软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2023,34(8):3659−3673 [doi: 10.13328/j.cnki.jos.006614]

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

Streett 自动机确定化工具*

王文胜[1,2], 田 聪[1,2], 段振华[1,2]

[1](西安电子科技大学 计算理论与技术研究所, 陕西 西安 710071)

[2](综合业务网理论及关键技术国家重点实验室 (西安电子科技大学), 陕西 西安 710071)

通信作者: 田聪, E-mail: ctian@mail.xidian.edu.cn

摘 要: 自动机的确定化是将非确定性自动机转换为接收相同语言的确定性自动机, 是自动机理论的基本问题之一. ω 自动机的确定化是诸多逻辑, 如 SnS, CTL*, μ 演算等, 判定过程的基础, 同时也是解决无限博弈求解问题的关键, 因此对 ω 自动机确定化的研究具有重要意义. 主要关注一类 ω 自动机——Streett 自动机的确定化. 非确定性 Streett 自动机可以转换为等价的确定性 Rabin 或 Parity 自动机, 在前期工作中已经分别得到了状态复杂度最优以及渐进最优算法, 为了验证提出的算法的实际效果, 也为了形象地展示确定化过程, 开发一款支持 Streett 自动机确定化的工具是必要的. 首先介绍 4 种不同的 Streett 确定化结构: μ-Safra tree 和 H-Safra tree (最优) 将 Streett 确定化为 Rabin 自动机, compact Streett Safra tree 和 LIR-H-Safra tree (渐进最优) 将 Streett 确定化为 Parity 自动机; 然后, 根据 Streett 确定化算法, 基于开源工具 GOAL (graphical tool for omega-automata and logics), 实现了 Streett 确定化工具 NS2DR&PT, 以支持上述 4 种结构; 最后, 通过随机生成 100 个 Streett 自动机, 构造相应的测试集, 进行对比实验, 结果表明各结构状态复杂度的实际效果与理论论证一致, 此外, 对运行效率也进行了比较分析.

关键词: Streett 自动机; 确定化; Rabin 自动机; Parity 自动机; 工具

中图法分类号: TP311

中文引用格式: 王文胜, 田聪, 段振华. Streett自动机确定化工具. 软件学报, 2023, 34(8): 3659–3673. http://www.jos.org.cn/1000-9825/6614.htm

英文引用格式: Wang WS, Tian C, Duan ZH. Tool for Determinization of Streett Automata. Ruan Jian Xue Bao/Journal of Software, 2023, 34(8): 3659–3673 (in Chinese). http://www.jos.org.cn/1000-9825/6614.htm

Tool for Determinization of Streett Automata

WANG Wen-Sheng[1,2], TIAN Cong[1,2], DUAN Zhen-Hua[1,2]

[1](Institute of Computing Theory and Technology, Xidian University, Xi'an 710071, China)

[2](State Key Laboratory of Integrated Service Networks (Xidian University), Xi'an 710071, China)

Abstract: Determinization of an automaton refers to the transformation of a nondeterministic automaton into a deterministic automaton recognizing the same language, which is one of the fundamental notions in automata theory. Determinization of ω automata serves as a basic step in the decision procedures of SnS, CTL*, and μ-calculus. Meanwhile, it is also the key to solving infinite games. Therefore, studies on the determinization of ω automata are of great significance. This study focuses on a kind of ω automata called Streett automata. Nondeterministic Streett automata can be transformed into equivalent deterministic Rabin or Parity automata. In the previous work, the study has obtained algorithms with optimal state complexity and optimal asymptotical performance, respectively. Furthermore, it is necessary to develop a tool supporting Streett determinization, so as to evaluate the actual effect of proposed algorithms and show the procedure of determinization visually. This study first introduces four different Streett determinization structures including μ-Safra trees, H-

* 基金项目: 科技创新 2030—"新一代人工智能"重大项目 (2018AAA0103202); 国家自然科学基金 (61732013); 陕西省重点科技创新团队 (2019TD-001)
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: 2021-08-24; 修改时间: 2021-10-14; 采用时间: 2022-01-10; jos 在线出版时间: 2022-01-28
CNKI 网络首发时间: 2023-01-19

Safra trees, compact Streett Safra trees, and LIR-H-Safra trees. By H-Safra trees, which are optimal, and μ-Safra trees, deterministic Rabin transformation automata are obtained. In addition, deterministic parity transformation automata are constructed via another two structures, where LIR-H-Safra trees are asymptotically optimal. Furthermore, based on the open source software named graphical tool for omega-automata and logics (GOAL), the study develops a tool for Streett determinization and names it NS2DR&PT to support the four structures. Besides, corresponding test sets are constructed by randomly generating 100 Streett automata, and comparative experiments are carried out. Results show that the actual effect of state complexity in each structure is consistent with theoretical analysis. Moreover, the efficiency of different algorithms is compared and analyzed.

Key words: Streett automata; determinization; Rabin automata; Parity automata; tool

自动机的确定化是自动机理论的基本问题之一, 对于非确定性自动机$\mathcal{A}$, $\mathcal{A}$的确定化是指构造一个确定性自动机$\mathcal{B}$, 使得$\mathcal{A}$和$\mathcal{B}$接收相同的语言. ω 自动机于 20 世纪 60 年代首次提出, 独特的接收条件使其可以接收无限字, 用于解决数学和逻辑中的一些基本判定问题[1−3], 以及规约验证和非终止系统的合成[4]. 在模型检测中, 线性时序逻辑 (linear temporal logic, LTL)[5]是一种重要的规范语言, 自动机理论要求首先要将 LTL 公式转换为 ω 自动机, 然后再对该自动机与待检测系统的计算结果进行分析[6]. 而 ω 自动机确定化的应用更为广泛, 最直观的是可以用于求补, 例如, 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 [7] 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. 此外, ω 自动机的确定化是诸如 SnS、CTL*、μ 演算等逻辑判定过程的基础[8]; 也有助于解决无限博弈求解问题, 特别地, 针对 LTL 公式博弈求解的工具, 目前实际效果最好的 Strix[9]便是基于确定性 Parity 自动机实现的. 因此, 对 ω 自动机确定化的研究至关重要, 同时, 这也是计算复杂度理论中的重要问题.

根据接收条件的不同, ω 自动机可具体分为 Büchi[1]、Streett[10]、Rabin[2]以及 Parity[11]自动机等. 其中, 最常见的是 Büchi 自动机, 它的接收条件是状态集合的子集, 称为终态. 本文对 Streett 自动机进行研究, 其接收条件是由 k 个 Streett pairs 组成的集合, 每个 Streett pair 形如$\langle G_i, B_i\rangle$, G_i 和 B_i 均为状态集合的子集. 相比于 Büchi 自动机, Streett 自动机在描述系统的无限行为时更加简练[12], 因此, 在对并发和反应系统的行为进行建模方面具有一定优势[10].

关于 Streett 自动机确定化的研究已经长达数十年之久. 1992 年, Safra[13]首次创造性地提出了关于非确定性 Streett 自动机 (nondeterministic Streett automata, NSA) 的树状确定化结构, 称为 Streett Safra tree, 基于该结构, NSA 可以确定化为 Rabin 或 Parity 自动机. 此前, 确定化为 Rabin 自动机的最优结构是通过减少 Streett Safra tree 中节点索引标签的冗余, 以及使用批处理模式的节点命名规则得到的, 称为 μ-Safra tree[14]; 而确定化为 Parity 自动机的最优结构 compact Streett Safra tree[2]是通过对 Streett Safra tree 中的节点使用动态命名规则得到的.

在我们的前期工作[15]中, 分别对上述两种结构进行了改进. 在 μ-Safra tree 的基础上, 引入了索引节点命名规则, 即节点的名字仅根据索引标签决定, 由此得到一种新的确定化结构 H-Safra tree; 接下来, 通过在 H-Safra tree 上增加记录节点生成顺序的集合 LIR (later tntroduction records), 得到 LIR-H-Safra tree. 经过这两种结构, NSA 分别被确定化为具有更优状态复杂度的 Rabin (Rabin pair 数量有所增加) 和 Parity 自动机, 并且, 文献 [13] 证明了状态复杂度已分别达到最优 (紧界) 以及渐进最优 (渐近紧界).

而关于 ω 自动机确定化的工具, 目前仅有 OmegaDet[16]、Spot[17]和 GOAL (graphical tool for omega-autamata and logics)[18]. 其中, 工具 OmegaDet 是为了比较 Büchi 自动机的两种确定化算法 Safra[8]和 Muller-Schupp[19]的差异而开发的, 遗憾的是, 该工具后期并没有继续扩展和维护, 至今仍只支持初始的两种算法, 并且该工具缺少图形交互界面, 可读性与可操作性较差. Spot 是自动机理论领域著名的开源工具, 目前已支持各种类型自动机的确定化, 但也不支持图形交互界面, 为使用增加了难度. GOAL 是用于定义和操作 ω 自动机、时序逻辑公式以及博弈的图形交互工具, 有助于对计算机逻辑领域的理解. 该工具提供 ω 自动机的图形交互界面, 用户可方便、直观地修改状态、迁移和接收条件, 或进行其他相关操作. 经过版本的不断更新, 该工具已较全面的支持 Büchi 自动机的确定化算法, 但对其他 ω 自动机的确定化操作涉及很少, 特别是一些最新的算法.

总之, 现有的工具要么局限于 Büchi 自动机的确定化, 要么不支持图形交互界面, 无法满足我们的需求. 因此, 为了验证我们在文献 [15] 中提出的两种确定化结构的实际效果, 也为了形象地展示确定化过程, 使其更易理解,

我们需要开发一款可以自动化地完成 Streett 自动机确定化的工具, 以代替人工繁琐的操作. 而 Streett 自动机比 Büchi 自动机更具普遍性、更复杂. 同时, Streett 自动机的确定化树状结构与算法比 Büchi 自动机的复杂得多, 过程更抽象、繁杂, 导致实现相应的支撑工具也更困难, 并且工具的实现并不是对现有 Büchi 自动机确定化模块的简单修改, 而是对整体设计的重新考量, 这不仅要求对 Streett 自动机确定化结构与算法有深刻的理解, 而且需要对数据结构的设计有着准确的把握.

我们在前期的理论工作中已经对 Streett 自动机确定化过程有了深入探索, 这为我们工具的实现打下了坚实基础; 此外, 幸运的是, 具有强大图形交互界面的工具 GOAL 是开源的. 基于此, 本文设计实现了针对 Streett 自动机的确定化工具 NS2DR&PT, 支持 4 种确定化结构: μ-Safra tree 和 H-Safra tree, 以及 compact Streett Safra tree 和 LIR-H-Safra tree, 用户可以根据自己的需求进行选择. 通过调用 GOAL 的图形交互功能, 可以轻松、直观地实现 Streett 自动机的输入、定义和修改等, 生成的确定性自动机也可以形象地展示出来, 同时还可以查看每个状态对应的树状结构. 随后, 本文进行了实验性能比较, 由于缺少现有的测试集, 也为了实验的一般性, 我们利用 GOAL 中自动机随机生成的功能, 随机生成了 100 个 NSA 作为测试集. 使用工具 NS2DR&PT 对该测试集中所有的 NSA 分别经 4 种确定化结构进行确定化, 对比各算法在实例中的性能差异, 并以此证明工具 NS2DR&PT 的可用性.

本文第 1 节介绍 ω 自动机的定义与相关的接收条件. 第 2 节对 4 种确定化结构进行介绍. 第 3 节详细描述工具 NS2DR&PT 的整体框架和实现过程. 第 4 节展示工具的工作效果以及测试集的实验结果. 第 5 节总结全文并展望未来的工作.

1 ω 自动机

一般自动机的接收条件是定义在状态上的, 即接收条件的集合元素是状态; 而迁移自动机的接收条件是定义在迁移上的. 在确定化过程中, 得到的确定性自动机为迁移自动机时, 其状态数更少. 而确定化为一般自动机时, 需要为树状结构增加额外的集合来记录相关信息, 导致状态复杂度增大, 也就意味着工具实现以及实验对比将付出更多的成本. 同时, 迁移自动机的比较足以说明各算法之间的性能差异. 本文采用迁移自动机作为确定化得到的自动机. 因此, 本节将直接介绍接收条件定义在迁移上的 ω 自动机.

定义 1. ω 自动机. ω 自动机是一个五元组 $\mathcal{A}=\{Q, Q_0, \Sigma, \delta, \lambda\}$, 其中,

- Q 是非空有限状态集合.
- $Q_0\subseteq Q$ 是初始状态集合.
- Σ 是有限字母集合, 称为字母表.
- δ: $Q\times\Sigma\to 2^Q$ 是迁移函数, 具体地, $\delta(q, \sigma)$ 表示 $\mathcal{A}$ 读入 $\sigma\in\Sigma$ 后, 从 $q\in Q$ 能到达的所有状态集合.
- λ 是接收条件.

ω 自动机可以接收无限字. 一个无限字 α 是一个无限字母序列, 每个字母都属于 Σ, 形式化描述为 α: $\mathbb{N}\to\Sigma$, 其中, $\mathbb{N}$ 为非负整数集. 因此, 我们用 $\alpha(i)$ 表示 α 中第 i ($i\geqslant 0$) 个字母. Inf(α) 表示 α 中出现无限次的字母集合, 即 $\mathrm{Inf}(\alpha)=\{\sigma\in\Sigma|\exists^{\omega}n\in\mathbb{N}:\alpha(n)=\sigma$, 这里 $\exists^{\omega}n\in\mathbb{N}$ 的意思是在 $\mathbb{N}$ 中存在无限个不同的 n.

ω 自动机 $\mathcal{A}$ 关于无限字 α 的一条运行 (run)ρ 是一个无限迁移序列 ρ: $\mathbb{N}\to\delta$, 使得 $\rho(0)=(q_0, \alpha(0), q_1)$, 其中, $q_0\in Q_0$, $q_1\in\delta(q_0, \alpha(0))$, 并且对于所有的 $i\in\mathbb{N}$, $\rho(i)=(q_i, \alpha(i), q_{i+1})$, 其中, $q_{i+1}\in\delta(q_i, \alpha(i))$. 如果$|Q_0|=1$, 并且对于任意 $q\in Q$, $\sigma\in\Sigma$, 满足$|\delta(q, \sigma)|\leqslant 1$, 那么 $\mathcal{A}$ 被称为确定性的; 否则 $\mathcal{A}$ 被称为非确定性的. 和无限字类似, 用 Inf(ρ) 表示 ρ 中出现无限次的迁移集合, 即 $\mathrm{Inf}(\rho)=\{\delta_{\mathrm{Inf}}\in\delta|\exists^{\omega}n\in\mathbb{N}:\rho(n)=\delta_{\mathrm{Inf}}$.

不同的接收条件定义不同的自动机, 这里我们仅展示其中的 3 种.

- Streett: $\lambda=\{\langle G_1, B_1\rangle, \langle G_2, B_2\rangle, \ldots, \langle G_k, B_k\rangle\}$, 其中 $G_i, B_i\subseteq\delta$, 称$\langle G_i, B_i\rangle$为 Streett pair. ρ 是可接收的当且仅当对于所有的 $1\leqslant i\leqslant k$, 都有 $\mathrm{Inf}(\rho)\cap G_i\neq\varnothing$或者 $\mathrm{Inf}(\rho)\cap B_i=\varnothing$.
- Rabin: $\lambda=\{\langle A_1, R_1\rangle, \langle A_2, R_2\rangle, \ldots, \langle A_k, R_k\rangle\}$, 其中 $A_i, R_i\subseteq\delta$, 称$\langle A_i, R_i\rangle$为 Rabin pair. ρ 是可接收的当且仅当存在 i ($1\leqslant i\leqslant k$), 使得 $\mathrm{Inf}(\rho)\cap A_i\neq\varnothing$并且 $\mathrm{Inf}(\rho)\cap R_i=\varnothing$.
- Parity: $\lambda=\{\lambda_1, \lambda_2, \ldots, \lambda_{2k}\}$且 $\lambda_1\cup\lambda_2\cup\ldots\cup\lambda_{2k}=\delta$, 称 λ_i 为 priority. ρ 是可接收的当且仅当满足 $\mathrm{Inf}(\rho)\cap\lambda_i\neq\varnothing$的最

小索引 i 是偶数.

对于自动机 $\mathcal{A}$ 以及无限字 α, 若 $\mathcal{A}$ 中存在一条关于 α 的可接收 run, 则 $\mathcal{A}$ 接收 α. $\mathcal{A}$ 接收的所有字组成的集合称为语言, 记为 $L(\mathcal{A})$.

2 确定化结构

对于有限自动机的确定化, 子集构造法[20]是正确且完备的, 得到的自动机的状态是原自动机的状态集合. 但该方法并不适用于 ω 自动机, 以 Büchi 自动机 (最简单的 ω 自动机) 为例, 利用子集构造法对其进行确定化会扩大接收的语言, 即得到的自动机可能接收不被原 Büchi 自动机接收的字. 例如图 1 所示的非确定性 Büchi 自动机 $\mathcal{A}$, 终态集合为 $\{b\}$. 图 2 是子集构造法得到的自动机 $\mathcal{B}$, 每个状态是由 $\mathcal{A}$ 中状态组成的集合, 终态集合为 $\{\{a, b\}, \{a, b, c\}\}$. 而无限字 p^{ω} 被 $\mathcal{A}$ 拒绝却可以被 $\mathcal{B}$ 接收.

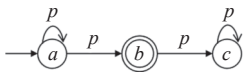


图 1 非确定性 Büchi 自动机 $\mathcal{A}$

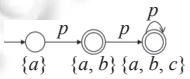


图 2 经子集构造法得到的自动机 $\mathcal{B}$

因此, ω 自动机的确定化需要对子集构造法进行扩展, 确定化得到的自动机不再是原自动机的状态集合, 而是由状态的子集组成的树状结构. 对于 NSA 的确定化过程, 概括来讲, 首先构造初始状态, 即初始树状结构; 然后自初始树开始, 依次读入字母表中的字母, 每读入一个字母, 树状结构都会发生转换, 得到的新树即为一个新状态, 期间需要记录树中某些节点信息, 以此循环进行, 直至没有新树生成, 所有的树构成状态集合, 树之间的转换关系便是迁移; 最后根据迁移中的节点信息定义自动机的接收条件. 由此便可得到所需的确定性自动机.

本节将对 Streett 自动机的 4 种树状确定化结构 (μ-Safra tree, H-Safra tree, compact Streett Safra tree 以及 LIR-H-Safra tree) 逐一进行介绍.

首先介绍将 NSA 确定化为确定性 Rabin 迁移自动机 (deterministic Rabin transition automata, DRTA) 的两种结构.

2.1 μ-Safra tree 和 H-Safra tree

μ-Safra tree 与 H-Safra tree 的不同之处仅在于树中节点的命名方式, 除此之外, 它们具有相同的结构, 称为包含状态和索引标签的结构有序树. 给定一个具有 n 个状态和 k 个 Streett pairs 的 NSA S=(Σ, Q, Q_0, δ, λ), S 的一棵包含状态和索引标签的结构有序树是一个五元组 T_{si}=⟨T_o, V, l, h, *stor*⟩, 其中,

- T_o 是一棵有序树.
- V 是 T_o 中所有节点的集合.
- l: $V\to 2^Q$ 是节点的状态标签 (state label), 并且满足: 每个节点的状态标签等于它所有孩子节点状态标签的并集; 任何两个兄弟节点的状态标签都不相交.
- h: $V\to 2^{[k]}$是节点的索引标签 (index label). 根节点的索引标签为 $[k]$=$\{1, 2, \ldots, k\}$. 每个非根节点 τ 的索引标签都包含在其父节点 τ_p 的索引标签中, 最多比父节点的索引标签少一个元素. 非叶子节点至少有一个索引标签是它的真子集的子节点.
- *stor* 定义兄弟节点之间的结构顺序. 对于非根节点 τ, 令 $j(\tau)=\max\{(h(\tau_p)\cup\{0\})-h(\tau)\}$, *stor* 的意思是对于任意两个兄弟节点 τ 和 τ', τ'的位置在 τ 的右侧当且仅当 $j(\tau)>j(\tau')$; 或 $j(\tau)=j(\tau')$ 且 τ 比 τ'先生成.

图 3 是一棵具有 9 个节点的包含状态和索引标签的结构有序树, 节点内的状态集合是状态标签, 集合 h 表示索引标签. 在此基础上, 不同的节点命名规则将对应不同的确定化结构.

μ-Safra tree 是 2012 年由 Cai 等人[14]提出的, 应用批处理模式的节点命名规则, 将节点划分到具有不同名字的分支, 接着为分支中的每个节点命名, 具体如下.

规则 1. 批处理模式命名规则 M_b[14]. 如果 τ 是名字为 b 的分支中的第 i 个节点, 那么 τ 的名字为 $b.i$, 即 $M_b(\tau)=b.i$.

由此可得 μ-Safra tree 的定义.

定义 1. μ-Safra tree[14]. 给定一个 NSA S, 它的一棵 μ-Safra tree 是一个二元组$\langle T_{si}, M_b\rangle$, 其中, T_{si} 是关于 S 的一棵包含状态和索引标签的结构有序树, M_b 是批处理节点命名规则.

图 4 是将图 3 中的节点按照规则 1 命名得到的一棵 μ-Safra tree, 包含 3 个分支, 即{1.1, 1.2, 1.3, 1.4}, {2.1, 2.2, 2.3}, {3.1, 3.2}. 图 4 中节点名字用加粗字体表示.

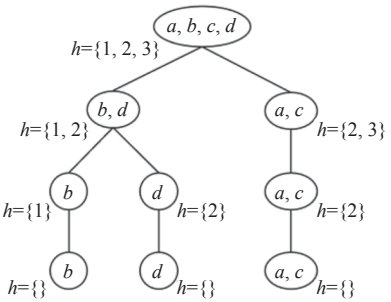


图 3 一棵包含状态和索引标签的结构有序树

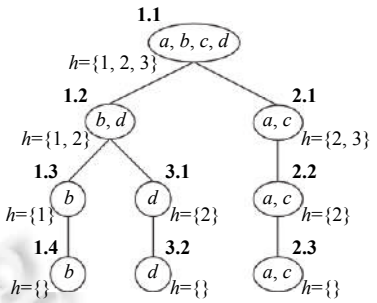


图 4 一棵 μ-Safra tree

而我们在前期工作[15]中提出的 H-Safra tree 对应索引节点命名规则. 节点的名字仅依赖于其索引标签与位置, 即给定一棵包含状态和索引标签的结构有序树, 其中每个节点的名字便唯一确定.

规则 2. 索引节点命名规则 M_h[15].

- 对于根节点 τ_r, $M_h(\tau_r)=\varepsilon$.
- 对于第 2 层的节点 τ, $M_h(\tau)=j(\tau)^{i+1}$, 其中 $i=|\{\tau' \mid \tau'$是 τ 的左兄弟, 且 $j(\tau')=j(\tau)\}|$.
- 对于其他的节点 τ, $M_h(\tau)=M_h(\tau_p).j(\tau)^{i+1}$.

应用该规则, 我们可以得到 H-Safra tree.

定义 2. H-Safra tree[15]. 给定一个 NSA S, 它的一棵 H-Safra tree 是一个二元组$\langle T_{si}, M_h\rangle$, 其中, T_{si} 是关于 S 的一棵包含状态和索引标签的结构有序树, M_h 是索引节点命名规则.

将图 3 中的每个节点按照规则 2 命名, 得到一棵 H-Safra tree, 如图 5 所示. 图 5 中加粗字体表示节点的名字.

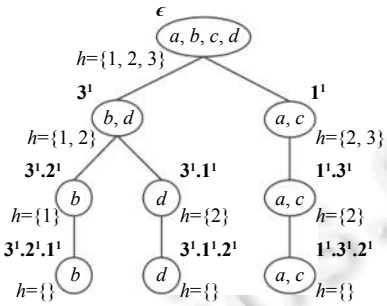


图 5 一棵 H-Safra tree

2.2 Compact Streett Safra tree 和 LIR-H-Safra tree

下面对将 NSA 确定化为确定性 Parity 迁移自动机 (deterministic Parity transition automata, DPTA) 的两种结构进行介绍. 2007 年, Piterman[12]通过动态节点命名, 得到了 compact Streett Safra tree.

定义 3. Compact Streett Safra tree[12]. 给定一个 NSA S, 它的一棵 compact Streett Safra tree 是一个五元组$\langle T_o, V, l, h, M_d\rangle$, 其中, T_o, V, l, h 与 T_{si} 中的定义相同, M_d:$V\rightarrow[n(k+1)]$ 是动态命名规则, 节点按生成顺序依次、连续命名.

图 6 是一棵 compact Streett Safra tree, 节点的名字用加粗字体表示. 除了节点命名方式不同, compact Streett Safra tree 并不要求兄弟节点之间满足结构顺序.

值得注意的是, 在确定化过程中, μ-Safra tree 和 H-Safra tree 要求叶子节点不断生成孩子节点, 直至叶子节点的索引标签满足相应的条件, 而 compact Streett Safra tree 仅要求叶子节点生成一次孩子节点, 因此, 一般情况下其深度偏小.

接下来, 我们前期工作[15]中提出的 LIR-H-Safra tree 是在 H-Safra tree 的基础上增加记录节点生成顺序的集合得到的.

定义 4. LIR-H-Safra tree[15]. 给定一个 NSA S, 它的一棵 LIR-H-Safra tree 是一个二元组$\langle H, LIR\rangle$, 其中, H 是关于 S 的一棵 H-Safra tree, LIR 是记录 H 中所有节点生成顺序的集合.

图 7 是一棵 LIR-H-Safra tree, LIR 中各节点用其名字表示, 按生成的先后顺序依次排列.

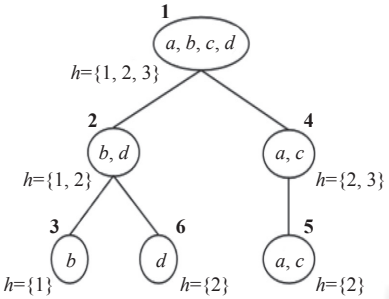


图 6 一棵 compact Streett Safra tree

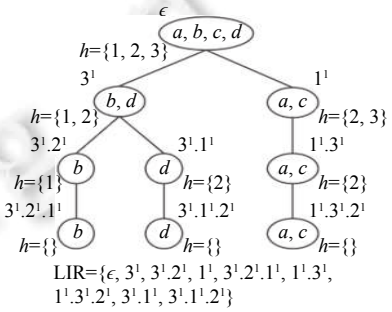


图 7 一棵 LIR-H-Safra tree

3 工具实现

本文在 GOAL 的基础上设计实现了关于 NSA 的确定化工具 NS2DR&PT, 该工具是基于 Windows 平台采用 Java 语言开发的, 整体框架如图 8 所示, 支持 4 种 NSA 确定化结构 (即图 8 中的树状结构, 确定化算法详见相应的参考文献).

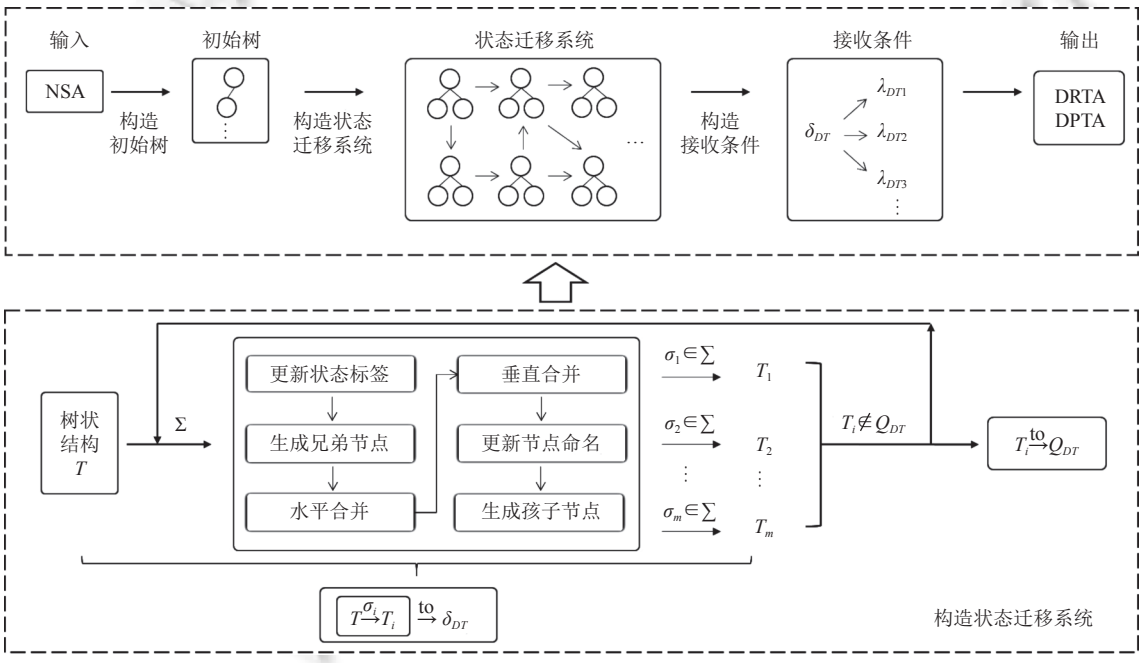


图 8 NS2DR&PT 的框架

• μ-Safra tree (NSA 确定化为 DRTA)[14].

• H-Safra tree (NSA 确定化为 DRTA)[15].

• compact Streett Safra tree (NSA 确定化为 DPTA)[12].

• LIR-H-Safra tree (NSA 确定化为 DPTA)[15].

在该工具中, 输入、输出操作均调用 GOAL 的相关模块, 充分利用其强大的图形交互界面. 下面我们主要对构造初始树、构造状态迁移系统和构造接收条件这 3 个模块进行介绍.

3.1 构造初始树

为了将 NSA $S=\{Q, Q_0, \Sigma, \delta, \lambda\}$确定化为 DRTA 或 DPTA $DT=\{Q_{DT}, Q_{DT0}, \Sigma, \delta_{DT}, \lambda_{DT}\}$, 首先要构造相应的初始树. 除了初始 compact Streett Safra tree 是单节点标记树, 其他 3 种初始结构均为单分支标记树.

图 9 是一个 NSA 关于字母 σ 的相关迁移片段, 若初始状态为 a 和 d, 其初始树如图 10 所示, $T_{\mu 0}$、H_0、LH_0、T_{c0} 分别是对应的初始 μ-Safra tree、H-Safra tree、LIR-H-Safra tree 以及 compact Streett Safra tree. 初始树将作为确定化后自动机 DT 的初始状态. 在程序中, 基本的数据结构是树, 树中的节点包含标签、名字、父亲和孩子等元素, 同时, 为了便于展示与比较, 我们定义树的特征字符串.

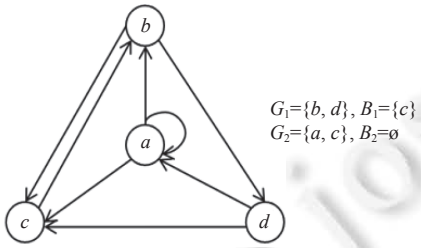


图 9 一个 NSA 关于 σ 的相关迁移

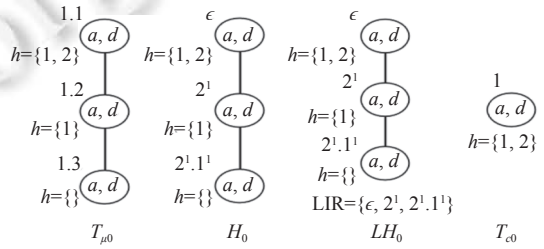


图 10 4 种结构的初始树

对于一棵具有 m 个节点的树 T, 从根节点开始, 层层递进, 每一层从左至右, 所有节点依次排列的顺序为 $\tau_1, \tau_2, \ldots, \tau_m$. 那么 T 的特征字符串为 $F(T)=F(\tau_1)F(\tau_2)\ldots F(\tau_m)$, 其中, 对于任意 $1\leqslant i\leqslant m$, $F(\tau_i)=f(\tau_i)[f(\tau_{i1})\ldots f(\tau_{ik})]$, $\tau_{i1}, \ldots, \tau_{ik}$ 为 τ_i 的所有孩子节点, 并且, $f(\tau_i)=\{name\}$-$\{l\}$-$\{h\}$表示节点 τ_i 的名字、状态标签、索引标签的连接, 对于 H-Safra tree, 节点的名字由索引标签唯一决定, 因此只需记录索引标签, 即 $f(\tau_i)=\{l\}$-$\{h\}$. 而对于 LIR-H-Safra tree, 其特征字符串 $F(T)=F(\tau_i)\ldots F(\tau_m)$-LIR.

那么, 图 10 中的初始树在程序中对应的特征字符串分别为:

$F(T_{\mu 0})$={1.1}-{a, d}-{1, 2}[{1.2}-{a, d}-{1}]{1.2}-{a, d}-{1}[{1.3}-{a, d}-{ }]{1.3}-{a, d}-{ }[ ];

$F(H_0)$={a, d}-{1, 2}[{a, d}-{1}]{a, d}-{1}[{a, d}-{ }]{a, d}-{ }[ ];

$F(LH_0)$={a, d}-{1, 2}[{a, d}-{1}]{a, d}-{1}[{a, d}-{ }]{a, d}-{ }[ ]-{ε, 2^1, $2^1.1^1$};

$F(T_{c0})$={1}-{a, d}-{1, 2}[ ].

3.2 构造状态迁移系统

自初始树开始, 依次读入 Σ 中的字母, 每个字母的读入都是树状结构的转换, 由此循环进行, 直至没有新树生成, 此间, 需比较特征字符串来判断生成的新树是否已存在.

经过循环转换过程, 每棵树都对应唯一的特征字符串, 均作为 DT 的一个状态存在, 生成的所有不同的树构成 DT 的状态集合 Q_{DT}, 而迁移集合 δ_{DT} 的元素对应树与树之间的转换. 因此, 在工具 NS2DR&PT 中, 最关键的是树的转换环节, 即当一棵树读入一个字母时, 如何得到另一棵树. 该转换过程需要 6 步完成, 是整个工具的核心.

第 1 步. 更新状态标签. 当一棵树读入一个字母时, 树中节点的状态标签需要根据 NSA 中的相关迁移进行更新.

第 2 步. 生成兄弟节点. 当节点的状态标签更新之后, 需要检查是否有状态经过了某个 Streett pair, 若有, 则将这些状态提取出来, 作为一个新生成的兄弟节点的状态标签, 同时需定义该节点的索引标签. 兄弟节点之间按结构

顺序排列. 需要注意的是, Piterman 的确定化算法并不要求兄弟节点之间满足结构顺序.

第 3 步. 水平合并. 此时兄弟节点的状态标签之间可能存在交集, 违反节点状态标签的定义. 因此, 需要将重复的状态从某些节点的状态标签中删除. 之后, 树中可能存在状态标签为空的节点, 需要将其删除, 在此, 删除的空节点称为拒绝节点 (rejecting nodes), 用于构造接收条件.

第 4 步. 垂直合并. 当前树中兄弟节点之间的状态标签已不相交, 且不存在空节点, 但可能存在某个节点的索引标签与其各个孩子节点的索引标签相等, 这违反了节点索引标签的定义. 因此, 需要删除该节点的所有后代节点, 并称此类节点为接收节点 (accepting nodes), 同样用于构造接收条件. 删除的后代节点为拒绝节点.

第 5 步. 更新节点命名. 因之前步骤中有节点被删除, 导致可能存在名字不符合命名规则的节点. 因此, 需要对节点的命名进行更新. 名字经过修改的节点同样称为拒绝节点.

第 6 步. 生成孩子节点. 经过前 5 步, 树中删除了某些节点. 因此, 需要生成新的孩子节点来补充, 以便在读入其他字母时更快的得到接收节点. 值得注意的是, 在 μ-Safra tree, H-Safra tree 和 LIR-H-Safra tree 中, 每个叶子节点生成的孩子将作为新的叶子节点, 继续生成孩子节点, 重复该操作, 直至无新节点生成. 而对于 compact Streett Safra tree, 每个叶子节点生成的孩子将不再生成新节点.

经过以上 6 步, 一棵树 T 读入字母 σ 可以转换得到另一棵树 T'. 然后, 通过匹配特征字符串来判断 T'是否已存在, 若不存在, T'将作为 DT 的一个新状态, $T \xrightarrow{\sigma} T'$ 作为 DT 的一条新迁移; 若存在, DT 仅新增一条迁移, 该迁移指向与 T'具有相同特征字符串的状态. 同时, 在此过程中, 需要对接收、拒绝节点进行记录. 对 μ-Safra tree 和 H-Safra tree, 每一次转换均需定义集合 Sig_{acc} 和 Sig_{rej}, 其中 Sig_{acc} 记录在此过程中所有的接收节点, Sig_{rej} 记录拒绝节点, 集合中的节点用其名字表示, 空集合即该过程中无接收或拒绝节点. 而对于 compact Streett Safra tree 和 LIR-H Safra tree, 树中包含所有节点的生成顺序, 每一次转换只需定义二元组 Sig=(i, st) 来记录接收和拒绝节点中生成顺序最小的节点, i 表示该节点的名字或在 LIR 中的位置, st 代表该节点是接收 (*acc*) 或拒绝 (*rej*) 的, Sig=∅ 表明该过程无接收或拒绝节点.

对于图 10 中的 4 种初始树, 读入 σ, 分别得到一棵新树, 如图 11 所示, 对应的特征字符串分别为:

$F(T_{\mu 1})$={1.1}-{a, b, c}-{1, 2}[{1.2}-{b}-{1}{2.1}-{a, c}-{2}]{1.2}-{b}-{1}[{1.3}-{b}-{ }]{2.1}-{a, c}-{2}[{2.2}-{a, c}-{ }]{1.3}-{b}-{ }[ ]{2.2}-{a, c}-{ }[ ];

$F(H_1)$={a, b, c}-{1, 2}[{b}-{1}{a, c}-{2}]{b}-{1}[{b}-{ }]{a, c}-{2}[{a, c}-{ }]{b}-{ }[ ]{a, c}-{ }[ ];

$F(LH_1)$={a, b, c}-{1, 2}[{b}-{1}{a, c}-{2}]{b}-{1}[{b}-{ }]{a, c}-{2}[{a, c}-{ }]{b}-{ }[ ]{a, c}-{ }[ ]-{ε, 2^1, 1^1, $2^1.1^1$, $1^1.2^1$};

$F(T_{c1})$={1}-{a, b, c}-{1, 2}[{2}-{a, b, c}-{1}]{2}-{a, b, c}-{1}[ ].

每棵树均作为对应自动机的新状态, 转换过程作为迁移, 并且, 在 $T_{\mu 0} \xrightarrow{\sigma} T_{\mu 1}$ 中, Sig_{acc}={1.2}, Sig_{rej}={1.3}; 在 $H_0 \xrightarrow{\sigma} H_1$ 中, Sig_{acc}={2^1}, Sig_{rej}={$2^1.1^1$}; 在 $LH_0 \xrightarrow{\sigma} LH_1$ 中, Sig=(2, *acc*); 在 $T_{c0} \xrightarrow{\sigma} T_{c1}$ 中, Sig=∅.

经过构造状态迁移系统模块, DT 的迁移系统便构造完成, 最后, 将确定接收条件 λ_{DT}.

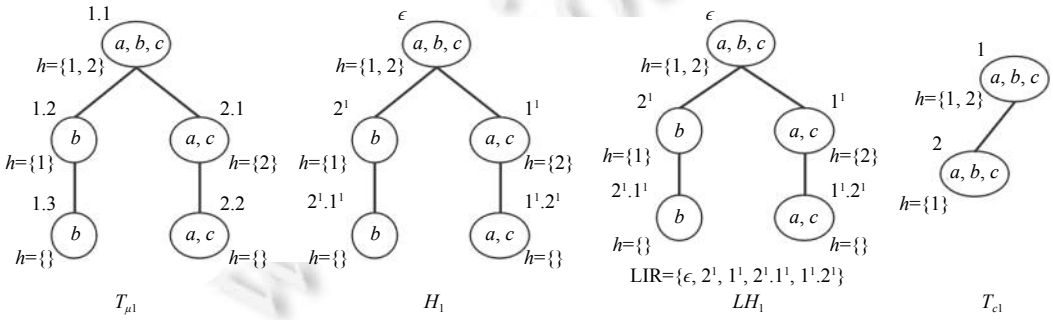


图 11 初始树读入 σ 的结果

3.3 构造接收条件

该模块根据每一条迁移记录的 Sig_* (Sig_{acc}、Sig_{rej} 或 Sig) 来分配该迁移.

• 若 DT 为 DRTA, 接收条件 λ_{DT} 为 Rabin pair (A_M, R_M) 组成的集合, 其中, A_M 是迁移集合, 且迁移中存在名字为 M 的接收节点; R_M 是迁移集合, 且迁移中存在名字为 M 的拒绝节点.

• 若 DT 为 DPTA, 接收条件 λ_{DT} 为 priority λ_m 组成的集合, 其中, λ_m 亦为迁移集合, 当 m 为偶数时, λ_m 包含 $Sig=(m/2, acc)$ 的迁移, 当 m 为奇数时, λ_m 包含 $Sig=((m+1)/2, rej)$ 的迁移, 特别地, λ_m 中最后一个 priority 包含 $Sig=\varnothing$ 或 $Sig=(1, rej)$ (根节点为拒绝节点) 的迁移.

对于图 10 至图 11 的迁移, (1) 在 $T_{\mu 0} \xrightarrow{\sigma} T_{\mu 1}$ 中, $Sig_{acc}=\{1.2\}$且 $Sig_{rej}=\{1.3\}$, 则该迁移属于 $A_{1.2}$ 以及 $R_{1.3}$; (2) 在 $H_0 \xrightarrow{\sigma} H_1$ 中, $Sig_{acc}=\{2^1\}$且 $Sig_{rej}=\{2^1.1^1\}$, 则该迁移属于 A_{2^1}以及 $R_{2^1.1^1}$; (3) 在 $LH_0 \xrightarrow{\sigma} LH_1$ 中, $Sig=(2, acc)$, 则该迁移属于 λ_4; (4) 在 $T_{c0} \xrightarrow{\sigma} T_{c1}$ 中, $Sig=\varnothing$, 则该迁移属于 λ_{DT} 中最后一个 priority λ_{2N+1}, 其中, N 为 compact Streett Safra tree 中允许的最多节点数.

在程序实现中, 该过程是运行时动态分配的 (on-the-fly), 当一条迁移构造完成, 便立即对其进行分配, 而非待整个迁移系统构造完成之后再逐一分配, 这样可以避免记录所有迁移的 Sig_*, 减少内存消耗.

至此, 迁移系统及其接收条件即构成与 NSA 等价的确定性自动机 DT. 工具 NS2DR&PT 同时支持这 4 种确定化结构, 可根据需求灵活选择.

4 工具展示

4.1 实例展示

本节通过一个 NSA 确定化的实例展示工具 NS2DR&PT 的工作效果. 图 12 为 NSA 示例, 包含 3 个状态, s_0 为初始状态, 接收条件中有 2 个 Streett pairs, 字母表为 $\{p, \neg p\}$, 为了显示方便, 工具中用~p 表示$\neg p$, True 表示 $p \vee \neg p$. 接下来利用工具将该 NSA 确定化为 DRTA 以及 DPTA.

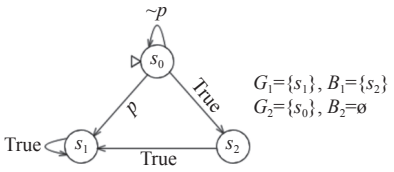


图 12 一个 NSA 示例

通过运行, 经 μ-Safra tree 将图 12 中的 NSA 确定化为 DRTA, 如图 13 所示, 且运行时间为 54 ms, 消耗内存 2 460 KB. 该 DRTA 具有 8 个状态和 4 个 Rabin pairs, 其中, s_0 为初始状态, Rabin pair 中的每个元素均为迁移集合. 此外, 每个状态均为 μ-Safra tree, 用其特征字符串来表示. 图 14 中各状态下方的字符串即为其对应的 μ-Safra tree 的特征字符串.

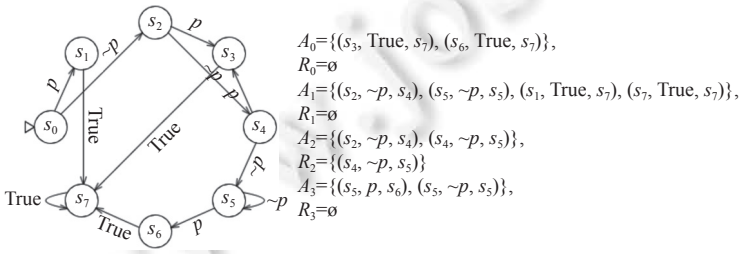


图 13 通过 μ-Safra tree 构造的 DRTA

同样地, 经 H-Safra tree, 工具也可以将图 12 中的 NSA 确定化为具有 7 个状态和 4 个 Rabin pairs 的 DRTA, 如图 15 所示, 且运行时间为 42 ms, 消耗内存 1 855 KB. 该 DRTA 的每个状态均为 H-Safra tree, 图 16 展示了各状

态的特征字符串. 此实例可以初步说明 H-Safra tree 能够减少 NSA 确定化的状态复杂度, 同时, 时间与内存的消耗也会相应减少. 至于 H-Safra tree 可以降低状态复杂度的原因, 从该实例中便可以看出, 在图 14 中, $s_5 \xrightarrow{p} s_6$, 而 s_6 与 s_3 的区别仅在于节点的名字, H-Safra tree 巧妙规避了节点名字的影响, 图 14 中的两个状态 s_6 与 s_3 等价于图 16 中的一个状态 s_3, 由此即缩减状态复杂度.

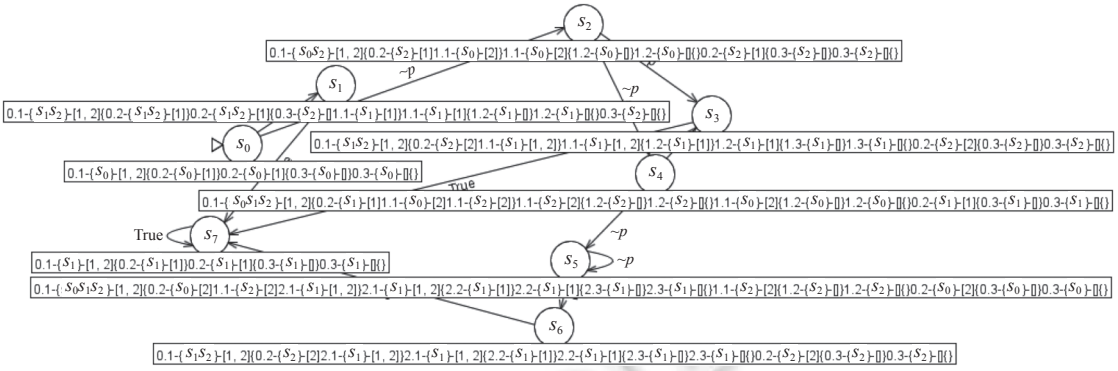
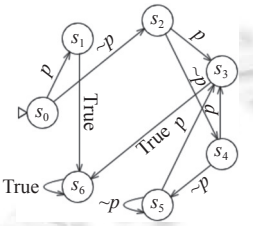


图 14 各状态对应 μ-Safra tree 的特征字符串



A_0={(s_2, ~p, s_4), (s_4, ~p, s_5), (s_5, ~p, s_5)},
R_0={(s_2, p, s_3), (s_4, p, s_3), (s_5, p, s_3), (s_3, True, s_6)}
A_1={(s_2, ~p, s_4), (s_1, True, s_6), (s_6, True, s_6)},
R_1={(s_2, p, s_3), (s_4, p, s_3), (s_4, ~p, s_5)}
A_2={(s_5, p, s_3), (s_5, ~p, s_5)},
R_2={(s_3, True, s_6)}
A_3={(s_3, True, s_6)},
R_3=ø

图 15 通过 H-Safra tree 构造的 DRTA

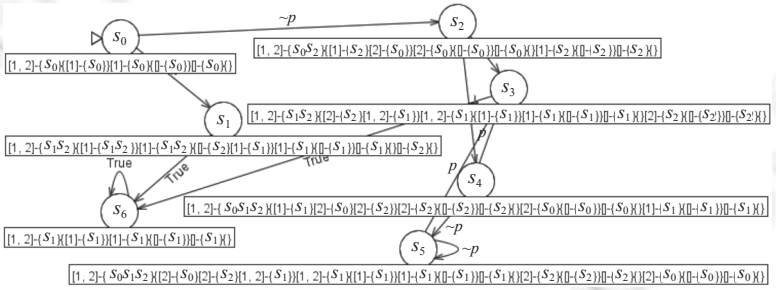
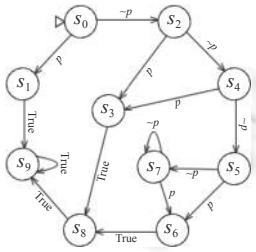


图 16 各状态对应 H-Safra tree 的特征字符串

此外, 经 compact Streett Safra tree, 图 12 中的 NSA 被确定化为 DPTA, 图 17 展示了工具的结果, 且运行时间为 37 ms, 消耗内存 1 263 KB. 该 DPTA 具有 10 个状态和 4 个 priorities, 并且每个状态均为 compact Streett Safra tree, 其特征字符串如图 18 所示.



λ_0={(s_6, True, s_8), (s_3, True, s_8)},
λ_1={(s_2, p, s_3), (s_4, p, s_3), (s_4, ~p, s_5), (s_5, p, s_6), (s_7, p, s_6)},
λ_2={(s_2, ~p, s_4), (s_5, ~p, s_7), (s_7, ~p, s_7), (s_9, True, s_9), (s_1, True, s_9)},
λ_3={(s_0, p, s_1), (s_0, ~p, s_2), (s_8,True, s_9)}

图 17 通过 compact Streett Safra tree 构造的 DPTA

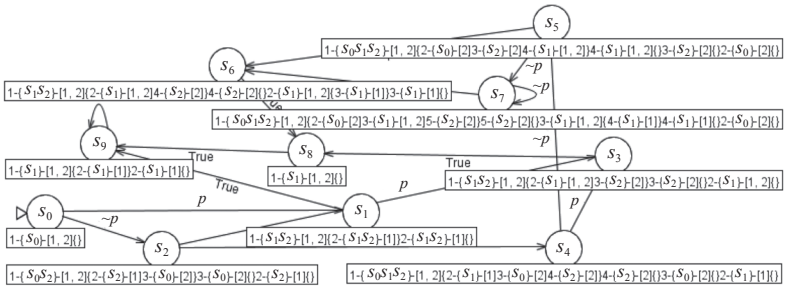


图 18　各状态对应 compact Streett Safra tree 的特征字符串

同样地, 经 LIR-H-Safra tree, 图 12 中的 NSA 也被确定化为 DPTA, 如图 19 所示, 而运行时间为 43 ms, 消耗内存 2646 KB. 该 DPTA 具有 9 个状态和 4 个 priorities, 各状态均为 LIR-H-Safra tree, 对应的特征字符串如图 20 所示. 该实例初步证明 LIR-H-Safra tree 在状态复杂度方面优于 compact Streett Safra tree, 但需要更多的时间和内存消耗.

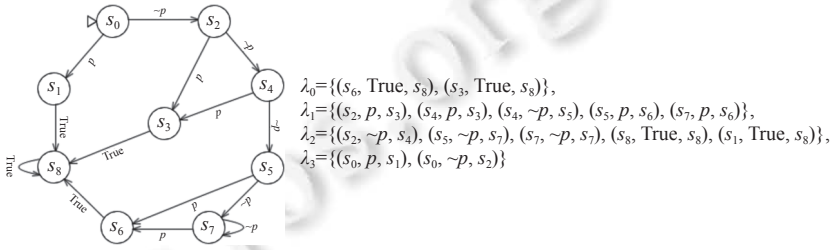


图 19　通过 LIR-H-Safra tree 构造的 DPTA

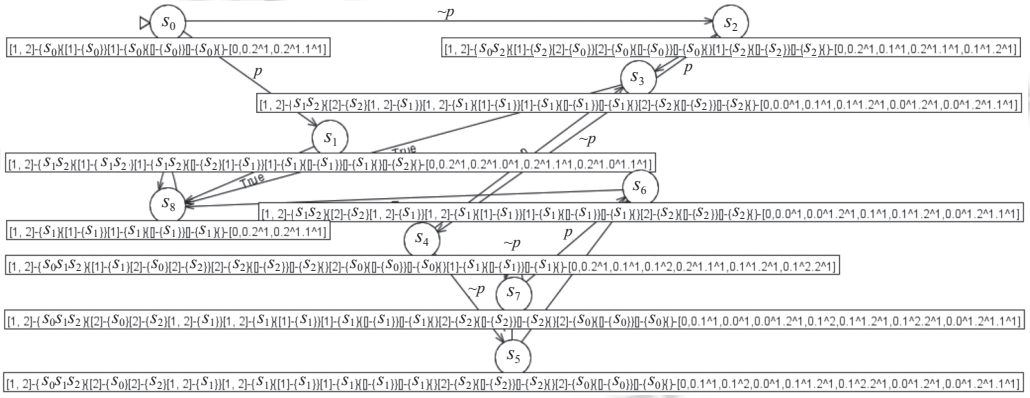


图 20　各状态对应 LIR-H-Safra tree 的特征字符串

至于确定化得到的自动机的正确性, 我们使用工具 Spot 来进行验证, 该工具的参数 equivalent_to 具有验证任意两个自动机等价性的功能. 图 21 展示的是本小节中涉及的 5 个自动机等价性的验证结果, 其中, NSA.hoa、DRTA_mu.hoa、DRTA_H.hoa、DPTA_com.hoa 和 DPTA_LIR.hoa 分别表示图 12、图 13、图 15、图 17 和图 19 中的自动机的 HOA 格式文件, 依次读入到参数 $a1$、$a2$、$a3$、$a4$ 和 $a5$ 中, $a1$.equivalent_to($a2$) 的输出结果为 True, 表示 $a1$ 和 $a2$ 代表的 NSA 和 DRTA_mu 这两个自动机是等价的. 该验证结果表明本

```
>>> import spot
>>> from spot.jupyter import display_inline
>>> spot.setup(show_default='.a')
>>> a1 = spot.automaton('NSA.hoa')
>>> a2 = spot.automaton('DRTA_mu.hoa')
>>> a3 = spot.automaton('DRTA_H.hoa')
>>> a4 = spot.automaton('DPTA_com.hoa')
>>> a5 = spot.automaton('DPTA_LIR.hoa')
>>> a1.equivalent_to(a2)
True
>>> a2.equivalent_to(a3)
True
>>> a4.equivalent_to(a5)
True
>>> a2.equivalent_to(a4)
True
```

图 21　自动机等价性的验证结果

小节涉及的 5 个自动机是互相等价的.

4.2 实验与分析

为了展示 4 种 NSA 确定化结构的实际效果, 以及工具 NS2DR&PT 的可用性, 需要利用测试集进行实验, 遗憾的是, 目前并没有相应的测试集可供使用. 因此, 为了不失一般性, 我们将随机生成一系列 NSA 来用作测试集, 该过程借助工具 GOAL 中的自动机随机生成功能, 仅需固定状态数和字母表, 迁移与 Streett pair 将随机分配. 由此, 我们构造了一个包含 100 个随机生成的 NSA 的测试集, 其中, 状态数为 5, 6,…, 14 的 NSA 各 10 个. 至于状态数的选择问题, 我们有如下考虑: 状态数小于 5 的 NSA 较简单, 通过不同结构确定化得到的自动机差别较小, 这种差别会随着状态数的增加而增大, 5 至 14 的状态数分布已完全满足实验需求, 过大的状态数将造成不必要的时间与内存消耗.

首先, 分别用工具 NS2DR&PT 中的两种确定化结构 μ-Safra tree 和 H-Safra tree 将测试集中的 NSA 确定化为 DRTA. 实验结果见表 1. 其中, NSA 表示测试集中不同状态的测试用例, 对于状态数相同的 NSA, 我们取其状态数、迁移数以及 Streett pairs 的平均值作为实验数据, 同样地, 相应的 DRTA 也取其平均值, 时间及内存表示整个确定化过程所消耗的时间和内存, 单位分别为 ms 和 KB. DRTA_μ 和 DRTA_H 分别表示经 μ-Safra tree 和 H-Safra tree 确定化得到的 DRTA.

表 1 NSA 确定化为 DRTA 的实验结果

NSA			DRTA_μ					DRTA_H					等价性
状态数	迁移数	Streett pairs	状态数	迁移数	Rabin pairs	时间 (ms)	内存 (KB)	状态数	迁移数	Rabin pairs	时间 (ms)	内存 (KB)	验证
5	16.5	5.1	39.5	79	5.7	123.9	4705.4	27.8	55.6	5.1	91.4	3382.8	T
6	16	5.9	68.4	136.8	4.9	111.5	12716.1	48.7	97.4	4.2	96.2	9176.7	T
7	19.9	7.5	319	638	8	1482.3	38005.7	147.6	295.2	8.4	346.2	23590	T
8	23.1	7.5	226.7	453.4	8.9	742.4	39542.1	154.4	308.8	10.6	501.5	28417	T
9	30.9	7.5	454.6	909.2	14.4	1963.1	74286.7	187.1	374.2	13.8	795.6	48413	T
10	44.9	5.9	362.8	725.6	17.8	1750.6	49722.9	189	378	18	912.8	33999	T
11	44.1	7.5	1071.8	2143.6	16.5	14007.9	92628.6	429.3	858.6	15.3	2904.4	63234	T
12	45.8	9.5	921.3	1842.6	16.5	8578.3	84616.3	532	1064	15.3	4047.6	44432	T
13	50.1	9.5	682.7	1365.4	15.5	5573.4	88149.3	430.7	861.4	14.8	2797.2	52365	T
14	62.3	8.1	1175.6	2351.2	24.2	26479.6	114039.3	552.4	1104.8	27.6	7194	55272	T

μ-Safra tree 与 H-Safra tree 的区别在于节点的命名方式, 而索引节点命名规则实质上是避免了节点名字对树状结构个数的影响, 从而降低确定化算法的状态复杂度. 表 1 的实验结果表明, 经 H-Safra tree 将 NSA 确定化得到的 DRTA 确实具有更少的状态数, 符合预期效果. 而对于状态数为 7、8、10 以及 14 的 NSA, 相应的 DRTA_H 具有较多的 Rabin pairs, 这与理论结果也是相符的. 在时间和内存方面, H-Safra tree 仍然占据优势, 这是因为两种结构的确定化过程相同, 而 H-Safra tree 的节点命名方式更加简洁, 有利于简化操作, 同时, DRTA 中更少的状态数意味着确定化过程中计算量更少, 也就消耗更少的时间和内存. 至于确定化前后自动机的等价性, 同样使用 Spot 工具中的 equivalent_to 参数对测试用例确定化的正确性进行验证, 表 1 中的最后一列展示了验证结果, 符号 T 表示确定化前后的自动机均是等价的.

接下来, 为了对比 compact Streett Safra tree 和 LIR-H-Safra tree 的效果, 分别用工具利用 2 种结构将测试集中的 NSA 确定化为 DPTA, 实验结果见表 2. 与表 1 相同, 表 2 中的数据亦为平均值. DPTA_com 和 DPTA_LIR 分别表示经 compact Streett Safra tree 和 LIR-H-Safra tree 确定化得到的 DPTA.

Compact Streett Safra tree 的动态节点命名规则与 LIR-H-Safra tree 的 LIR 并无本质上的区别, 目的均为记录节点的生成顺序. 这两种结构的主要区别在于 compact Streett Safra tree 在生成孩子节点时不需要反复操作, 且兄弟节点之间的顺序并无限制, 也就造成了冗余, 因此, 其确定化的状态复杂度高于 LIR-H-Safra tree. 表 2 的实验结果表明, 经 LIR-H-Safra tree 将 NSA 确定化得到的 DPTA 在状态数、迁移数以及 priorities 方面均优于 compact Streett Safra tree 的结果, 符合实验预期. 在时间和内存方面, compact Streett Safra tree 结构简单, 导致其具有较高确

定化状态复杂度的原因恰好为其减少大量操作, 这使得 compact Streett Safra tree 可能消耗更少的时间和内存, 关于状态数为 6、8、12 和 14 的 NSA 的实验恰好印证了这一点. 而 LIR-H-Safra tree 虽然操作复杂, 但得到的 DPTA 具有更少的状态数, 由此节省的时间和内存可以弥补由复杂操作增加的额外时间和内存消耗, 甚至节省的大于增加的, 此时, LIR-H-Safra tree 将更具优势, 实验中状态数为 5、7、9、10 和 11 的 NSA 确定化为此提供了佐证, 并且两种结构得到的 DPTA 的状态数差距越大, LIR-H-Safra tree 的优势越明显. 对于状态数为 13 的 NSA 确定化实验, LIR-H-Safra tree 所需的时间更少, 但内存消耗更高, 这表明该结构的复杂操作造成的内存消耗高于时间消耗, 在确定化过程中需记录更多的内容. 此外, 注意到状态数为 11 时, compact Streett Safra tree 消耗了大量的时间和内存, 甚至超过状态数更多的测试用例, 原因在于状态数少的 NSA 确定化后得到的自动机的状态数不一定少, 在测试集中存在一个状态数为 11 的 NSA, 其结构复杂, 确定化后的自动机拥有最大的状态数, 导致大量的时间和内存被消耗. 最后, 同样也对确定化前后自动机的等价性进行了验证, 结果符合预期.

表 2　NSA 确定化为 DPTA 的实验结果

NSA			DPTA_com					DPTA_LIR					等价性
状态数	迁移数	Streett pairs	状态数	迁移数	priorities	时间 (ms)	内存 (KB)	状态数	迁移数	priorities	时间 (ms)	内存 (KB)	验证
5	16.5	5.1	106.5	213	11.8	117.3	11833	31.6	63.2	7.6	108.9	8333.3	T
6	16	5.9	118.5	237	11	122.7	14054	50.8	101.6	9.1	131.4	14647	T
7	19.9	7.5	725	1450	14.2	5795.7	39035	170.8	341.6	12.9	677.9	27502	T
8	23.1	7.5	330.4	660.8	13.8	324	39315	176	352	12.5	608	49881	T
9	30.9	7.5	798.8	1597.6	14.6	8649.1	81899	249.7	499.4	12.6	1565.3	70362	T
10	44.9	5.9	576.1	1152.2	16.8	1359.5	55161	224.4	448.8	12.9	1047.2	50045	T
11	44.1	7.5	1340.2	2680.4	16.8	102144	95461	531	1062	16	4803.4	90782	T
12	45.8	9.5	890.1	1780.2	17.2	1994.3	71109	583.8	1167.6	15.2	5261.8	105301	T
13	50.1	9.5	743.8	1487.6	16.2	3285.2	51565	459.6	919.2	15.5	3133.4	73539	T
14	62.3	8.1	1117.3	2234.6	17.6	7906.1	49326	618.6	1237.2	13.7	8713.7	66459	T

为了了解工具 NS2DR&PT 确定化的极限能力, 我们将对一系列随机生成的 NSA 进行测试. 该测试集中 NSA 的状态数为 50, 60, 70, 80, …逐渐增大, 且每个状态数下的 NSA 各 10 个. 由于确定化难度与自动机的结构有关, 因此, 在随机生成 NSA 时, 有如下约束: (1) 从一个状态到另一个状态之间存在迁移的概率为 20%; (2) 一个状态属于某个 Streett pair 的概率为 20%; (3) Streett pair 的数目为状态数的 20% (这些数据均可进行修改). 由此尽可能保证所有 NSA 具有相同的结构, 但即便如此, 也不可避免存在差异. 在此基础上, 我们从状态数为 50 的 NSA 开始测试, 状态数依次增大, 设置测试时间上限为 1 小时. 因为要测试工具的极限能力, 所以我们记录每种算法中各个状态数的 NSA 所用的最小时间, 以便测试出工具在规定时间内可以处理的 NSA 的最大状态数. 需要注意的是, 该极限测试是对于我们的约束而言的, 实验环境为 3.40 GHz, Intel(R) Core(TM) i7-6700 CPU, 12 GB 内存 PC.

实验结果如表 3 所示, 展示了每种算法中各个状态数的 NSA 确定化所需的最小时间, “—”表示超时. 工具中的确定化结构 μ-Safra tree、H-Safra tree 以及 LIR-H-Safra tree 在 1 小时内可以处理测试集内 NSA 的最大状态数分别为 120、150 以及 140. 随着 NSA 状态数的增加, 运行时间也大幅增加, 原因在于减少确定化状态复杂度的措施引入了大量计算, NSA 的状态数越多, 计算量越大, 导致即使得到的确定性自动机状态数很小, 也会消耗大量的时间. 所以, 这 3 种算法确定化的运行时间不仅与得到的确定性自动机的状态数有关, 还跟 NSA 的状态数有密切关系.

表 3　每种算法中各个状态数的 NSA 确定化所需的最小时间 (s)

算法	50	60	70	80	90	100	110	120	130	140	150	160
DRTA_μ	343.2	459.0	703.8	774.5	1270.3	1514.3	2953.3	3588.4	—	—	—	—
DRTA_H	143.1	220.7	363.4	385.7	716.0	860.2	1725.7	1927.3	2325.7	3012.4	3426.2	—
DPTA_com	4.5	2.9	3.2	2.2	2.9	2.4	3.9	4.3	3.7	4.0	4.3	4.2
DPTA_LIR	160.3	235.8	387.1	406.6	747.2	890.3	1769.6	1987.1	2378.9	3025.4	—	—

而工具中 compact Streett Safra tree 对应的算法随 NSA 状态数的增加, 运行时间只有小幅度变化, 其确定化的极限能力并不明显, 且所用时间远远小于其他 3 种算法. 通过对得到的 DPTA 观察得知, 不同状态数的 NSA 得到的 DPTA 状态数差距不大, 且状态数均较小 (1 000 个状态以内). 这表明 compact Streett Safra tree 对应的算法运行时间与 NSA 的状态数关系不大, 主要与得到的 DPTA 的状态数有关. Compact Streett Safra tree 结构简单、确定化过程中计算量少并且得到的 DPTA 状态数较小是其运行时间短的原因. 因此, 尽管该算法确定化的状态复杂度较高, 但对状态数较大的 NSA 的实际支持性要明显优于其他 3 种算法.

综上, 我们在前期工作[15]中提出的两种确定化结构在实际应用中可以大幅缩减状态空间, 是对理论结果正确性的佐证. 时间和内存的数据统计有助于更全面的对比各结构之间的差异. 对确定化前后自动机等价性的验证充分说明各算法的正确性, 是对理论证明的补充, 也表明工具 NS2DR&PT 是对各算法的正确实现. 同时, 展示出工具 NS2DR&PT 的实际应用效果较好, 支持不同的确定化结构, 可以根据需求选择算法, 具有较强的灵活性与实用性. 最后, 对工具确定化的极限能力进行了讨论分析.

5 结 论

Streett 自动机的确定化应用于求补、非终止系统的合成与验证、时序逻辑的判定、无限博弈求解等, 具有重要的研究意义. 本文针对 4 种确定化结构 μ-Safra tree、H-Safra tree、compact Streett Safra tree 和 LIR-H-Safra tree, 基于可视化开源软件 GOAL, 实现了 Streett 自动机确定化工具 NS2DR&PT, 可以将 NSA 确定化为 DRTA 或 DPTA. 此外, 通过构造测试集, 进行对比实验, 结果与理论分析保持一致, 证明我们前期提出的 H-Safra tree 和 LIR-H-Safra tree 在实际状态空间缩减方面具有一定优势. 对时间和内存消耗的比较更全面地揭示了各结构之间的效率差异. 同时, 实验展示了该工具的可用性.

未来将进一步优化 Streett 自动机确定化工具 NS2DR&PT, 并在此基础上补充其他 ω 自动机的确定化算法, 以求全面支持 ω 自动机的确定化.

References:

[1] Büchi JR. On a decision method in restricted second order arithmetic. In: Nagel E, Suppes P, Tarski A, eds. 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] McNaughton R. Testing and generating infinite sequences by a finite automaton. Information and Control, 1966, 9(5): 521–530. [doi: 10.1016/S0019-9958(66)80013-X]

[4] 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]

[5] Pnueli A. The temporal logic of programs. In: Proc. of the 18th Annual Symp. on Foundations of Computer Science. Providence: IEEE, 1977. 46–57. [doi: 10.1109/SFCS.1977.32]

[6] Esparza J, Křetínský J, Sickert S. A unified translation of linear temporal logic to ω-automata. Journal of the ACM, 2020, 67(6): 33. [doi: 10.1145/3417995]

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

[8] 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]

[9] 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]

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

[11] Mostowski AW. Regular expressions for infinite trees and a standard form of automata. In: Proc. of the 5th Computation Theory. Zaborow: Springer, 1985. 157–168. [doi: 10.1007/3-540-16066-3_15]

[12] 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]

[13] 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. British Columbia: ACM, 1992. 275–282. [doi: 10.1145/129712.129739]

[14] Cai Y, Zhang T. Can nondeterminism help complementation? In: Proc. of the 3rd Int'l Symp. on Games, Automata, Logics and Formal Verification. Naples: GandALF, 2012. 57–70.

[15] 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]

[16] 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]

[17] 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]

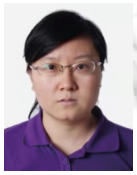
[18] Tsay YK, Chen YF, Tsai MH, Wu KN, Chan WC. GOAL: A graphical tool for manipulating Büchi automata and temporal formulae. In: Proc. of the 13th Int'l Conf. on Tools and Algorithms for the Construction and Analysis of Systems. Braga: Springer, 2007. 466–471. [doi: 10.1007/978-3-540-71209-1_35]

[19] Muller DE, Schupp PE. Simulating alternating tree automata by nondeterministic automata: New results and new proofs of the theorems of Rabin, McNaughton and Safra. Theoretical Computer Science, 1995, 141(1–2): 69–107. [doi: 10.1016/0304-3975(94)00214-4]

[20] Rabin MO, Scott D. Finite automata and their decision problems. IBM Journal of Research and Development, 1959, 3(2): 114–125. [doi: 10.1147/rd.32.0114]

王文胜(1993－), 男, 博士, 主要研究领域为自动机理论, 时序逻辑.

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.



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.