计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2018,27(9):176−181 [doi: 10.15888/j.cnki.csa.006546]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

多核模型检测工具 CTAV 的实现与优化①

赵　威

(中国科学院 软件研究所 计算机科学国家重点实验室, 北京 100190)
(中国科学院大学, 北京 100049)

摘　要: 在计算机计算能力大大增强的时代, 为了提高对时间自动机进行空性检测的效率, 进一步高效利用多核处理器的优势, 研究了利用 Büchi 自动机的多核空性判定算法改造 CTAV, 使它成为一款时间自动机模型关于线性时序逻辑的多核模型检测工具, 从而提高模型检测的效率. 通过对符号化状态之间包含关系的研究, 利用这种状态之间的包含关系更快的找到接收路径并避免不必要的状态展开, 实现了多核模型检测算法的优化, 对比了一些常见模型的验证数据, 取得了更好的效果.

关键词: 时间自动机; 线性时序逻辑; 多核并行模型检测; 符号化状态

引用格式: 赵威.多核模型检测工具 CTAV 的实现与优化.计算机系统应用,2018,27(9):176–181. http://www.c-s-a.org.cn/1003-3254/6546.html

Implementation and Optimization of Multi-Core Model Checker CTAV

ZHAO Wei

(State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China)
(University of Chinese Academy of Sciences, Beijing 100049, China)

Abstract: In the era of the increased computation capacity of computer, in order to improve the efficiency of checking emptiness of timed automaton, and make more efficient using of the advantages of multi-core processors, we use multi-core emptiness checking algorithm of timed Büchi automata to rebuild CTAV, making it become a multi-core model checker for linear temporal logic, which improves the efficiency of model checking. Since there are equivalence and inclusion relationship between symbolic states, by making use of this relationship, a model checker can find accepted path faster and avoid unnecessary state explorations. Thus, checking takes less time. Finally, the effectiveness of presented method is demonstrated by case study.

Key words: timed automata; Linear Temporal Logic (LTL); multi-core model checking; symbolic state

实时系统是指这样的一类计算机应用系统, 它要求在一定时限内及时的响应外部事件, 并在一定时限内完成事件的处理. 在这类系统中, 确保其准确性和可靠性非常重要, 一旦出现缺陷, 带来的经济财产损失都是巨大的. 实时系统的模型检测[1]是确保实时系统的可靠性和正确性的一种重要技术, 它通过遍历状态空间寻找是否存在不满足规约的状态或路径来验证系统是否正确可靠.

时间自动机模型是实时系统最广泛使用的一种数学模型, 针对实时系统的模型检测工具主要采取时间自动机作为形式模型, 以计算树逻辑 (Computation Tree Logic, CTL)[1]或线性时序逻辑 (Linear Temporal Logic, LTL)[1]作为性质描述语言. 典型的代表有 UPPAAL、KRONOS、RED 以及 CTAV 等.

CTAV[2]是中科院软件所在 2009 年实现的一个关于时间自动机的符号化模型检测工具, 在关于时间自

① 收稿时间: 2018-02-01; 修改时间: 2018-02-28; 采用时间: 2018-03-08; csa 在线出版时间: 2018-08-16

动机的模型检测中它是第一个针对线性时序逻辑 LTL 的模型检测工具, 它先将时间自动机关于 LTL 的模型检测化归为时间 Büchi 自动机的空性检测, 然后文献[2]中证明了时间 Büchi 自动机的空性检测可通过 zone 抽象化归为 Büchi 自动机的空性检测, 这样就可以利用 Büchi 自动机的空性检测算法来完成时间自动机关于 LTL 的模型检测. 基于这一方法, 文献[3]最早实现了时间自动机关于 LTL 的符号化模型检测, 这里符号化的含义是指模型检测使用的是时间自动机的基于 zone 的符号化语义. 文献[2]中使用的 Büchi 自动机的空性检测算法是单线程的. 2013 年, 文献[4–7]在其多核模型检测工具 LTSmin 的基础上利用文献[2]中的方法实现了时间自动机关于 LTL 的多核模型检测工具 LTSmin+opaal, 使得验证效率有了较大的提高. 与此同时捷克人在他们的多核模型检测工具 DIVINE3.0[8,9] 中也实现了时间自动机关于 LTL 的多核模型检测. 本文我们将在文献[10–12]中给出的关于 Büchi 自动机的多核模型检测算法的基础上在 CTAV 工具中实现时间 Büchi 自动机关于 LTL 的多核模型检测, 与 LTSmin+opaal 和 DIVINE3.0 相比, 在系统描述语言方面, 我们工具支持比时间自动机表达能力更强的时间 Büchi 自动机, 其它两个只支持时间自动机模型, 这一点对证明活性性质至关重要, 在性质描述语言方面, 由于 CTAV 本身可以支持 LTL 的实时扩展 $MTL_{0,\infty}$ (Metric Temporal Logic)[3], 所以我们的工具将能比其它两个工具支持更多的性质验证.

本文的主要内容如下: 研究了多种 Büchi 自动机的多核空性判定算法, 基于文献[10–12]实现时间 Büchi 自动机关于 LTL 的多核模型检测, 同时对比了多种多核算法在 CTAV 工具下的效果, 并基于符号化状态之间的包含关系给出了一种优化方法, 并最终对比了 divine 的实验结果.

1 基本概念

1.1 时间自动机

时间自动机[13]就是带有时钟集的有限自动机. 时钟集是有限个时钟的集合, 每个时钟都是一个取值为非负实数的变量. 时间自动机状态之间的转换要满足时钟约束才可能发生. 时间自动机的状态可以附加上“节点不变量”的属性, 这也是一个时钟约束, 用来保证状态不会保持在原地不动.

时间自动机的形式化定义[14]是一个六元组 $M=<L, l_0, \Sigma, X, Inv, E>$, 其中,

(1) X 为时钟变量的有限集合, $C(X)$ 为 X 上时钟约束的集合, 其定义为:

$$\psi ::= true|x \sim c|\psi 1 \wedge \psi 2 \quad (1)$$

其中, $x \in X$, $\sim \in \{<, \leq, ==, >, \geq\}$, c 是非负整数.

(2) L 是节点的非空有限集合, $l_0 \in L$ 是初始节点.

(3) $Inv : L \mapsto C(X)$为每个节点指定时钟约束, 称为节点不变量.

(4) Σ 为标号的有限集合.

(5) $E \in L \times C(X) \times \sum \times 2^X \times L$是迁移的集合. 迁移 $(l_0, g, a, Y, l_1) \in E$ 表示在满足约束条件 g 的前提下, 通过标号为 $a \in \Sigma$ 的迁移, 可以从节点 l_0 迁移到它的后继节点 l_1, 同时, 时钟集$Y(Y \subseteq X)$中的时钟被重置为 0.

时间自动机 M 的迁移动作主要包含 2 种, 分别是延迟迁移和离散迁移.

1.2 Zone[15]

假设 X 为时钟变量的有限集合, $\Phi(X)$ 为时钟变量集合 X 上扩展时钟约束的集合, 扩展时钟约束的语法定义如下:

$$\psi ::= true|x \sim c|x-y \sim c|\psi 1 \wedge \psi 2 \quad (2)$$

式中: $x, y \in X$, $\sim \in \{<, \leq, >, \geq\}$, c 是非负整数. 对于时钟赋值 u, u 满足扩展的时钟约束的充分必要条件是在赋值 u 下为真.

Zone 通常用 DBM (Difference Bound Matrices)[16] 来表示.

DBM 是一个矩阵, 矩阵中的每一个元素是一组时钟的两两差值. 其定义了一个值为 0 的绝对时钟, 对于 DBM D 中的元素 D_{ij} 表示其中的第 i 行第 j 列, 它表示 zone 中时钟变量 i 和时钟变量 j 形成的时钟约束.

例如, 一个时钟区域 $x \geq 6 \wedge 0 \leq y \leq 3 \wedge y \leq x-1$, 引入绝对时钟后可以将表达式改写成 $0-x \leq -6 \wedge 0-y \leq 0 \wedge y-0 \leq 3 \wedge y-x \leq -1$, 用 3×3 矩阵表示, 分别为:

$$\begin{bmatrix} 0-0 \leq 0 & 0-x \leq -6 & 0-y \leq 0 \\ x-0 \leq \text{INF} & x-x \leq 0 & x-y \leq \text{INF} \\ y-0 \leq 3 & y-x \leq -1 & y-y \leq 0 \end{bmatrix} \text{和}$$

$$\begin{bmatrix} \leq 0 & \leq -6 & \leq 0 \\ \leq \text{INF} & \leq 0 & \leq \text{INF} \\ \leq 3 & \leq -1 & \leq 0 \end{bmatrix},$$

其中 INF 表示无穷大.

1.3 符号化语义

假设 D 表示一个 zone, Y 是时钟集合, 对其延迟和

重置定义如下:

$D\uparrow=\{u+d|u\in D, d\in R^{\geq 0}\}$, 表示 zone 的延迟定义;

$r(D)=\{u[Y:=0]|u\in D\}$, 表示 zone 的重置.

时间自动机 (如图 1) 的符号化状态是 (l, D), l 是节点, D 是 zone. 初始状态$(l_0, D_0\uparrow\wedge Inv(l_0))$, l_0 是初始节点, D_0 表示所有时钟都为 0.

时间自动机的符号化语义为以 (l, D) 为状态的迁移系统, 迁移$(l', D')\xrightarrow{a}(l'', D'')$, 如果有 $(l, g, a, Y, l_1)\in E$, 并且 $D''=((D'\wedge g)[Y:=0]\wedge Inv(l''))\uparrow\wedge Inv(l'')$, D''不为空.

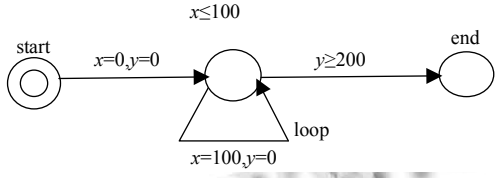

图 1 简单的自动机

图 1 中的时间自动机在符号化语义下的一个可能迁移序列如下:

(start, $x>=0\wedge y>=0\wedge x=y$) →

(loop, $0<=x<=100\wedge 0<=y<=100\wedge y=x$) →

(loop, $0<=x<=100\wedge 100<=y<=200\wedge y=x+100$) →

(loop, $0<=x<=100\wedge 200<=y<=300\wedge y=x+200$) →

(loop, $0<=x<=100\wedge 300<=y<=400\wedge y=x+300$) →

(end, $x>=0\wedge y>=300\wedge y=x+300$)

其中 loop 表示自循环迁移, 并且这个迁移前后的状态不相同, 这个迁移系统有无穷多个状态.

2 原理

2.1 模型检测

给定时间自动机 $M=(L, l_0, \Sigma, x, I, E)$ 和 LTL 公式 φ, 要检验 LTL 性质是否被时间自动机 M 所满足. 在文献[2]中提出了一种检测算法, 主要思路是先将公式 $\neg\varphi$ 转化为一个 Büchi 自动机 $M\neg\varphi$, 然后检验 M 与 $M\neg\varphi$ 的合成 $M\|M\neg\varphi$ 是否为空, 这样将 LTL 性质的检验转换为对 $M\|M\neg\varphi$ 的空性判定, 而对此类空性判定, 通过使用基于 zone 的符号化语义将 $M\|M\neg\varphi$ 转化成一个 Büchi 自动机, 然后利用 Büchi 自动机空性检测算法来完成.

将检验 LTL 性质是否被时间自动机所满足转换为对 Büchi 自动机空性检测的结果, 若空性检测的结果为空, 则性质被自动机 M 所满足; 反之若空性检测的结果不为空, 则性质不被自动机 M 所满足.

2.2 符号化状态

在符号化状态抽象表示过程中, 状态的表示我们是采取 DBM 来表示的, 因此状态之间会产生集合的包含关系, 在状态展开的遍历过程中我们采取判断状态是否遍历过或者已经在被删除状态集合中来对其进行后续处理, 在文献[4]中我们得到以下引理.

引理 1. 在状态空间中如果有$S_k\subseteq S_j$, 并且 S_k 可以到达接受环, 那么 S_j 也可以到达.

证明: 假设 S_j 不能到达可接受环, 而 S_k 能到达, 而 $S_k\subseteq S_j$代表自动机中各个进程所处节点一样, 仅仅时间约束不同, 如图 2, S_j 如果不能到达可接受环, 那么 S_k 肯定不能, 与条件矛盾.

引理 2. 在状态空间中如果有$S_k\subseteq S_j$, 并且 S_k 到 S_j 之间包含一个接受状态, 那么 S_k 可以到达接受环.

证明: 状态 S_j 是状态 S_k 的超集, 而状态 S_k 能到达 S_j, 由 DBM 表示的时间约束是凸性的, 可以将 S_j 分为 S_k 和 S_j–S_k 两部分, 故可得发现环.

由引理 1 还可以得出以下结论, 在状态空间中如果有$S_j\subseteq S_k$, 并且 S_k 不可以到达接受环, 那么 S_j 也不可以到达. 这个结论有助于避免不必要的状态展开, 由引理 1 逆否命题可得.

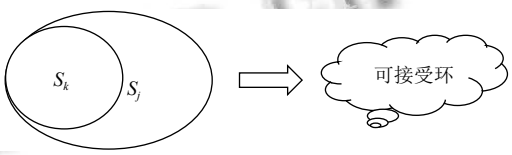

图 2 S_k 与 S_j

3 多核模型检测工具的实现与优化

3.1 算法简介

文献[10]中提出了一种基于 Tarjan[17]和 Dijkstra[18]算法而形成的多核并行空性检测算法, 并且实现在 SPOT 库中, 这个算法的主要思路如下: 采用共享的终止条件 stop 和一个并查集结构使得多个线程可以共享状态遍历结果, 而状态展开过程就是一个深度优先遍历过程, 在这个过程中的每一步会首先找出状态节点是处在未遍历状态、已经遍历或者已经遍历完全的状态. 从初始状态开始, 把遍历序号依次赋值给展开状态, 其中当前图的所有状态的序号都不为 0. 如果当前状态

的所有后继都已经被遍历过, 并且仍然不满足接受条件, 则说明该状态和其后续状态不在接受路径中, 将它们的遍历序号赋值为 0, 表示已经从当前图中删除, 放入被删除状态中. 在找到一条接受路径或者完全展开全部状态后停止. 算法伪代码如图 3 和图 4.

```
Ec(str:Strategy, tid:int)          // 主流程
  Seed(tid)
  Push(q0)          //push  表示  将状态加入  dfs
  While  栈  dfs  非空  and  stop != 1
    st = dfs.top()
    If st  存在后继:
      随机选取迁移      t
      switch  Get_status(t.dst)
        case Dead
          Skip
        case LIVE
          Update(t.acc,t.dst)          // 更新  相
关状态标记,      Tarjan  和 D ijkstra  实现  不同
        case UNKNOWN
          Push(t.dst)
      else
        Pop(step)      // 状态  后继遍历完全
```

图 3 算法主流程

在算法展开状态过程中假设 S 表示状态节点, 有迁移 $S_0 \rightarrow S_1$, 则 S_1 是当前遍历到的状态节点, 有如下可能:

(1) S_1 没有遍历, 返回 UNKNOWN(表示未遍历), 将其放入深度优先栈 *dfs* 中, 同时加入 hash 表中, 其中 dfs 表示存储遍历过程中状态的栈, 其每个元素由<当前状态, 当前迁移的接收条件, 标识符, 当前状态未遍历的后继>这样的四元组构成, hash 表存储所有状态并给予唯一的标识符, Dead 表示删除状态集合, 即已遍历完全无法满足接受条件的状态集合, Livenum 存储活着的（不属于 Dead 集合的）状态.

(2) S_1 出现在了 Livenum 表中, 即出现在当前图, 说明发现了环, 这样我们需要判断该环是否满足接受条件, 如果满足则赋值 stop=1, 标记终止符为 1 表示结束算法, 当前线程直接返回, 其它线程收到 stop=1, 也立即返回, 表示找到了路径, 性质不满足.

(3) S_1 出现在 Dead 中, 即 S_1 的所有后继已经遍历, 不会出现满足条件的接受环, 那么直接跳过.

```
Main(string:Strategy)
  Stop=0
  uf.make_set(<dead, 空集 >)
  if(str==Mixed)
    str=Dijkstra
    前一半线程采取 dijkstra策略
    str=Tarjan
    后一半线程采取 tarjan 策略
  else
    全部采取strategy策略
Get_status(q)        // 判断  状态属性
  If q∈Livenum
    return  LIVE
  else q∈Dead
    return DEAD
  else
    return UNKNOWN
```

图 4 混合算法

3.2 算法优化

基于此我们采取了相应的改进措施, 通过文献[3]应用于 LTSmin 多线程算法的改进, 我们将其应用到了我们算法上, 主要改进思路有以下两点:

如果有迁移 $S_1 \rightarrow S_2$, 如果 Get_status(S_2) 返回 UNKNOWN, 我们增加以下判断:

(1) 如果 S_2 是已遍历而且非死亡状态中的状态的超集, 即存在 S_3 属于 livenum, 并且 S_3 包含于 S_2, 则我们可以将其作为发现一个环, 以此判断找到了环, 进行是否包含接受条件的判断.

(2) 如果 S_2 是 Dead 状态的子集, 直接由引理 1 得, S_2 后继肯定不会出现环, 可以直接跳过.

基于这两点, 我们对算法进行了相应的改动, 改动伪代码如图 5, 算法及优化全部实现在 CTAV 工具中, 我们对结果进行了对比研究, 主要基于展开状态的变化, 如表 1 结果显示, 在部分模型取得了不错的效果.

3.3 算法效果比较

在多线程算法中, 单一算法在发现环的过程中往往在某一方面表现十分优秀, 而在某些例子也存在极差的表现, 在工具实现过程中基于 Tarjan、Dijkstra 算

法, 我们比较了这两种多核算法的区别, Tarjan 算法主要表现在每发现环的时候只 pop 最后的边, 而 Dijkstra 算法在发现环的时候将当前图除根节点全部 pop, 这会直接导致发现接受图的效率问题, 为此, 我们采取了混合算法, 让一部分线程跑 Tarjan, 一部分线程跑 Dijkstra, 在实验中, 我们通过给线程加上整数标识, 让奇数线程运行 Dijkstra 算法, 偶数线程运行 Tarjan 算法. 最终表现结果如表 2, 在结果中我们发现混合算法表现良好, 在各种模型下能取得相对于单一算法更佳的时间效率效果.

```
Get_status(q)
  If  q ∈ Livenum      or   q ⊇ liveSub
    // liveSub   为 增加的按超集查找的保存
live  状态  的数据结构
    return  LIVE
  else  q ∈ Dead      or   q ⊆ deadChild
    //deadChild    为 按子集查找的保存被删
除状态的数据结构
    return DEAD
  else
    return UNKNOWN
```

图 5 算法改进

表 1 改进效果对比

模型	原多核算法 (状态数)	改进后 (状态数)
train-gate 6train	31 610	26 248
train-gate 4train	914	826
fischer	26 496	26 024
csmacd 6 进程	766	750
fddi	327	305

表 2 混合算法效果对比结果

	Tarjan	Dijkstra	Mixed
train-gate 6train	0.48	0.61	0.48
fischer	0.015	0.015	0.015
csmacd	0.018	0.045	0.018
viking	0.017	0.017	0.017
fddi	0.03	0.03	0.03

4 实验研究

最终我们完成了多核模型检测工具的开发和优化, 如表 3 显示 train-gate 模型下性质“[](“trans(0).Appr”→<>(“trains(0).Cross”))”为例线程数量和模型复杂化后时间效率的提升变化, 表中“原”代表原单核工具, 后续数字表示新工具采取的线程数量. 实验效果显示多核模型检测工具的效果提升是显著的, 基本与线程数量成线性相关, 当然由于机器核数的影响, 在达到峰值即机器支持的线程核数后有所下降.

表 3 多核 CTAV 对比单核表现结果

	原	2	4	6	8	16
5train	0.22	0.13	0.107	0.110	0.114	0.16
6train	1.339	0.849	0.479	0.479	0.473	0.736
7train	10.34	5.799	3.32	3.32	3.16	4.091
8train	89.717	59.689	27.135	27.135	23.754	32.471

同时, 我们对比了 divine 工具和我们工具在 4 线程下的时间效率, 在部分模型中取得了更好的效果, 如表 4.

表 4 divine 和 ctav 对比

	Divine	CTAV
fddi	0.17	0.16
csmacd	0.04	0.02
train-gate	0.06	0.03
fddi2	0.33	0.04

5 结束语

通过采用基于 Tarjan 和 dijkstra 的多线程并行的模型检测算法, 我们提高了 CTAV 工具对 CPU 的利用率, 以此提高了 CTAV 工具的时间效率, 同时通过符号化状态的包含关系我们提出了多核算法下的优化方案, 对状态空间的约减有一定的优化作用, 同时在 CTAV 工具下对比了多种多核空性检测算法的效率, 并采取混合多线程算法完成了工具的开发. 当然, 多核模型检测工具的优化仍然需要进一步的研究, 简单的利用符号化状态的包含关系约减的优化效果并不明显, 进一步的优化也是我们下一步的工作. 同时, 最新的研究中也产生了很多采取分布式资源对系统进行模型检测的研究, 如文献[19]提出采用分布式资源对发布订阅结构系统进行模型检测. 对于如何利用分布式系统进一步提高工具的效率也是我们下一步的研究方向.

参考文献

1 Clarke EM, Grumberg O, Peled DA. Model Checking. The MIT Press, 1999.
2 Li GY. Checking timed Büchi automata emptiness using LU-abstractions. Proceedings of the 7th International Conference

on Formal Modeling and Analysis of Timed Systems. 2009: 228–242.

3 http://lcs.ios.ac.cn/~ligy/tools/CTAV_LTL/.

4 Laarman A, Olesen MC, Dalsgaard AE, *et al*. Multi-core emptiness checking of timed Büchi automata using inclusion abstraction. In: Sharygina N, Veith H, eds. CAV 2013. Lecture Notes in Computer Science, vol 8044. Springer. Berlin, Heidelberg. 2013. 968–983. [doi: 10.1007/978-3-642-39799-8_69]

5 Laarman A, Langerak R, van de Pol J, *et al*. Multi-core nested depth-first search. In: Bultan T, Hsiung PA, eds. Automated Technology for Verification and Analysis. ATVA 2011. Lecture Notes in Computer Science, vol 6996. Springer. Berlin, Heidelberg. 2011. 321–335. [doi: 10.1007/978-3-642-24372-1_23]

6 Evangelista S, Petrucci L, Youcef S. Parallel nested depth-first searches for LTL model checking. In: Bultan T, Hsiung PA, eds. Automated Technology for Verification and Analysis. ATVA 2011. Lecture Notes in Computer Science, vol 6996. Springer. Berlin, Heidelberg. 2011. 381–396. [doi: 10.1007/978-3-642-24372-1_27]

7 Evangelista S, Laarman A, Petrucci L, *et al*. Improved multi-core nested depth-first search. In: Chakraborty S, Mukund M, eds. Automated Technology for Verification and Analysis. ATVA 2012. Lecture Notes in Computer Science, vol 7561. Springer. Berlin, Heidelberg. 2012. 269–283. [doi: 10.1007/978-3-642-33386-6_22]

8 Barnat J, Brim L, Rockai P. DiVinE multi-core–a parallel LTL model-checker. In: Cha S, Choi JY, Kim M, *et al*., eds. Automated Technology for Verification and Analysis. Lecture Notes in Computer Science, vol 5311. Springer. Berlin, Heidelberg. 2008. 234–239. [doi: 10.1007/978-3-540-88387-6_20]

9 Barnat J, Brim L, Havel V, *et al*. DiVinE 3.0–an explicit-state model checker for multithreaded C & C++ programs. In: Sharygina N, Veith H, eds. Computer Aided Verification. CAV 2013. Lecture Notes in Computer Science, vol 8044. Springer. Berlin, Heidelberg. 2013. 863–868. [doi: 10.1007/978-3-642-39799-8_60]

10 Renault E, Duret-Lutz A, Kordon F, *et al*. Parallel explicit model checking for generalized Büchi automata. TACAS'15, vol. 9035 of LNCS. Springer. 2015. 613–627.

11 Renault E, Duret-Lutz A, Kordon F, *et al*. Variations on parallel explicit emptiness checks for generalized Büchi automata. International Journal on Software Tools for Technology Transfer, 2017, 19(6): 653–673. [doi: 10.1007/s10009-016-0422-5]

12 Renault E, Duret-Lutz A, Kordon F, *et al*. Three SCC-based emptiness checks for generalized Büchi automata. In: McMillan K, Middeldorp A, Voronkov A, eds. Logic for Programming, Artificial Intelligence, and Reasoning. LPAR 2013. Lecture Notes in Computer Science, vol 8312. Springer, Berlin, Heidelberg. 2013. 668–682. [doi: 10.1007/978-3-642-45221-5_44]

13 Alur R, Dill DL. A theory of timed automata. Theoretical Computer Science, 1994, 126(2): 183–226. [doi: 10.1016/0304-3975(94)90010-8]

14 晏荣杰. 基于整数时间的实时系统符号化模型检测技术[博士学位论文]. 北京: 中国科学院软件研究所, 2007.

15 Henzinger TA, Nicollin X, Sifakis J, *et al*. Symbolic model checking for real-time systems. Journal of Information and Computation, 1994, 111(2): 193–244. [doi: 10.1006/inco.1994.1045]

16 Bengtsson J, Yi W. Timed automata: Semantics, algorithms and tools. In: Desel J, Reisig W, Rozenberg G, eds. Lectures on Concurrency and Petri Nets. ACPN 2003. Lecture Notes in Computer Science, vol 3098. Springer, Berlin, Heidelberg. 2004. 87–124. [doi: 10.1007/978-3-540-27755-2_3]

17 Tarjan R. Depth-first search and linear graph algorithms. SIAM Journal on Computing, 1972, 1(2): 146–160. [doi: 10.1137/0201010]

18 Dijkstra EW. EWD 376: Finding the maximum strong components in a directed graph. http://www.cs.utexas.edu/users/EWD/ewd03xx/WD376.PDF. [1973-05].

19 Ruiz V, Diaz G, Cambronero ME. Timed automata modeling and verification for publish-subscribe structures using distributed resources. IEEE Transactions on Software Engineering, 2017, 43(1): 76–99. [doi: 10.1109/TSE.2016.2560842]