for hybrid systems. Int'l Journal on Software Tools for Technology Transfer, 1997, 1: 110–122. [doi: 10.1007/s100090050008]

[18] Bozga M, Daws C, Maler O, *et al*. Kronos: A model-checking tool for real-time systems. In: Hu AJ, Vardi MY, eds. Proc. of the CAV'98. Berlin, Heidelberg: Springer, 1998. 546–550. [doi: 10.1007/BFb0028779]

[19] Georges M, Christoph S. Fully symbolic TCTL model checking for complete and incomplete real-time systems. Science of Computer Programming, 2015, 111: 248–276. [doi: 10.1016/j.scico.2015.08.002]

[20] McMillan KL. The SMV system—For SMV version 2.5.4. Carnegie Mellon University, 2000. https://www.cs.cmu.edu/~modelcheck/smv/smvmanual.ps

[21] Gabbay DM, Pnueli A. A sound and complete deductive system for CTL* verification. Logic Journal of the IGPL, 2008, 16(6): 499–536. [doi: 10.1093/jigpal/jzn018]

骆翔宇(1974－), 男, 博士, 副教授, CCF 专业会员, 主要研究领域为形式化方法, 模型检测, 时态逻辑, 多智能体系统.

苏开乐(1964－), 男, 博士, 教授, 博士生导师, CCF 高级会员, 主要研究领域为形式化方法, 模型检测, 多智能体系统, 难解问题的算法设计.

黄欣玥(1994－), 女, 硕士生, CCF 学生会员, 主要研究领域为形式化方法, 模型检测, 时态逻辑.

陈祖希(1981－), 男, 博士, 讲师, CCF 专业会员, 主要研究领域为形式化方法, 程序验证, 软硬件系统安全评估.

古天龙(1964－), 男, 博士, 教授, 博士生导师, CCF 高级会员, 主要研究领域为形式化方法, 人工智能安全, 人工智能伦理, 数据治理.

郑黎晓(1983－), 女, 博士, 副教授, CCF 专业会员, 主要研究领域为形式语言与自动机, 数据库理论, 软件测试.