软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2024,35(9):4310−4323 [doi: 10.13328/j.cnki.jos.007139]

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

Büchi 自动机确定化分析工具*

马润哲, 田 聪, 王文胜, 段振华

(西安电子科技大学 计算机科学与技术学院, 陕西 西安 710071)

通信作者: 田聪, E-mail: ctian@mail.xidian.edu.cn; 王文胜, E-mail: wswang@xidian.edu.cn

摘 要: 无限字自动机的确定化是理论计算机研究重要的一部分, 在形式化验证, 时序逻辑, 模型检测等方面有重要应用. 自 Büchi 自动机提出半个世纪以来, 其自动机的确定化算法始终是其中的基础. 有别于当初只是在理论上对其大小上下界的探索, 利用日新月异的高性能计算机技术不失为一种有效的辅助手段. 为了深入研究非确定性 Büchi 自动机确定化算法的实现过程, 希望重点研究确定化过程中的索引能否继续被优化的问题, 实现确定化研究工具 NB2DR. 可以对非确定性 Büchi 自动机进行高效的确定化, 并通过工具提供的分析其确定化过程来达到对其确定化算法改进的目的. 通过对生成的确定性无限字自动机的索引的深入分析来探索相关索引的理论. 该工具还实现了可以根据需要的 Büchi 自动机的大小与字母表参数, 生成确定化的 Rabin 自动机族, 亦可以反向根据需要的指定索引的大小来生成全部 Büchi 自动机族, 测试生成无限字自动机的等价性等功能.

关键词: Büchi 自动机; Rabin 自动机; 无限字自动机确定化

中图法分类号: TP301

中文引用格式: 马润哲, 田聪, 王文胜, 段振华. Büchi自动机确定化分析工具. 软件学报, 2024, 35(9): 4310–4323. http://www.jos.org.cn/1000-9825/7139.htm

英文引用格式: Ma RZ, Tian C, Wang WS, Duan ZH. Tool for Determinization of Büchi Automata. Ruan Jian Xue Bao/Journal of Software, 2024, 35(9): 4310–4323 (in Chinese). http://www.jos.org.cn/1000-9825/7139.htm

Tool for Determinization of Büchi Automata

MA Run-Zhe, TIAN Cong, WANG Wen-Sheng, DUAN Zhen-Hua

(School of Computer Science and Technology, Xidian University, Xi'an 710071, China)

Abstract: The ω automata determinization is an important part of theoretical computer research and has important applications in formal verification, sequential logic, and model checking. Since the Büchi automata was proposed for half a century, its deterministic algorithm has always been the basis. Different from the exploration of the upper and lower bounds of its size in theory at the beginning, the use of ever-changing high-performance computer technology is an effective auxiliary means. To deeply study the implementation process of the deterministic algorithm of non-deterministic Büchi automata, this study focuses on the question of whether the index can continue to be optimized and realizes the deterministic research tool NB2DR. The non-deterministic Büchi automata can be determined efficiently, and the determinization algorithm can be improved by analyzing the determinization process provided by the tool. A theoretical upper bound on correlation indexing is explored through an in-depth analysis of the indexing of generated deterministic ω automata. The tool also realizes that the deterministic Rabin automaton family can be generated according to the size of the required Büchi automaton and the alphabet parameters. It can also be reversed to generate all the Büchi automata families according to the size of the specified index required and test the equivalence of ω functions.

Key words: Büchi automata; Rabin automaton; ω automata determinization

---

* 基金项目: 国家自然科学基金 (62192734); 国家重点研发计划 (2018AAA0103202)
本文由"形式化方法与应用"专题特约编辑曹钦翔副教授、宋富研究员、詹乃军研究员推荐.
收稿时间: 2023-09-18; 修改时间: 2023-10-30; 采用时间: 2023-12-13; jos 在线出版时间: 2024-01-05
CNKI 网络首发时间: 2024-05-18

自动机的确定化是理论计算机领域的基础问题之一, 在形式化验证与模型检测中具有重要的应用[1−5]. 自动机的确定化与求补密切相关, 求补是指对一个接受语言为L的自动机A, 其求补是指找到一个自动机$\tilde{A}$, 满足$\tilde{A}$接受的语言为L的补集. 为了验证系统是否满足期望的性质, 首先将系统表达为自动机A, 性质表达为自动机B, 然后对B进行求补, 再与A取交, 得到的自动机C, 若C接受的语言为空集, 则可以判定系统满足该性质. 这个问题和自动机的确定化是强烈相关的. 假如一个自动机是确定化的, 那么我们只需要对这个自动机对偶化, 即可得到其求补的自动机. 并且, 对于 Büchi 自动机, 尽管在理论复杂度方面, 不需要确定化的求补方法优于基于确定化的求补方法, 但文献 [6] 中的实验结果表明基于确定化的求补方法在实际中具有更好的效果. 无限字自动机的确定化是计算复杂度理论中的重要问题, 是逻辑判定过程的基础[7], 也有助于解决无限博弈求解问题; 目前实际效果最好的公式博弈求解工具[8]便是基于确定性 parity 自动机实现的. 无限字自动机的确定化还应用于马尔科夫决策过程的推理、基于监控的运行时验证等.

对于定义在有限字上的有限状态自动机来说, 其确定化是很明确的. 通常对于一个状态为n的非确定的自动机来说, 可以采用子集构造法来得到状态为2^n的确定性自动机. 其核心思想就是通过幂集枚举出自动机运行中所有可达的状态合集, 只需要将这些所有可能都看作新的自动机的状态, 那么这个新的自动机将是确定化的.

但是对于定义在无限字上的有限状态自动机, 同样的问题将变得更为复杂, 因而无限字自动机的确定化研究具有重要意义. Büchi 自动机是无限字自动机的一种, 在基于线性时序逻辑 (LTL) 的模型检测过程中, LTL 等可以等价地转换成 Büchi 自动机, 对该逻辑公式的可满足性检查相应的会变为对 Büchi 自动机的确定化过程, Büchi 自动机的确定化在模型检测领域有重要的作用.

Büchi [1]在他的论文中证明一元二阶逻辑是可判定性时, 首先介绍了 Büchi 自动机. 这是有限自动机在无限字上的推广. 与普通有限字上的自动机相比, 不同点在于其接受条件, 很自然的, 区别于普通自动机的接受条件, 即“状态序列访问到接受节点”, Büchi 自动机的接受条件为“状态序列访问无限多次接受节点”. 此外, 有限字的确定性自动机和非确定性自动机的表达能力是一致的, 而区别于此, 确定性的 Büchi 自动机的表达能力严格低于非确定性的 Büchi 自动机. 换言之, 非确定性的 Büchi 自动机不存在一种算法来得到确定性的 Büchi 自动机.

为了研究 Büchi 自动机的确定化问题, 许多表达能力更强的无限字上的自动机被提出, 包括 Rabin 自动机, Streett 自动机, parity 自动机, Emerson-Lei 自动机[2,7,9−11]等. 这些自动机主要通过添加复杂的接受条件限制, 来提高其表达能力, 以接受更广泛的无限字, 拥有更广阔的语言集. 经过半个世纪左右的研究, 如今 Büchi 自动机确定化理论体系基本完备[3,4,12,13], 相关自动机的理论状态上下界很多都得到证明[14], 但仍有不少问题是开放的, 而随着计算机技术性能的提升与算法的进步, 很多不好得到的大型自动机状态也可以通过工具进行仿真模拟运行.

然而目前这方面的工具并不多, 包括 SPOT, GOAL[15−17]等. 本工具参考了相关工具的优点, 具备自动机的相关建模能力, 可以选择相关确定化算法对输入的自动机进行建模, 同时根据实际需要可以完成对自动机族的筛选, 自动机确定化过程中对接受条件的分析, 以及比较, 还可以进行高效的等价性测试, 重点支持历史树算法的分析推演, 探究历史树的节点规律, 为进一步的确定化研究提供帮助. 在对非确定自动机的确定化研究中, 非确定性自动机的确定化会碰到各种问题, 即使有很多理论上下界的数值支撑, 但是实际应用中的自动机远远达不到这些理论边界, 我们推出的工具可以在已知的理论算法基础上进行高效的确定化操作. 同时可以观测确定化中的自动机的状态, 对进一步研究做出实验贡献.

本文第 1 节介绍基础自动机理论的相关知识. 第 2 节介绍工具的整体框架与功能. 第 3 节展示工具的效果. 第 4 节介绍相关工作. 第 5 节总结全文并展望未来发展方向.

1 基础知识

本节将介绍基础自动机理论相关知识.

1.1 无限字与无限字自动机

自动机是一个五元组$A = \{\Sigma, Q, Q_0, \delta, \lambda\}$, 其中$\Sigma$是字母表, 表示有限字母的集合; Q是状态集, 表示有限状态的

集合; Q_0 是初始状态集, 表示初始状态的集合; δ 是转移函数, 表示一个状态读入一个字母可达状态集合的映射 $\delta \subseteq Q \times \Sigma \times Q$; λ 是接受条件.

有限字 w 是由一串字母构成的有序序列 $w = a_1a_2a_3 \ldots a_n$. 其中 $a_i \in \Sigma (1 \leqslant i \leqslant n)$. 无限字为无限长度的字母序列 $\alpha = a_1a_2a_3 \ldots$. 其中, 常见的无限字为循环无限字, 类似于无限循环小数, 表示为 $\alpha = \alpha_1(\alpha_2)^{\omega}$, 其中 α_1 为循环前缀, α_2 为循环主体.

给定无限字 α, 以及自动机 A, 那么 α 在 A 上的运行序列为 $\rho = \rho_0\rho_1 \ldots \in Q^{\omega}$, 满足 $\rho_0 \in Q_0$ 并且 $< \rho_i, w_i, \rho_{i+1} > \in \delta$ 对所有 $i \geqslant 0$ 成立. 若 ρ 满足接受条件 λ, 则称 α 是可以被自动机 A 接受的, 所有被接受的字组成的集合则是自动机 A 的语言, 记作 $L(A)$.

对自动机 A 和长度为 l 的字 w, 转移图 ($\delta \sim graph$) 记为 $G_w^A = (V_w^A, E_w^A)$. 其中顶点集 $V_w^A = \{\langle p, i\rangle \mid p \in Q, 0 \leqslant i \leqslant l\}$, 边集 $E_w^A = \{\langle\langle p, i\rangle, \langle q, i+1\rangle\rangle \mid \langle p, w(i), q\rangle \in \delta\}$. 该图的意义在于可以说明对于状态 p 经由字 w 到达 q, 等价于在 G_w^A 图中存在一条路径从 $\langle p, 0\rangle$ 到达 $\langle q, l\rangle$.

ω 自动机是自动机的一种, 区别在于接受条件不同. 一个无限字能否被 ω 自动机接受取决于这个无限字在自动机上的运行 ρ 与该自动机接受条件的关系. 我们用 $Inf(\rho)$ 指该运行中出现无限次的状态集合, 这里主要介绍本文涉及的 ω 自动机.

Büchi 自动机: $\lambda = F \subseteq Q$, ρ 是可接受的当且仅当 $Inf(\rho) \bigcap F \neq \varnothing$.

Generalized Büchi 自动机: $\lambda = \{F_1, F_2, F_3, \ldots, F_k\}$, 其中 $F_i \subseteq Q (1 \leqslant i \leqslant k)$. ρ 是可接受的当且仅当对于 $\forall i (1 \leqslant i \leqslant k)$, 都有 $Inf(\rho) \bigcap F_i \neq \varnothing$.

Rabin 自动机: $\lambda = \{\langle G_1, B_1\rangle, \ldots, \langle G_k, B_k\rangle\}$, 其中, $G_i, B_i \subseteq Q (1 \leqslant i \leqslant k)$, 称 $\langle G_i, B_i\rangle$ 为 Rabin pair. ρ 是可接受的当且仅当 $\exists i (1 \leqslant i \leqslant k)$, 使得 $Inf(\rho) \bigcap G_i \neq \varnothing$, 并且 $Inf(\rho) \bigcap B_i = \varnothing$.

1.2 自动机的确定化与 Büchi 自动机的确定化

一个自动机 $A = \{\Sigma, Q, Q_0, \delta, \lambda\}$ 被称作确定的, 当且仅当满足条件: $|Q_0| = 1$ 并且对所有 $p \in Q, a \in \Sigma$, 都有 $\{|\{q \in Q \mid \langle p, a, q\rangle \in \delta\}|\} \leqslant 1$. 将一个自动机由非确定性转变为确定性自动机被称为自动机的确定化.

对于 Büchi 自动机来说, Büchi 自动机确定化和 Büchi 自动机的求补, 是成对出现的, 如果一个 Büchi 自动机是确定化的, 那么其补自动机就可以通过对其接受条件的对偶化直接得出, 而自动机求补在模型检测领域有广泛重要的应用, 这反映出了 Büchi 自动机的确定化在基础理论方面具有非常重要的意义.

1.3 基于状态的自动机和基于转移的自动机

通常我们的自动机都是基于状态的, 也就是接受条件的集合元素是状态. 但是随着确定化理论的研究, 往往基于转移的自动机是比较容易获得的. 另外, 线性时序逻辑 LTL 公式也可以直接转化成基于转移的 generalized Büchi 自动机. 这在后面的确定化算法中有详细的说明, 随着确定化的进行, 每一个转移 (即状态到字母到状态的有序三元组) 所拥有的信息量更加的丰富, 具有更强的表达能力, 因而将一个转移本身作为新的自动机的接受条件就变得更加自然与便捷. 这两者一定条件下可以互相等价转化, 但是不同自动机的状态数等规模可能会有指数级别的差距.

1.4 有序树

在有限字上的有限状态机的子集构造法, 通过状态的幂集即可表达有限状态机的所有可能状态. 这在扩展到无限字的情况时无法直接应用, 因为仅知道状态的幂集, 并不能反映出在无限序列中的状态, 比如对于状态的一个幂集, 有限状态机中表示运行序列到达过这些状态, 但是在无限字和无限序列中, 仅知道“到达过”这些状态是不够的, 还需要知道是否无限次的访问. 因为对有限的状态和无限的字来说, 依据抽屉原理, 一定是有部分状态是无限多次访问的. 如何记录这些被无限多次访问的状态, 将其和被有限次访问的状态区别开来, 这是对无限字自动机确定化研究所需要面对的首要问题. 通过对子集构造法思想的扩展, 将不同状态集合依照一定规则组合成树形数据结构成为一种有效的手段.

首先比较重要的研究成果是 Muller[4]的 Muller-Schupp 树, 核心是通过着色 (3 种颜色) 的方式记录不同的节

点, 优点是非常直观, 但是有大量冗余信息. Safra[12]为了缩减冗余节点, 引入了有序树的概念. 有序树是一种全新的数据结构, 基本思想是根节点的状态集为该树包含状态的超集, 每一个节点的所有子节点的状态集合的并集为该节点状态集合的真子集. 同时每一棵树的所有节点依照一定的命名规则对应有自己唯一的识别标识. 有序树简化了 Muller-Schupp 树中右边子冗余的部分, 只保留了 (会包含全部到达最终状态的) 左边部分, 因此可以大幅降低存储数据的空间开销. Safra 在有序树的基础上建立了 Safra-树, 其核心命名规则是依照有序树中的每个节点从根节点出发的生成顺序依次来命名. 基于 Safra 树的 Büchi 自动机确定化算法依然是当今主流的 Büchi 自动机确定化算法之一. Schewe 在 Safra 树的基础上对命名规则进行了改进, 建立了历史树. 历史树是一种数据结构, 反映了无限自动机在运行中的当前状态相关的性质, 主要体现在可以即使反应自动机所经历的循环中, 可能无限多次访问的状态节点和一次未曾访问的节点的性质. 历史树的结构为节点的集合, 分为根节点和子节点, 每个节点包含 3 种主要属性, 分别为标签 label, 命名 name, 标记 flag. 其中标签存储对应的 Büchi 自动机的状态集合, 反映了该节点所经历的所有状态. 命名为该节点依据历史树的唯一对应位置坐标, 根节点命名为 root, 其他节点命名依据有序树的规则, 例如 1.3.2 表示根节点的第 1 子节点的第 3 个子节点的第 2 个子节点. 所有树的每一个点都满足该要求则说明这棵树是紧凑的, 例如假如一个节点为根节点的第 2 个子节点, 但是根节点的第 1 个子节点被删除了, 那么这个节点将“向左”移动成为根节点的第 1 个子节点, 其命名将从 2 变为 1. 该操作称为节点的重命名, 对历史树所有节点进行重命名被称作历史树的序列化. 标记指当前节点的状态, 共有 3 种可能, 分别为无状态, 接受状态, 不稳定状态. 新生成的节点默认都是无状态, 后面两种状态在确定化算法中会随着树之间的转化与节点的增删而改变.

我们在 Schewe 的历史树基础上通过对节点分类, 将每个节点指向的索引进行冗余分析, 依据是否可以合并提出了新的命名规则, 建立了 *c*-历史树[18]. 具体地说, 通过原本历史树上的每一个节点都具有绝对的路径名, 例如上面的 1.3.2, 在一定条件下, 这个节点所决定的索引对的拒绝集, 是可以与另一个节点比如 1.4.1 所决定的索引对的拒绝集相交为空. 那么这两个索引就可以完成合并, 因为合并之后的新的接受集和拒绝集将于原来的两个索引对保持语义的一致. 这样就可以使这两个节点享用同一个名字, 这就是新的命名规则. 依据此命名规则得到的历史树就是 c-历史树, 依据 c-历史树将大幅降低 Büchi 自动机确定化为 Rabin 自动机的索引数.

1.5 基于有序树的 Büchi 自动机的确定化算法

对于 Büchi 自动机的确定化算法, 目前主流是基于有序树的确定化算法. 这可以近似看作是子集构造法的扩展, 将状态子集扩展为有序树. 本质上通过对有序树节点的命名来达到记忆是否无穷多次出现的目的. 其算法思想如下.

(1) 通过 Büchi 自动机的初始状态和命名规则构造初始有序树. 初始有序树即为确定化后的 Rabin 自动机初始状态.

(2) 依据 Büchi 自动机的转移关系从初始状态出发, 通过初始有序树繁衍生成全部有序树. 其中有序树的生成规则不依赖于命名规则的不同而不同, 生成的全部有序树将作为确定化后的 Rabin 自动机的全部状态集.

(3) 依据有序树之间的转移关系得到确定化后的 Rabin 自动机的转移关系, 同时按相应命名规则标记树上的节点为接受/拒绝节点, 并依此得到确定化后的基于转移的 Rabin 自动机的接受条件.

(4) (可选) 将基于转移的确定性 Rabin 自动机等价转换为基于状态的确定性 Rabin 自动机.

目前存在的 3 种命名规则: Safra-树命名规则, 历史树命名规则和 *c*-历史树命名规则分别对应了 Safra 算法, Schewe 算法和我们的算法.

1.6 完全自动机

一个完全自动机 $FA=\{\Sigma,Q,Q_0,\delta,\lambda\}$, 其字母表定义为 $\Sigma=Powerset(Q\times Q)$, 转移关系$\delta$ 定义为: 对所有 $p,q\in Q,a\in\Sigma,\langle p,a,q\rangle\in\delta$ iff $\langle p,q\rangle\in a$.

完全自动机是一种特殊的自动机[19], 其思想是将转移关系表扩充到最大化, 使得任意一个相同状态数, 相同字母表的自动机都能“嵌入”完全自动机其中, 使得同状态的所有自动机都将是他的子集. 为此, 其字母表的大小达到了 $2^{(|S|)^2}$, 由于其庞大的字母表, 在自动机理论研究证明中, 对完全自动机构造合适的字将简化证明复杂度.

完全自动机技术是一种用于证明自动机理论上下界的处理思路. 其核心思想将普通自动机的字母表扩充到最

大的可能性上, 得到最大的完全自动机, 在证明应用中, 我们只需要把精力放在构造巧妙的字, 来代替之前证明中需要构造的自动机家族, 这可以极大的简化有关的上下界证明, 甚至得出新的更紧上下界.

这里引用一个例子来说明完全自动机在相关理论证明的优越性.

定理 1. 对任意状态为 n, 字母表大小为 2 的自动机, 其补自动机的状态数下界为 2^n.

这一定理虽然是直观的, 但是其证明并不简单, 传统方法历经等多人的研究[20], 才通过构建无限自动机家族的形式得到证明. 然而使用完全自动机的辅助, 构造合适的“桥梁”字, 可以直接得到上面定理的证明. 证明过程简要如下.

证明: 事实上, 我们只需要能构造出一个满足要求的 n 状态的自动机, 证明其补自动机的状态数至少为 2^n, 即可完成证明, 而完全自动机就刚好可以满足要求. 记 $FA_n=(\Sigma_n,Q_n,I_n,\delta_n,\lambda_n)$ 为状态为 n 的完全自动机, 其中满足 $Q_n=I_n=\lambda_n=\{s_0,s_1,\ldots,s_{n-1}\}$. 对每个状态子集 $T\subseteq Q_n$, 设一个函数 $Id(T)$ 用于表示该子集中所有状态“自己到自己”的字的集合: $Id(T)=\{\langle q,q\rangle \mid q\in T\}$. 取任意两个字 u_T,v_T, 满足 $u_T\in Id(T),v_T\in Id(Q_n\backslash T)$. 则 u_Tv_T 不是 FA_n 的语言. 记 FA_n 的一个补集自动机为 CA, 那么 u_Tv_T 则是 CA 的语言, 也就是说, 存在一个 $\widehat{q}_T$ 使得 $\widehat{q}_I\xrightarrow{u_T}\widehat{q}_T\xrightarrow{v_T}\widehat{q}_F$. 考虑到 S 的子集 T 相应的个数为 2^n, 若能证明每一个 T 对应至少一个不同的 $\widehat{q}_T$, 那么即可证明. 考察 $T_1\neq T_2$, 假设对应的 $\widehat{q}_{T_1}=\widehat{q}_{T_2}$. 一定存在至少一个状态 s 满足 s 属于 $\{T_1\setminus T_2\}$, 因此 $s\xrightarrow{u_{T_1}}s\xrightarrow{v_{T_2}}s$, 则 $u_{T_1}\ v_{T_2}$ 是原自动机 FA_n 的语言. 另一方面, 又由于 $\widehat{q}_I\xrightarrow{u_{T_1}}\widehat{q}_{T_1}=\widehat{q}_{T_2}\xrightarrow{v_{T_2}}\widehat{q}_F$, 则 $u_{T_1}\ \ v_{T_2}$ 是原自动机 FA_n 的补集 CA 的语言, 得到矛盾. 所以可以得出 FA_n 的补集至少要 2^n 个状态数.

完全自动机理论的证明关键是构造合适的完全自动机, 将验证的问题延迟到了转移分析的问题上, 从而由传统的构造合适的自动机族优化为构造合适的字, 这往往更加直观和方便.

2 工具实现

2.1 工具架构

工具分为数据处理、确定化算法、分析验证这 3 个模块. 具体架构如图 1 所示.

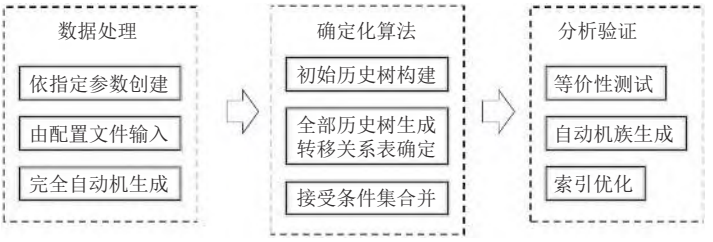

图 1 NB2DR 工具框架

使用 Java 编写 (JDK 版本为 18.0.1.1), 代码总计非空行约 3 万行. 数据处理模块将自动机格式标准化, 兼容了其他工具如 GOAL 的自动机格式, 同时包含了无限字的创建, 历史树的建模, 以及各种自动机的接受条件判定通式和完全自动机生成. 确定化算法模块整合了索引数最优的 Safra 算法, 状态数最优的 Schewe 算法[13], 同时还实现了基于新的命名规则, 剪掉了冗余的部分历史树分叉, 在状态数与 Schewe 持平条件下达到更小索引的我们的算法[18]. 分析验证模块面向确定化研究的实际问题, 针对性的实现研究需要的常用功能, 包括等价性测试, 索引包含分析, 自动机族生成等部分. 等价性测试采用近似的穷举无限字的算法, 可以对输入的任意两个自动机进行等价性测试. 自动机生成与分析模块可以按照输入要求生成指定类型的自动机家族, 相当于新建满足一定条件的自动机数据集, 同时对其的确定化可以分析其运行状态和需要的诸如状态数, 索引数, pair 集合判断合并情况等运行细节.

2.2 数据处理

该模块主要定义了工具运行所需要的数据结构, 包括各种无限字自动机结构. 输入部分可以选择自行构建任

意结构的自动机, 也支持以 GOAL 程序的配置文件标准导入相应的自动机[8]. 对于完全自动机可以支持对任意状态数, 任意接受条件的生成. 同时输出部分引入 Graphviz 插件[21], 将生成的自动机转移关系输出成该插件支持的 DOT 语言, 方便将简单的自动机生成可视化图像. 该插件是一款常见的开源可视化工具, 基于多种布局决策算法, 可以根据 DOT 文件生成较为直观的有向图. 重要的数据结构如下.

● ω 自动机结构. ω 自动机数据结构主要分为状态节点集合, 字母表集合, 转移关系集合, 接受条件类型部分. 对于 n 状态的自动机, 节点标签默认为 0 到 n–1. 对于字母表大小为 m 的自动机, 字母默认顺序为从 0 到 m–1. 转移关系使用继承自二维数组的类来存放, 第 1 维度为起始状态, 第 2 维度为字母, 值为该起始状态经由该字母所到达的终末状态集合. 如果为空则设为空集, 如果每一个终末集合均为单元素集合, 那么该自动机就是确定的.

● 完全自动机结构. 完全自动机技术是一种用于证明自动机理论上下界的处理思路. 其核心思想将普通自动机的字母表扩充到最大的可能性上, 得到最大的完全自动机, 使得同状态的所有自动机都将是他的子集. 由于其庞大的字母表, 在证明应用中, 我们只需要把精力放在构造巧妙的字, 来代替之前证明中需要构造的自动机家族, 这可以极大的简化有关的上下界证明, 甚至得出新的更紧上下界. 现有工具并没有相关的完全自动机实现, 本工具可以实现任意指定状态数 n 的完全自动机生成, 并对较小的 n 进行可视化展示.

2.3 确定化算法

该模块为工具核心部分, 整合了索引数最优的 Safra 算法, 状态数最优的 Schewe 算法, 以及在状态数与 Schewe 持平条件下达到很小索引的我们的算法. 这三者算法均是基于有序树的 Büchi 自动机确定化算法, 区别主要在命名规则的不同上.

有序树作为确定化算法的核心, 负责存储节点数据信息. 本工具采用离散的哈希表存储有序树, 其键为树中节点的绝对坐标, 依照 Schewe 算法的历史树坐标命名记录. 值为树中的节点. 其信息包含节点标记, 节点标签, 节点属性, 节点状态. 节点标记为标记其对应命名规则的参数, Safra 为 1, Schewe 为 2, 我们的算法为 3. 节点标签记录该节点在对应命名规则下的命名. 节点属性记录该节点代表的自动机状态集合, 由于状态都是由数字从 1–n 依次表示, 该属性由 vector 类存储. 节点状态用于标记节点当前是否处于接受/不稳定状态.

对应命名规则主要有 3 种. Safra 的命名规则核心是依据有序树上节点的生成顺序, 确定化后其规模上界为 $2^{O(n\log n)}$, 得到的 Rabin pair 索引数目上界为 n. Schewe 命名规则依据有序树的节点绝对坐标, 确定化后其规模上界为 $(1.65n)^n$, 得到的 Rabin pair 索引数目上界为 2^n. 我们的命名规则依据剪掉冗余历史树分叉节点, 依据可达性缩减历史树节点的名称数目, 得到新的 c-历史树, 确定化后其规模上界为 $(1.65n)^n$, 得到的 Rabin pair 索引数目上界为 $2^{\frac{n}{2}}$.

这里首先以 Schewe 算法为例, 演示确定化算法. 输入为标准的非确定性 Büchi 状态自动机 $B=\{\Sigma,Q,Q_0,\delta,F\}$ (如图 2), 输出为确定性 Rabin 转移自动机, $R=\{\Sigma,Q_r,Q,\delta_r,\lambda_r\}$.

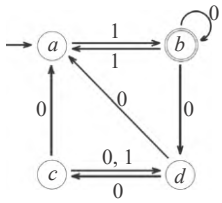

图 2 Büchi 自动机 B 实例

(1) 首先根据自动机的初始状态生成初始历史树 h_0. 具体为, 取其根节点的标签为 B 的初始状态. 将该历史树 h_0 加入输出的 Rabin 自动机的状态集, 并将其设置为该状态机的初始状态.

(2) 由 h_0 出发开始进行历史树遍历. 这一步将完成生成全部可能历史树的目的. 由于历史树的性质, 其根节点标签最大为 B 的状态数 n, 而节点具有“任意节点 t 的全部子节点的标签并集为 t 的标签真子集”这一特点, 每一层都会减少, 因此历史树最多有 n 层 (不会是无限树), 因此历史树的总量是一定的. 也就是说, 可以通过遍历搜索的

思路得到全部的历史树可能. 换言之, 这些历史树的集合将作为确定性 Rabin 自动机的所有状态集, 而这些历史树的最大值即为相应确定性 Rabin 自动机的状态数上界. 下面是具体的遍历算法.

1) 数据预处理. 建立历 Rabin 自动机状态集 Q_r, 亦即历史树集合, 将 h_0 添加进入该集合, $Q_r=\{h_0\}$. 建立 Rabin 自动机初始状态集合 $Q_{r0}=\{h_0\}$, 建立 Rabin 自动机的转移关系集合 δ, 建立 Rabin 自动机的接受条件集合 $\lambda=\{\langle G_1,B_1\rangle,\langle G_2,B_2\rangle,\ldots,\langle G_k,B_k\rangle\}$

建立当前待处理历史树队列 qh 并将 h_0 压入该队列, 即 qh={ h_0 }, 队列满足先进先出的原则.

2) 弹出 qh 中的元素 h, 即得到当前待处理历史树 h. 建立当前待处理字母队列 qa 并将字母表中的字母全部压入该队列.

3) 弹出 qa 中的元素 a, 即得到当前待处理字母 a, 依照历史树转移算法得到 h 经由 a 的后继历史树 h', 即 (h,a,h'). 具体历史树转移算法如下:

① 对历史树 h 进行全节点扩充. 首先预处理需要将 h 上的全部节点的标记改为无状态. 对于 h 树上的每一个节点 t, 添加一个该节点的最右子节点 t'. 并且 t' 标签为 t 节点与接受状态 λ 的交际, t' 的命名依照之前历史树的节点命名规则不变, t' 的标记统一为无状态.

② 对①中得到树从上到下每一层进行相同性检验. 对于每一层的节点来说, 建立该层的已有状态队列 qse. 从最左节点开始往右逐个扫描其标签, 判断其标签中的每一个状态是否在 qse 中. 如果不在, 则将该状态加入 qse 中; 如果在则删除该标签中的该状态. 重复此过程直至该层从左到右所有节点均被进行相同性检验.

③ 对②中得到的树的每一个节点进行包含性检验. 建立这一步的包容性检验队列 qc, 首先将根节点 root 压入该队列 qc. 从 qc 读取当前节点 t, 判断该节点 t 的标签状态集是否与 t 的所有子节点的标签状态集的并集相同, 如果是, 则删除 t 的全部子节点, 然后将 t 的标签改为接受状态; 如果否, 则将 t 的全部子节点均压入 qc 中. 重复此操作直至 qc 中没有节点为止.

④ 对③中得到的树进行删除空节点. 这一步就是遍历全部树中的节点, 将所有空节点直接删除.

⑤ 对④中得到的树进行序列化. 建立这一步的序列化队列 qo, 首先将根节点 root 压入该队列 qo. 从 qo 读取当前节点 t, 首先判断该节点是否标记为不稳定节点. 如果是, 则将其子节点全部修改标签为不稳定节点, 并压入 qo 中. 如果否, 则继续判断该节点的命名是否正确 (例如 3.3.2 节点实际位置应该是 2.3.2), 如果正确, 则将该节点的所有子节点压入 qo 中; 如果不正确, 则将其重命名为正确的名字, 并将节点的标签改为不稳定节点. 从 qo 中弹出该节点并反复操作直至 qo 中再无节点. 至此得到的树即是 h 经字 a 得到的后继历史树 h'.

将得到的转移 (h,a,h') 添加进入 Rabin 自动机转移关系中. 如果 h' 中存在不稳定节点, 则将该转移添加进入该节点名字 i 对应的 $\lambda_r\langle G_i,B_i\rangle$ 的 B_i 中. 如果 h' 中存在接受节点, 则将该转移添加进入该节点名字 i 对应的 $\lambda_r\langle G_i,B_i\rangle$ 的 G_i 中.

4) 判断 h' 是否存在于 Q_r 中, 如果不存在, 则将 h' 添加进入 Q_r 中, 并将 h' 压入 qh 队列. 重复步骤 3) 直到 qa 为空, 即全部字母表中的字母均经历过历史树 h' 的遍历.

5) 重复步骤 2) 直到 qh 队列为空, 此时完成全部历史树生成, 同时也相应完成了全部转移关系和接受条件的生成. 如图 3 所示.

(3) 整合上一步得到的确定性 Rabin 转移自动机, 检查其接受条件 λ_r, 将其中的空 *pair* 删除, 得到最终接受条件.

接受条件为 $\lambda=\{(\{h_0[1]h_2,h_4[0]h_6,h_9[0]h_{10},h_{10}[0]h_6\},\{\varnothing\}),(\{h_5[1]h_3,h_9[1]h_{12},h_{11}[1]h_{13}\},\{\varnothing\}),(\{h_7[1]h_{10}\},\{\varnothing\}),(\{h_{10}[1]h_7,h_{13}[1]h_{11}\},\{h_{11}[0]h_4\}),(\{h_{12}[1]h_9\},\{h_{11}[0]h_4\})\}$.

如图 4 所示为最终确定化得到的 Rabin 自动机.

值得注意的是, 之前得到的是基于转移的确定性 Rabin 自动机, 状态数大小为 $O((1.65n)^n)$. 可以继续利用相应算法将其转化为基于状态的确定性 Rabin 自动机. 目前主要有两种方法, 分别适合于不同大小字母表时的情况.

第 1 种较简单的思路是根据每一个转移和字将状态数扩充. 由于转移是 (h,a,h') 对, 将每一个后继的 h'和其经历的字 a 可以看作一个静态的整体进行绑定当作新的确定的基于状态的 Rabin 自动机的状态, 这样得到的 Rabin

自动机状态数为 $l\cdot O((1.65n)^n)$, 其中 n 为基于转移的 Rabin 自动机状态数, l 为基于转移的 Rabin 自动机的字母表大小. 这样做虽然增大了确定化的自动机的状态数, 不过可以得到更加直观的基于状态的 Rabin 自动机. 该思路在字母表比较小的时候很容易达成, 但是由于 n 个状态的自动机的字母表上限高达 2^{n^2} 个, 所以在具有较大字母表时, 会对最终确定化的状态数有指数级的影响.

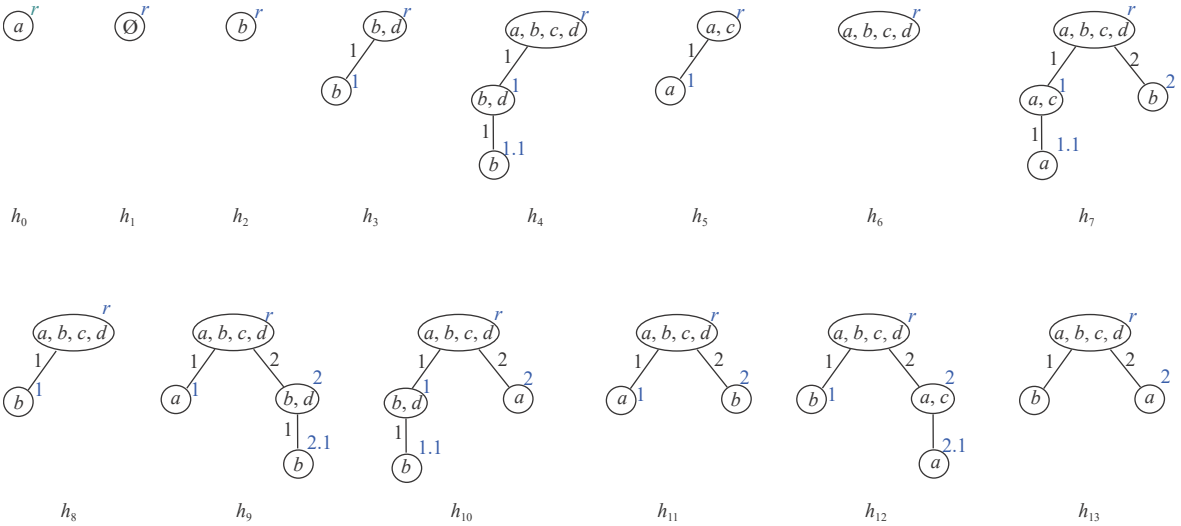

图 3 确定化为 Rabin 自动机算法中产生得到全部历史树

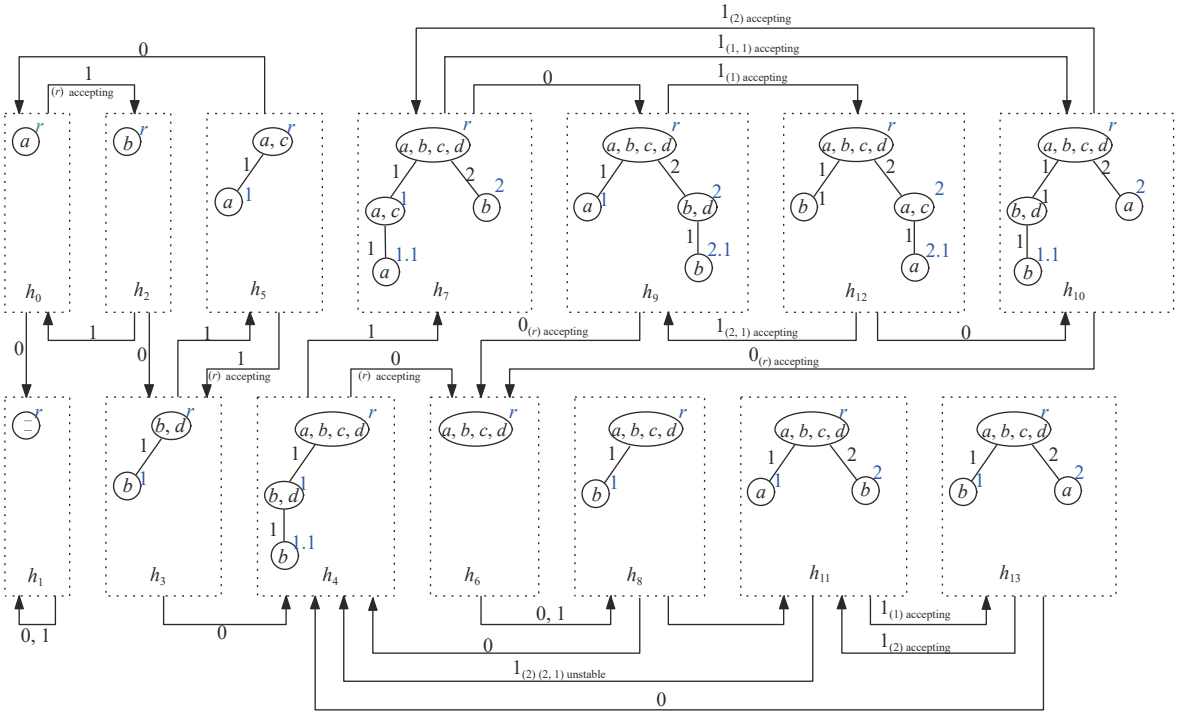

图 4 生成的 Rabin 自动机 R 实例

这时, 可以采用第 2 种策略, 也就是从历史树本身入手对转移关系进行改进. 转移关系的本质是在 (Rabin) 接受条件中, pair 的选择决定于两棵树之间转移关系中出现的接受节点和不稳定节点. 那么很自然的思路就是可以

对历史树依据接受节点和不稳定节点进行扩展. 首先对于每一个历史树 T, 记其节点数为 m, 那么每一个节点有接受节点、不稳定节点和其他节点 3 种可能, 也就是 3^n 个不同的历史树, 接下来再根据历史树的性质将其中大多数不合理的树删除. 首先接受节点和不稳定节点只能是叶子节点, 同时若一个节点为接受节点, 那么其左侧兄弟节点和父节点均不可能是不稳定节点. 其次对于不稳定节点来说, 若一个节点为不稳定节点, 那么其右侧的兄弟节点和全部子节点一定是不稳定节点. 基于此得到的历史树族就是该历史树 T 经由添加节点接受信息后的扩展历史树 (enriched history tree), 将全部历史树的扩展历史树算出即可得到新的基于状态的确定性 Rabin 自动机的状态节点, 同时接受条件变为由对应树节点的该节点为接受节点的扩展历史树集合和该节点为不稳定节点的扩展历史树集合构成的对的集合. 该全部扩展历史树的数目可以通过递归估算为 $\mathrm{O}((2.66n)^n)$ 个, 即相应确定性 Rabin 自动机状态复杂度为 $\mathrm{O}((2.66n)^n)$.

2.4 分析验证

这一模块可以辅助自动机领域的部分研究. 包括等价性测试, 索引包含分析, 自动机家族生成, 完全自动机生成等部分.

● 等价性测试. 对于任意无限字的有限状态自动机来说, 想直接得到其接受语言集是比较困难的, 甚至是不可能在有限空间内表示清楚的. 因此, 想直接验证两个自动机的语言是否等价往往需要依赖于自动机本身的特点. 通常, 如果能直观看出自动机的语言集, 尤其是在自动机本身状态数较小时还是相对容易的. 部分工具的等价性验证需要将自动机重新转回时序逻辑语言, 利用求解器进行辅助. 但随着自动机的状态数增多, 接受条件的索引也增多, 尤其是两个自动机种类不同时, 接受条件也更加的复杂, 这时候等价性验证将变得非常困难. 本工具在这个问题上进行了一定尝试, 具体的思路就是生成一系列的无限字, 然后观察其在待验证自动机的运行状态, 是否满足相对应的接受条件. 只要找到一个反例即可判定两个自动机是不等价的, 如果在足够多的无限字上都能满足接受, 那么就可以判断在一定的置信区间上这两个自动机是等价的.

具体来说, 首先是生成一个无限字. 这个无限字默认采用循环无限字的结构, 根据字母表随机产生其前缀字和循环节字, 然后对于 ω 来说, 实际采用 w 次, 通常设置 w 为对应自动机的状态数的 mn 倍. 接下来让两个自动机分别运行这个字, 判断其接受条件, 对于需要无限次访问的参数判断是否达到 m 次, 若达到则判定其结论为接受. 如果两个自动机的接受结论不一致, 则可以判定两个自动机的语言不一致. 如果结论一致, 则继续重复这个过程, 重复 m 次, 如果 m 次两个自动机的语言都一致, 那么得到结论两个自动机的语言是一致的.

● 自动机族生成. 同类自动机家族生成模块可以方便我们生成指定条件的自动机族. 对于给定不大的状态数为 n 且字母表大小为 k 的 Büchi 自动机, 本工具支持生成符合条件的随机自动机 (单个) 以及在内存设置允许的情况下进行枚举所有自动机 (全部). 依照的算法是根据状态数 n 和字母表大小 k 确定可能的转移关系规模, 得到 k 个由 n 到 2^n 的映射作为 Büchi 自动机的转移关系, 且对应的全部 nk 个幂集按照空集, 单状态集, 双状态集… 直到含所有状态的集其全部子集的顺序生成, 即共有 $(2^n)^{nk}$ 种可能. 该枚举操作做了多线程优化, 可以一边生成自动机一边对其进行其他如确定化的操作. 同时进行了内存管理, 若该次操作生成的自动机大小规模超过了指定内存大小, 则会终止运行.

生成的自动机族可以实时进行按需要筛选保存. 例如可以生成全部状态数为 3, 字母表数为 3 的 Büchi 自动机家族, 满足其确定化之后得到的 Rabin 自动机索引数为 4. 这为分析 Büchi 自动机等价转化为确定性 Rabin 自动机时的运行流程和其确定化规律提供了直观的大数据用例. 同时还可以筛选出包含有任意可指定的相同子转移关系的 Büchi 自动机, 换言之这些自动机会包含同样的转移关系部分. 这些自动机相当于是在同样的原初自动机基础上添加了额外的若干转移关系得到的, 在对这些 Büchi 自动机族做确定化之后, 就可以看出对于确定的部分转移关系保持不变的情况下, 新添加的这些转移关系会对确定化后的 Rabin 自动机产生怎样的影响.

● 索引分析. 该分析可以对于 Rabin 自动机的接受条件进行直观的展示分析, 往往我们通过 Schewe 算法得到的 Rabin 转移自动机拥有大量的空索引和重复索引, 甚至有时会出现矛盾索引, 该模块可以直观指出不同索引之间的相互依赖关系. 遗憾的是目前还无法直接优化确定化后的 Rabin 转移自动机使其达到最精简的地步, 只能起到相应的辅助的作用. 要想得到最优的确定性 Rabin 自动机还需要人工根据该模块进行识别其接受状态, 再做手动筛选除去空索引和重复索引, 依照矛盾索引对无冲突的索引进行并集操作合并简化优化. 通过对索引的包含分

析, 我们可以观察确定性的自动机的索引分布, 探究其不同合并可能的内在规律. 例如上一节中的例子中, 可以看到接受条件为:

$$\lambda=\{(\{h_0[1]h_2,h_4[0]h_6,h_9[0]h_{10},h_{10}[0]h_6\},\{\varnothing\}),(\{h_5[1]h_3,h_9[1]h_{12},h_{11}[1]h_{13}\},\{\varnothing\}),$$
$$(\{h_7[1]h_{10}\},\{\varnothing\}),(\{h_{10}[1]h_7,h_{13}[1]h_{11}\},\{h_{11}[0]h_4\}),(\{h_{12}[1]h_9\},\{h_{11}[0]h_4\})\}.$$

这里前 3 个 Rabin pair 由于拒绝集均为空集, 因此可以将这三者合并到一个 pair 中, 起到减少索引数的作用. Tian 等人在 Büchi 自动机的相关削减索引理论方面取得了部分成果, 通过对冗余历史树的节点合并, 重新为每个节点分配新的名字, 得到新的命名方式, 并基于此得到新的 Büchi 自动机确定化算法[18], 我们工具也实现了这一算法, 方便验证得到更小的确定性 Rabin 自动机.

3 工具展示

3.1 确定化算法展示

本节通过两个 Büchi 自动机实例的确定化来展示工具的确定化效果, 这两个自动机分别是任意自动机和完全自动机.

Büchi 自动机 B1 如图 5 所示, 将其按照状态数, 字母表数, 以及状态转移关系和接受状态集合作为参数输入. 运行 Büchi 自动机确定化 (历史树算法), 可以直接得到确定化的基于转移的 Rabin 自动机. 生成的 Büchi 自动机如图 6 所示.

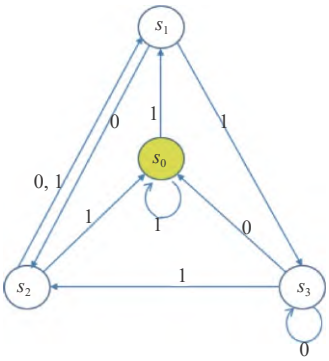

图 5 输入的 Büchi 自动机 B1 实例

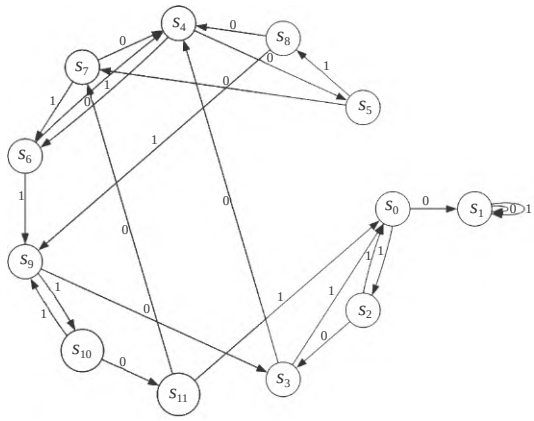

图 6 生成的 Rabin 自动机实例

完全自动机因为比较字母表非常大, 这里只取 n=2 时的完全自动机采用工具生成, 如图 7 所示, 其中初始状态设置为 s_0, 接收状态设置为 s_1. 运行 Büchi 自动机确定化 (历史树算法), 可以直接得到确定化的基于转移的 Rabin 自动机. 生成的 Büchi 自动机如后文图 8 所示.

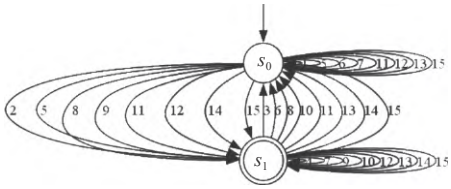

图 7 输入的完全 Büchi 自动机实例

3.2 自动机族生成

对于指定输入 Rabin pair 数目为 2, 目标 Büchi 自动机的状态数为 3 时, 可以输出部分满足该条件的 Büchi 自动机, 如图 9 所示.

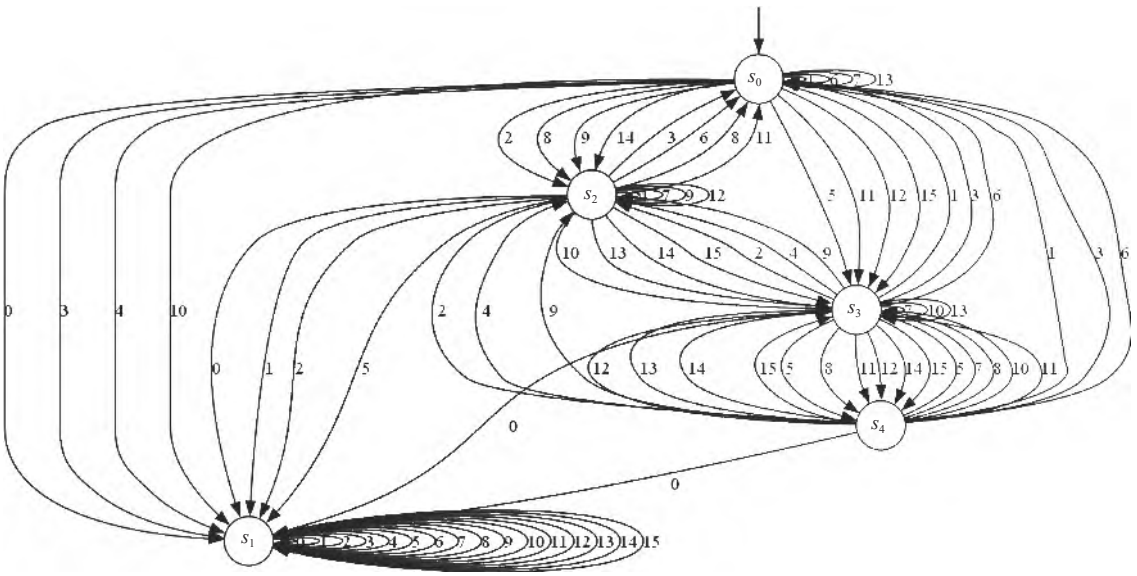

图 8 输出的完全 Rabin 自动机实例

	0	1
0	[2]	[0, 1]
1	[1]	[]
2	[]	[]

	1	0
0	[0, 1]	[1, 2]
1	[]	[1]
2	[]	[]

	0	1
0	[2]	[0, 1, 2]
1	[1]	[]
2	[]	[]

	0	1
0	[0, 1]	[1, 2]
1	[1]	[]
2	[]	[]

	0	1
0	[1, 2]	[0, 1, 2]
1	[]	[1]
2	[]	[]

	0	1
0	[0, 1]	[1, 2]
1	[1, 2]	[]
2	[]	[]

	0	1
0	[0, 1, 2]	[1, 2]
1	[1, 2]	[]
2	[]	[]

	0	1
0	[1, 2]	[0, 1, 2]
1	[1]	[] |
2	[]	[]

图 9 n=3 时产生的 Büchi 自动机族

3.3 实验与分析

这部分我们考虑得到的确定性 Rabin 自动机的索引数目的优化问题. 我们采用之前工具的自动机族生成功能产生需要的数据集. 考察不同状态数和字母表大小对索引的影响. 其中数据集选择为: 指定状态数由 3–8, 字母表大小分别设置为 2, 5, 10, 随机生成 200 个符合条件的 Büchi 自动机. 首先并使用工具对其确定化, 并进行等价性测试. 再依算法[18]对其索引数进行优化. 全部每组的初始 Büchi 自动机, 确定化后的 Rabin 自动机, 索引优化后的 Rabin 自动机三者均通过了等价性测试. 其中索引优化的具体结果如表 1 所示.

NBW 和 DRW 分别表示非确定 Büchi 自动机和确定化后得到的确定性 Rabin 自动机, #状态数和#字母表分别表示 Büchi 自动机的状态数目和字母表大小, 每组数据均产生了 200 个相应条件的自动机. #索引数为确定化后得到的全部 Rabin 自动机索引数目的平均值. 可以从表 1 中看到, 字母表和状态数对索引数目都有影响, 其增大都

会导致索引数目的增大. 我们的优化算法确实有一定效果, 尤其在字母表增大的情况下比较明显.

本节展示不同的 3 种 Büchi 自动机确定化算法的效果与工具的可用性. 这里数据集全部由之前工具的自动机族生成功能产生我们需要的数据集, 非确定性 Büchi 自动机族. 这里数据集选择为状态数指定从 3–8, 每个状态对 3 种字母表大小, 分别为 2, 5, 10, 每一种参数随机生成 200 个符合条件的非确定性 Büchi 自动机. 对每个自动机分别采用 3 种确定化算法得到确定 Rabin 自动机. 实验结果见表 1.

表 1　3 种 Büchi 自动机算法确定化效果比较

NBW		DRW #Safra		DRW #Schewe		DRW #our		#eq
#states	#alpha	#states	#Rabin pairs	#states	#Rabin pairs	#states	#Rabin pairs	
3	2	18	1.150	9.1	1.240	9.1	1.240	T
3	5	20	1.730	10.2	1.900	10.2	1.900	T
3	10	21	2.530	12.2	2.470	12.2	2.350	T
4	2	75	2.160	24.3	1.600	24.3	1.600	T
4	5	80	2.735	25.1	2.960	25.1	2.955	T
4	10	123	3.450	25.9	4.030	25.9	3.880	T
5	2	320	2.365	110	2.020	110	2.010	T
5	5	439	3.670	120	4.020	120	3.895	T
5	10	460	3.970	138	6.180	108	5.445	T
6	2	1 513	3.530	552	1.995	552	1.995	T
6	5	1 783	4.150	593	4.445	593	4.435	T
6	10	2 319	4.815	609	9.100	609	7.650	T
7	2	6 739	4.174	1 208	2.140	1 208	2.140	T
7	5	7 809	4.530	1 243	4.920	1 243	4.915	T
7	10	8 630	5.130	1 381	9.030	1 381	8.650	T
8	2	13 748	4.375	1893	1.990	1893	1.990	T
8	5	15 416	4.920	2 173	4.520	2 173	4.510	T
8	10	17 312	5.740	2 453	9.700	2 453	8.670	T

NBW 和 DRW 分别表示非确定的 Büchi 自动机和确定的 Rabin 自动机. #Safra, #Schewe, #our 分别表示 Safra 的基于 Safra 树的确定化算法, Schewe 的基于历史树的确定化算法和我们的基于 c-历史树的确定化算法. #state #alpha #Rabin pairs 分别表示状态数, 字母表大小和 Rabin 自动机接受条件的索引数. #eq 表示等价性验证. 从表 1 中数据可以看出, Safra 的算法相比于 Schewe 和我们的算法, 虽然索引较小, 但是状态数很大. 我们的算法和 Schewe 相比在相同状态数下有较小的索引数. 不过这在 Büchi 自动机状态数较小和字母表较少时并不明显, 这是因为我们的算法的命名规则需要在状态数很大, 迁移数较多时才能有明显的缩减效果.

为此, 我们针对性的设计数据集为较大迁移数的 Büchi 自动机族 (接近完全自动机). 选择状态数从 3–7 的非确定 Büchi 自动机, 每一个状态值对应迁移数分别为 11, 24, 40, 60, 84, 字母表大小分别为 4, 6, 8, 10, 12. 其确定化结果见表 2.

表 2　复杂情况下我们与 Schewe 算法对比

NBW			DRW					
					#Schewe		#our	
#states	#trans	#alpha	#states	#trans	#Rabin pair	#time (s)	#Rabin pair	#time (s)
3	11	4	9	36	4	0.01	3	0.01
4	24	6	30	180	7	0.05	4	0.05
5	40	8	228	1824	15	0.26	6	0.29
6	60	10	2 316	23 160	31	34.65	11	35.25
7	84	12	—	—	—	T.O.	—	T.O.

#trans 为自动机的转移数目, #time 为运行时间 (s), T.O. 为运行超时 (超过 1 h). 可以看到在状态数为 7 时对应 Büchi 自动机的确定化已经超时, 在状态数为 5 和 6 时, 我们的算法得到的索引数为 6 和 11, 大幅小于 Schewe

的算法对应的 15 和 31.

4 相关工作

Althoff 等人[15]开发的 OmegaDet 工具对 Büchi 自动机的确定化算法进行了研究, 他们主要对比采用了 Safra 和 Muller-Schupp 的确定化算法, 该工具虽然局限于发表时间较早只有两种并不最优的确定化算法进行实验, 但是 OmegaDet 通过直观对比 MullerSchupp 树和 Safra 树的异同, 展示了无限字自动机确定化领域的发展思想, 即通过删除冗余节点达到简化系统的目的. 这对我们的工具的研究是一种很重要的启发.

Duret-Lutz 等人[16]开发的 SPOT 工具主要面向与模型检测实际需求中的 LTL 到基于转移的 Generalized Büchi 自动机的转换, 它更偏向于作为模型检测器与数据类型算法结构等的桥梁, 而不是在于自动机确定化方面的研究.

GOAL[17]是一款比较成熟的无限字自动机与逻辑的研究工具, 支持多种自动机的生成与确定化算法, 以及求补等操作. 正因为 GOAL 设计的比较全面而且代码比较早, 其自动机的底层数据结构比较复杂, 一定程度上增大了运行大规模自动机, 尤其是运行完全自动机时的时间与空间开销, 更适用于面向教学演示等规模较小的自动机应用场景. 相比之下我们的工具可以精准的解决面临的问题, 更好的发挥服务器潜能, 铺垫后续的研究, 这也是我们团队实现本工具的初衷.

5 总结与展望

无限字自动机的确定化一直是理论计算机领域, 特别是形式化验证与时序逻辑领域重要的组成部分, 随着如今计算机性能的高速发展, 使用适当工具对复杂自动机进行确定化是研究其内在确定化逻辑的重要方法. 我们实现了确定化研究工具 NB2DR. 可以对非确定性 Büchi 自动机进行高效准确的确定化, 并通过工具提供的分析其确定化过程来达到对其确定化算法改进的目的.

未来将进一步提高工具的普适性, 加入对于 Emerson-lei 等自动机的支持, 并对无限字自动机的索引理论上界进行深入研究.

References:

[1] Büchi JR. On a decision method in restricted second order arithmetic. In: Proc. of the 1962 Int'l Congress on Logic, Methodology and Philosophy of Science. Stanford: Stanford University Press, 1962. 1–12.

[2] Rabin MO. Decidability of second-order theories and automata on infinite trees. Trans. of the American Mathematical Society, 1969, 141: 1–35. [doi: 10.2307/1995086]

[3] Boker U, Kupferman O. Translating to co-Büchi made tight, unified, and useful. ACM Trans. on Computational Logic, 2012, 13(4): 29. [doi: 10.1145/2362355.2362357]

[4] Muller DE. Infinite sequences and finite machines. In: Proc. of the 4th Annual Symp. on Switching Circuit Theory and Logical Design. Chicago: IEEE, 1963. 3–16. [doi: 10.1109/SWCT.1963.8]

[5] Vardi MY. The Büchi complementation saga. In: Thomas W, Weil P, eds. Proc. of the 24th Annual Symp. on Theoretical Aspects of Computer Science. Aachen: Springer, 2007. 12–22. [doi: 10.1007/978-3-540-70918-3_2]

[6] Tsai MH, Fogarty S, Vardi MY, Tsay YK. State of Büchi complementation. Logical Methods in Computer Science, 2014, 10(4): 1–27. [doi: 10.2168/LMCS-10(4:13)2014]

[7] Safra S. Exponential determinization for Ω-automata with strong-fairness acceptance condition (extended abstract). In: Proc. of the 24th Annual ACM Symp. on Theory of Computing. Victoria: ACM, 1992. 275–282. [doi: 10.1145/129712.129739]

[8] Meyer PJ, Sickert S, Luttenberger M. Strix: Explicit reactive synthesis strikes back! In: Proc. of the 30th Int'l Conf. on Computer Aided Verification. Oxford: Springer, 2018. 578–586. [doi: 10.1007/978-3-319-96145-3_31]

[9] Streett RS. Propositional dynamic logic of looping and converse. In: Proc. of the 13th Annual ACM Symp. on Theory of Computing. Milwaukee: ACM, 1981. 375–383. [doi: 10.1145/800076.802492]

[10] Mostowski AW. Regular expressions for infinite trees and a standard form of automata. In: Proc. of the 5th Symp. on Computation Theory. Zaborow: Springer, 1984. 157–168. [doi: 10.1007/3-540-16066-3_15]

[11] Piterman N. From nondeterministic Büchi and Streett automata to deterministic parity automata. Logical Methods in Computer Science, 2007, 3(3): 1–21. [doi: 10.2168/LMCS-3(3:5)2007]

[12] Safra S. On the complexity of ω-automata. In: Proc. of the 29th Annual Symp. on Foundations of Computer Science. White Plains: IEEE, 1988. 319–327. [doi: 10.1109/SFCS.1988.21948]

[13] Schewe S. Tighter bounds for the determinisation of Büchi automata. In: Proc. of the 12th Int'l Conf. on Foundations of Software Science and Computational Structures. York: Springer, 2009. 167–181. [doi: 10.1007/978-3-642-00596-1_13]

[14] Tian C, Wang WS, Duan ZH. Making streett determinization tight. In: Proc. of the 35th Annual ACM/IEEE Symp. on Logic in Computer Science. Saarbrücken: ACM, 2020. 859–872. [doi: 10.1145/3373718.3394757]

[15] Althoff CS, Thomas W, Wallmeier N. Observations on determinization of Büchi automata. Theoretical Computer Science, 2006, 363(2): 224–233. [doi: 10.1016/j.tcs.2006.07.026]

[16] Duret-Lutz A, Poitrenaud D. SPOT: An extensible model checking library using transition-based generalized Büchi automata. In: Proc. of the 12th IEEE Computer Society's Annual Int'l Symp. on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems. Volendam: IEEE, 2004. 76–83. [doi: 10.1109/MASCOT.2004.1348184]

[17] Tsay YK, Chen YF, Tsai MH, Wu KN, Chan WC. GOAL: A graphical tool for manipulating Büchi automata and temporal formulae. In: Proc. of the 13th Int'l Conf. on Tools and Algorithms for the Construction and Analysis of Systems. Braga: Springer, 2007. 466–471. [doi: 10.1007/978-3-540-71209-1_35]

[18] Tian C, Duan ZH. Büchi determinization made tighter. arXiv:1404.1436, 2014.

[19] Yan QQ. Lower bounds for complementation of ω-automata via the full automata technique. In: Bugliesi M, Preneel B, Sassone V, Wegener I, eds. Proc. of the 33rd Int'l Colloquium Automata, Languages and Programming. Venice: Springer, 2006. 589–600. [doi: 10.1007/11787006_50]

[20] Jirásková G. State complexity of some operations on binary regular languages. Theoretical Computer Science, 2005, 330(2): 287–298. [doi: 10.1016/j.tcs.2004.04.011]

[21] Ellson J, Gansner E, Koutsofios L, North SC, Woodhull G. Graphviz—Open source graph drawing tools. In: Mutzel P, Jünger M, Leipert S, eds. Proc. of the 9th Int'l Symp. on Graph Drawing. Vienna: Springer, 2002. 483–484. [doi: 10.1007/3-540-45848-4_57]

马润哲(1990－), 男, 博士生, CCF 学生会员, 主要研究领域为自动机理论, 形式化方法, 时序逻辑.

王文胜(1993－), 男, 博士, CCF 专业会员, 主要研究领域为自动机理论, 时序逻辑, 神经网络可信性验证.

田聪(1981－), 女, 博士, 教授, 博士生导师, CCF 杰出会员, 主要研究领域为软件安全, 智能软件开发方法, 可信软件基础理论与方法.

段振华(1948－), 男, 博士, 教授, 博士生导师, CCF 会士, 主要研究领域为形式化方法, 可信软件基础理论与方法.