软件学报 ISSN 1000-9825, CODEN RUXUEW
*Journal of Software*, 2022,33(8):2797−2814 [doi: 10.13328/j.cnki.jos.006599]

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

# 运用时间分类树的确定单时钟时间自动机学习*

米钧日[1], 张苗苗[1], 安 杰[2], 杜博闻[3]

[1](同济大学 软件学院, 上海 201804)

[2](Max Planck Institute for Software Systems, D-67663 Kaiserslautern, Germany)

[3](University of Warwick, Coventry, CV4 7AL, UK)

通信作者: 张苗苗, E-mail: miaomiao@tongji.edu.cn

**摘 要**: 时间自动机的模型学习算法旨在通过提供输入和观察输出构建软硬件系统的形式化模型. 确定性单时钟时间自动机的学习是其中的一个重要研究方向, 但是该算法具有一定的局限性, 在状态较多时学习速度较慢, 很难应用到复杂的系统中. 由此, 提出了一种改进的学习算法, 使用逻辑时间分类树代替逻辑时间观察表作为学习算法的内部数据结构, 有效地减少了成员查询次数, 降低了算法的空间复杂度, 并能够高效率地构建假设自动机. 最后进行了相关实验, 实验结果表明, 提出的改进算法减少了 60%左右的成员查询和 5%左右的等价查询. 同时在该实验中, 改进算法的学习速度最高可提高 45 倍以上.

**关键词**: 模型学习; 主动学习; 确定性单时钟时间自动机; 时间语言; 逻辑时间分类树

**中图法分类号**: TP311

中文引用格式: 米钧日, 张苗苗, 安杰, 杜博闻. 运用时间分类树的确定单时钟时间自动机学习. 软件学报, 2022, 33(8): 2797−2814. http://www.jos.org.cn/1000-9825/6599.htm

英文引用格式: Mi JR, Zhang MM, An J, Du BW. Learning Deterministic One-clock Timed Automata Based on Timed Classification Tree. Ruan Jian Xue Bao/Journal of Software, 2022, 33(8): 2797−2814 (in Chinese). http://www.jos.org.cn/1000-9825/6599.htm

## Learning Deterministic One-clock Timed Automata Based on Timed Classification Tree

MI Jun-Ri[1], ZHANG Miao-Miao[1], AN Jie[2], DU Bo-Wen[3]

[1](School of Software Engineering, Tongji University, Shanghai 201804, China)

[2](Max Planck Institute for Software Systems, D-67663 Kaiserslautern, Germany)

[3](University of Warwick, Coventry, CV4 7AL, UK)

**Abstract**: Model learning of timed automata (TA) aims to build a formal model of software and hardware systems by external inputs and outputs. Learning of deterministic one-clock TA is one of the important research directions, but current algorithm has some limitations and is difficult to be applied to complex systems. Therefore, an improved learning algorithm is proposed, which uses logical-time classification tree instead of logical-time observation table as the internal data structure of the learning algorithm, effectively reducing the number of membership queries and the space complexity of the algorithm. In addition, it can efficiently construct hypothetical automata. Finally, relevant experiments have been carried out, and the experimental results show that the improved algorithm proposed in this study reduces the number of member queries by 60% and the number of equivalent queries by 5%. At the same time, in this experiment, the learning speed of the improved algorithm can be increased by more than 50 times at most.

**Key words**: model learning; active learning; deterministic one clock timed automata (TA); timed language; logic timed classification tree

* 基金项目: 国家自然科学基金(61972284, 62032019)
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: 2021-09-26; 修改时间: 2021-10-14; 采用时间: 2022-01-10; jos 在线出版时间: 2022-01-28

1987 年, Angluin 提出的一种名为 L*的模型学习算法[1]. 给定一个正则语言 L, 该算法可以在多项式时间内学习得到一个最小的识别目标语言 L 的确定性有限自动机(deterministic finite automata, DFA). 该算法构造了基于两个角色: 老师(teacher)和学生(leaner)的主动学习框架, 即老师掌握目标语言, 学生可以通过成员查询和等价查询两种问题来向老师学习该语言. 其中, 成员查询是指判断给定的一个字(word)是否属于目标自动机的语言(language), 老师可以回答“是”或“否”; 等价查询是指判断构造的猜想自动机的语言是否与目标语言相等, 老师可以回答“是”或“否”, 并且在给出否定回答时提供一个字作为反例以表示两个语言的差异. 算法的过程是: 学生通过多次成员查询获取一些字的查询结果, 并将这些字和查询结果存放在一张观察表中, 当获取到足够信息并通过操作使得观察表满足封闭性(closed)和一致性(consistent)时, 学生利用观察表中的信息构造出一个 DFA 作为当前对目标语言的假设, 然后通过等价查询判断该假设是否正确. 如果正确, 学习算法结束, 即当前假设的语言与目标语言相等. 如果不正确, 学生根据老师给出的反例进行新的成员查询, 收集更多信息, 并进一步构造新的假设自动机. 该过程直至当前假设自动机的等价查询结果为真, 即构造的假设自动机的语言与目标自动机的语言相等.

在此基础上, 国内外学者将模型学习算法扩展到了更多的形式模型, 例如 Mealy 机[2−4]、非确定有限自动机[5]、符号自动机[6−8]、寄存器自动机[9−11]、Büchi 自动机[12,13]、马尔可夫决策过程[14]等. 这些算法在形式化模型的建模和验证[15]有着很多应用. 最近两年, 安杰等人在观察表中引入了时间信息, 提出了实时自动机(real timed automata, RTA)以及确定性单时钟时间自动机(deterministic one-clock timed automata, DOTA)的模型学习算法[16,17]. 沈炜等人将 PAC (probably approximately correct leaning)理论应用到 DOTA 的等价查询算法中, 为等价查询提供错误率和置信度等统计保证, 从而为 DOTA 的模型学习在实际场景中的应用提供了理论支持[18].

然而, 目前文献[17]基于观察表的 DOTA 的模型学习算法具有一定的局限性, 状态较多时学习速度较慢, 难以应用到较为复杂的实际系统中. 通过对该算法进行分析, 我们发现该算法主要存在以下两个问题.

(1) 在理论复杂度方面, 存在大量冗余的成员查询. 具体如下, 当等价查询不满足时, 老师给出一个反例, 学生处理这个反例的方法是将反例的所有前缀加入到观察表中. 由于并不是所有的前缀都携带了修改假设的有效信息, 这种处理方式必然会在观察表中增加冗余的前缀, 且这些前缀在后续的学习过程中会持续产生冗余的成员查询等负面影响, 从而降低了学习算法的效率.

(2) 在实际运行方面, 构造满足封闭性和一致性的观察表的操作耗时很多. 具体如下, 在学习算法执行过程中, 学生不断地对假设 DOTA 进行迭代更新. 每次迭代, 学生都需要重新检查观察表并通过相应操作使得观察表满足封闭性和一致性, 然后根据观察表构建出一个新的假设 DOTA. 当观察表的规模较大时, 相关操作耗时明显, 而在模型学习理论中这些操作的时间复杂度均被忽视. 再者, 对于迭代前后的两个假设 DOTA 来说, 其迁移信息大体相同, 仅有少量改动. 但由于观察表只能隐式地保存这些迁移信息, 所以即使是两个假设中相同的状态迁移, 也需要学生重新进行计算才能得到, 这是该学习算法效率低的另一个原因.

对于一些不带时间信息的形式模型的模型学习算法, 国内外学者提出了一些优化算法, 其中使用分类树代替观察表作为算法的内部数据结构是一种较为通用的方案. 该方案在 DFA 和符号自动机的模型学习算法中都取得了较好的效果[8,19]. 基于相关工作的启发, 针对文献[17]中灰盒情形下 DOTA 的学习, 我们提出了改进的学习算法并给予实现. 我们的主要工作总结如下.

(1) 受到文献[20]的启发, 我们在分类树中扩展了时钟信息, 提出了逻辑时间分类树的概念. 基于逻辑时间分类树, 我们提出了改进的 DOTA 学习算法, 该算法可以对反例进行分析找到假设和目标自动机第 1 次分歧的错误下标, 并且对该错误下标进行处理. 从而降低了成员查询的数量, 提高学习的速度, 这从根本上解决了上述的第 1 个问题.

(2) 运用逻辑时间分类树的相关操作取代了耗时的观察表检查和相应操作, 并且在逻辑时间分类树中增加了叶节点迁移用以存放假设自动机的迁移信息, 使学生不需要复杂的计算即可获得假设 DOTA 的状态迁移信息, 从而提高了构造假设 DOTA 的效率, 这解决了上述的第 2 个问题.

(3) 实现了改进的 DOTA 学习算法, 并与现有的 DOTA 学习算法在灰盒情形下进行了比较和评估, 实验表明我们提出的改进算法可以非常明显地减少了成员查询数量并且提高 DOTA 学习算法的效率.

本文第 1 节主要介绍 DOTA 的模型学习所涉及的一些背景知识. 第 2 节介绍逻辑时间分类树的形式化定义和常用操作. 第 3 节对整个改进算法的流程进行说明, 并分析算法的终止性和复杂性. 第 4 节通过一个例子说明本文提出的改进算法的流程. 第 5 节通过对比实验说明本文提出的改进算法的优越性. 最后在第 6 节对本文工作进行总结并提出进一步的研究方向.

## 1 预备知识

我们使用符号$\mathbb{N}$表示自然数, $\mathbb{R}_{\geqslant 0}$ 表示大于等于 0 的实数, $\mathbb{B}=\{\top,\bot\}$ 表示布尔集合, 其中, $\top$表示 true, $\bot$表示 false. 给定一个元组 $z=(\alpha_1,\alpha_2,...,\alpha_n)$, 其中, $n$>2, 用 $\Pi_{\{1,2\}}(z)=(\alpha_1,\alpha_2)$ 表示 $n$ 元组 $z$ 的前两位元素的投影, 即用元组 $z$ 的前两个元素组成的一个新元组. 对于一个元组序列 $Z=\{z_1,z_2,...,z_n\}$, 该操作可以扩充为 $\Pi_{\{1,2\}}(Z)=\{\Pi_{\{1,2)}(z_1),\Pi_{\{1,2)}(z_2),...,\Pi_{\{1,2)}(z_n)\}$.

### 1.1 确定性单时钟时间自动机

时钟变量(clock): 简称时钟, 是时间自动机描述时间变化的变量, 在本论文中我们用 $x$ 表示时钟变量.

时钟解释(clock interpretation): 指将时钟变量 $x$ 映射到一个非负实数上, 用 $v: X\to\mathbb{R}_{\geqslant 0}$ 来表示.

时钟重置(clock reset): 表示将时钟变量 $x$ 重置为 0.

时钟约束(clock constraint): 指时钟变量需要满足的条件, 我们用$\phi(x)$来表示 $x$ 的时钟约束, $\phi(x):=x\geqslant m\,|\,m\geqslant x\,|\,x<m\,|\,m<x\,|\,x=m\,|\,\phi_1\wedge\phi_2$, 其中, $m\in\mathbb{N}$.

**定义 1(单时钟时间自动机)**. 一个单时钟时间自动机(one-clock timed automaton, OTA)是一个六元组 $\mathcal{A}=(Q,q_0,\Sigma,F,x,E)$, 其中,

- $Q$ 表示位置的有穷集合;
- $q^0\in Q$ 表示初始位置;
- $\Sigma$是字母表, 表示动作的有穷集合;
- $F\subseteq Q$ 是接收位置的有穷集合;
- $x$ 表示唯一的时钟;
- $E\subseteq Q\times\Sigma\times\Phi\times\mathbb{B}\times Q$ 是迁移集合, 其中, $\Phi$是指时钟约束的集合. 迁移集合中的一条迁移 $e=(q,\sigma,\phi,b,q')$ 表示当时钟 $x$ 满足时钟约束$\phi$时, 时间自动机从源位置 $q$ 经过动作$\sigma$到达目标位置 $q'$, 如果布尔值 $b=\top$, 则表示发生这个迁移之后, 时钟 $x$ 重置为 0.

单时钟时间自动机的状态是一个二元组$(q,v)$, 其中, $q$ 表示时间自动机的一个位置, $v$ 是时钟 $x$ 的一个时钟解释. 对于单时钟时间自动机上的一个执行 $\rho=(q_0,v_0)\xrightarrow{\sigma_1,\tau_1}(q_1,v_1)\xrightarrow{\sigma_2,\tau_2}...\xrightarrow{\sigma_n,\tau_n}(q_n,v_n)$, 其对应的路径$\omega$是全局时间字, $\omega=trace(\rho)=(\sigma_1,\tau_1)(\sigma_2,\tau_2)...(\sigma_n,\tau_n)$, 其中, $\sigma_i\in\Sigma$ 表示字母表中的一个动作, $\tau_i\in\mathbb{R}_{\geqslant 0}$ 表示外部观测到的全局时间, 且对任意 $i\geqslant 1$, 都满足 $\tau_i\leqslant\tau_{i+1}$. 全局时间字$\omega$对应的全局重置时间字(global reset timed word)表示为 $\omega_r=trace_r(\rho)=(\sigma_1,\tau_1,b_1)(\sigma_2,\tau_2,b_2)...(\sigma_n,\tau_n,b_n)$, 其中, $b_i\in\mathbb{B}$ 表示动作 $\sigma_i$ 发生时, 时钟 $x$ 是否重置.

**定义 2(确定性单时钟时间自动机)**. 对于一个单时钟时间自动机, 给出任意一个全局时间字$\omega$, 如果该单时钟时间自动机中最多只存在一个对应的执行, 那么我们称该单时钟时间自动机是一个确定性的单时钟时间自动机.

### 1.2 时间语言和逻辑重置时间语言

对于一个 DOTA, 如果全局时间字$\omega$存在一个对应的执行, 且该执行最终到达了一个接收位置, 那么我们就把$\omega$称为全局接收时间字(global accepting timed word), 相应地, 我们把$\omega$对应的全局重置时间字$\omega_r$ 称为全

局重置接收时间字(global reset accepting timed word). 一个 DOTA $\mathcal{A}$ 的时间语言 $\mathcal{L}(\mathcal{A})$ 是它的所有全局接收时间字的集合; 相应地, 一个 DOTA $\mathcal{A}$ 的重置时间语言 $\mathcal{L}_r(\mathcal{A})$ 是它的所有全局重置接收时间字的集合. 如果两个 DOTA 的时间语言相同, 那么这两个 DOTA 是等价的.

全局时间字表示从系统外部对系统的观察, 而系统内部的逻辑时间与系统外部的全局时间不同, 我们使用逻辑时间字(logic timed word)表示从系统内部观察到的行为, 逻辑时间字用 $\gamma=(\sigma_1,t_1)(\sigma_2,t_2)...(\sigma_n,t_n)$ 表示, 其中 $t_i\in\mathbb{R}_{\geqslant 0}$ 表示执行动作 $\sigma_i$ 时从系统内部观察到的时钟值 $v(x)$. 逻辑重置时间字(logic reset timed word)在逻辑时间字的基础上扩展了时钟重置信息, 表示为 $\gamma_r=(\sigma_1,t_1,b_1)(\sigma_2,t_2,b_2)...(\sigma_n,t_n,b_n)$, 其中, $b_i\in\mathbb{B}$ 表示发生动作 $\sigma_i$ 时, 时钟 $x$ 是否重置. 我们使用 $\epsilon$ 表示一个空的逻辑时间字, $\epsilon_r$ 表示一个空的逻辑重置时间字.

给出一个逻辑重置时间字 $\gamma_r=(\sigma_1,t_1,b_1)(\sigma_2,t_2,b_2)...(\sigma_n,t_n,b_n)$, 可以计算出其唯一对应的全局重置时间字 $\omega_r=(\sigma_1,\tau_1,b_1)(\sigma_2,\tau_2,b_2)...(\sigma_n,\tau_n,b_n)$, 其中, $\tau_1=u_1$, 当 $i>1$ 时, 如果 $b_{i-1}=\top$, 则 $\tau_i=\tau_{i-1}+u_i$, 如果 $b_{i-1}=\bot$, 则 $\tau_i=\tau_{i-1}+u_i-u_{i-1}$. 如果一个逻辑重置时间字对应的全局重置时间字是一个全局重置接收时间字, 那么称这个逻辑重置时间字是一个逻辑重置接收时间字. 一个单时钟时间自动机的逻辑重置时间语言 $\mathcal{L}_{lr}(\mathcal{A})$ 是该单时钟时间自动机的所有逻辑重置接收时间字的集合.

### 1.3 DOTA的模型学习

给定两个 DOTA $A$ 和 $B$, 如果它们的逻辑重置时间语言相等, 则它们的时间语言相等, 即 $\mathcal{L}_{lr}(A)=\mathcal{L}_{lr}(B)\Rightarrow\mathcal{L}(A)=\mathcal{L}(B)$. 如果它们的时间语言不相等, 那么它们的逻辑重置时间语言必然不相等, 即 $\mathcal{L}(A)\neq\mathcal{L}(B)\Rightarrow\mathcal{L}_{lr}(A)\neq\mathcal{L}_{lr}(B)$. 因此, 学习一个目标时间语言 $\mathcal{L}(A)$ 对应的 DOTA $\mathcal{A}$, 可以构造一个猜想 DOTA $\mathcal{H}$ 满足它们的逻辑重置时间语言相等, 即 $\mathcal{L}_{lr}(\mathcal{H})=\mathcal{L}_{lr}(\mathcal{A})$. 同时对于 $\mathcal{H}$ 和 $\mathcal{A}$ 的等价性判断仍可在时间语言上进行比较, 即当 $\mathcal{L}(\mathcal{H})\neq\mathcal{L}(\mathcal{A})$ 时, 可以推出 $\mathcal{L}_{lr}(\mathcal{H})\neq\mathcal{L}_{lr}(\mathcal{A})$, 此时可以在 $\mathcal{H}$ 和 $\mathcal{A}$ 的逻辑重置时间语言上找到差异作为反例.

DOTA 的学习算法中存在两个角色: 学生和老师, 其中老师掌握了目标 DOTA $\mathcal{A}$ 的时间语言 $\mathcal{L}(\mathcal{A})$, 学生通过向老师提出成员查询和等价查询这两种问题进行学习. 成员查询和等价查询的定义如下.

成员查询 $MQ_{\mathcal{A}}(\gamma)$: 老师收到一个逻辑时间字 $\gamma$ 之后, 首先判断在 DOTA $\mathcal{A}$ 中是否存在一个对应的执行, 如果存在, 则返回一个在 $\mathcal{A}$ 中对应的逻辑重置时间字 $\xi_{\mathcal{A}}(\gamma)$(因为 $\gamma$ 在 $\mathcal{H}$ 和 $\mathcal{A}$ 中可能对应不同的逻辑重置时间字, 为了区分,我们使用 $\xi_{\mathcal{A}}(\gamma)$ 表示 $\gamma$ 在目标自动机 $\mathcal{A}$ 中对应的逻辑重置时间字,后续会使用 $\xi_{\mathcal{H}}(\gamma)$ 表示 $\gamma$ 在假设自动机 $\mathcal{H}$ 中对应的逻辑重置时间字), 以及 $\xi_{\mathcal{A}}(\gamma)$ 是否属于逻辑重置时间语言 $\mathcal{L}_{lr}(\mathcal{A})$ 的结果 $\lambda_{\mathcal{A}}(\gamma)\in\mathbb{B}$. 如果不存在, 说明 $\gamma$ 是不合法的, 即在执行 $\gamma$ 的第 $j$ 个逻辑时间动作时被阻挡, 老师返回一个逻辑重置时间字 $\xi_{\mathcal{A}}(\gamma)$(其中对于所有的 $i\geqslant j$ 位置的重置信息 $b_i=\top$)以及 $\lambda_{\mathcal{A}}(\gamma)=\bot$. 因此成员查询 $MQ_{\mathcal{A}}(\gamma)$ 的结果是一个二元组 $MQ_{\mathcal{A}}(\gamma)=(\xi_{\mathcal{A}}(\gamma),\lambda_{\mathcal{A}}(\gamma))$.

等价查询 $EQ(\mathcal{H})$: 老师收到一个假设 DOTA $\mathcal{H}$ 之后, 回答 $\mathcal{L}(\mathcal{H})=\mathcal{L}(\mathcal{A})$ 是否成立, 如果不成立, 则返回一个逻辑重置时间字 $ctx$ 作为反例.

在 DOTA 的学习算法中, 学生通过成员查询来收集信息, 并将收集到的信息存放在逻辑时间观察表中, 当逻辑时间观察表收集到足够的信息之后, 学生通过操作是得观察表满足相应性质并构造出一个假设 DOTA $\mathcal{H}$, 然后通过等价查询判断假设 DOTA $\mathcal{H}$ 的时间语言是否与目标时间语言相同, 若不相同, 老师会给出反例, 学生通过该反例继续进行学习, 重新构造假设 DOTA, 直到假设 DOTA 的时间语言与目标时间语言相同.

## 2 逻辑时间分类树

文献[13]所述的基于观察表的时间自动机学习算法, 在学习的过程中引入了较多的冗余信息, 因此该学习算法很难对较大规模的时间系统进行学习, 为此我们提出使用逻辑时间分类树代替逻辑时间观察表作为学

习算法的内部存储结构. 本节主要介绍逻辑时间分类树的定义以及相关操作.

### 2.1 逻辑时间分类树的定义

逻辑时间分类树是一种树形数据结构, 包含两种节点, 一种是中间节点, 对应一个逻辑时间字; 另一种是叶节点, 对应一个逻辑重置时间字.在下文中, 在不引起歧义的情况下, 我们将使用中间节点$\gamma$代表对应逻辑时间字$\gamma$的中间节点, 使用叶节点$\gamma_r$代表对应逻辑重置时间字$\gamma_r$的叶节点.逻辑时间分类树的形式化定义如下.

**定义 3(逻辑时间分类树)**. 一个逻辑时间分类树是一个五元组 $\mathcal{T}=(\Sigma,V,L,\boldsymbol{R},\boldsymbol{E})$, 其中,

- $\Sigma$是时间自动机动作的有穷集合, 即字母表;
- $V$ 表示中间节点的有穷集合;
- $L$ 表示叶节点的有穷集合;
- $\boldsymbol{R}\subseteq V\times \mathrm{K}\times\{V\cup L\}$ 表示迁移关系的有穷集合, 其中, K 是布尔向量的有穷集合. 对于任意一个迁移关系 $(u_1,\kappa,u_2)\in\boldsymbol{R}$, 其中, $u_1\in V$ 是一个中间节点, $u_2\in V\cup L$ 是一个中间节点或者叶节点, $\kappa\in\mathrm{K}=\pi(l,u_1)$ 表示成员查询 $MQ_{\mathcal{A}}\left(\Pi_{\{1,2\}}(l)\cdot u_1\right)$ 得到的 $u_1$ 的重置信息以及接收结果 $\lambda_{\mathcal{A}}\left(\Pi_{\{1,2\}}(l)\cdot u_1\right)$, 其中, $l$ 是节点 $u_2$ 下的任意一个叶节点(包括 $u_2$). 函数 $\pi(l,u_1)$ 的计算流程我们稍后进行介绍;
- $\boldsymbol{E}\subseteq L\times\boldsymbol{\Sigma_r}\times L$, 是叶节点迁移的有穷集合, 其中, $\boldsymbol{\Sigma_r}=\Sigma\times\mathbb{R}_{\geqslant 0}\times\mathbb{B}$ 表示逻辑重置时间动作的无穷集合, 对于任意一个叶节点迁移 $(l_1,\boldsymbol{\sigma_r},l_2)$, 其中, $l_1,l_2\in L$, $\boldsymbol{\sigma_r}\in\boldsymbol{\Sigma_r}$, 表示叶节点 $l_1$ 通过动作 $\boldsymbol{\sigma_r}$ 迁移到叶节点 $l_2$;

对于一个逻辑重置时间字$\gamma_r$和一个逻辑时间字 $d$, 函数 $\pi(\gamma_r,d)$ 表示成员查询 $MQ_{\mathcal{A}}\left(\Pi_{\{1,2\}}(\gamma_r)\cdot d\right)$ 得到的 $d$ 的重置信息以及接收结果 $\lambda_{\mathcal{A}}\left(\Pi_{\{1,2\}}(\gamma_r)\cdot d\right)$. 函数 $\pi(\gamma_r,d)$ 的结果是一个布尔向量$\kappa$, 可以通过如下方式获得.

(1) 根据定义, $\Pi_{\{1,2\}}(\gamma_r)$ 代表$\gamma_r$ 对应的逻辑时间字.我们对逻辑时间字 $\Pi_{\{1,2\}}(\gamma_r)\cdot d$ 进行成员查询 $MQ_{\mathcal{A}}\left(\Pi_{\{1,2\}}(\gamma_r)\cdot d\right)$, 结果会得到一个逻辑重置时间字 $\xi_{\mathcal{A}}\left(\Pi_{\{1,2\}}(\gamma_r)\cdot d\right)=\gamma_r\cdot d_r$, 其中, $d_r=(\sigma_1,t_1,b_1)(\sigma_2,t_2,b_2)...(\sigma_n,t_n,b_n)$, $b_i\in\mathbb{B}$ 表示时钟重置信息, 其次, 我们还会得到一个 $\gamma_r\cdot d_r$ 是否接收的信息 $\lambda\left(\Pi_{\{1,2\}}(\gamma_r)\cdot d\right)=b_{n+1}\in\mathbb{B}$.

(2) 布尔向量 $\kappa=(b_1,b_2,...,b_n,b_{n+1})$ 即为函数 $\pi(\gamma_r,d)$ 的结果.为了区分重置和接收信息, 在后文中, 我们会为 $b_{n+1}$ 对应的布尔值加上一个上标 $a$, 即使用 $b_{n+1}=\top^a$ 表示接收, $b_{n+1}=\bot^a$ 表示不接收.

对于逻辑时间分类树中的任意一个中间节点 $d$ 以及从节点 $d$ 的不同分支到达的两个叶节点 $l_1$ 和 $l_2$, 我们称中间节点 $d$ 对这两个叶节点进行区分, 即 $\pi(l_1,d)\neq\pi(l_2,d)$.

### 2.2 筛分操作

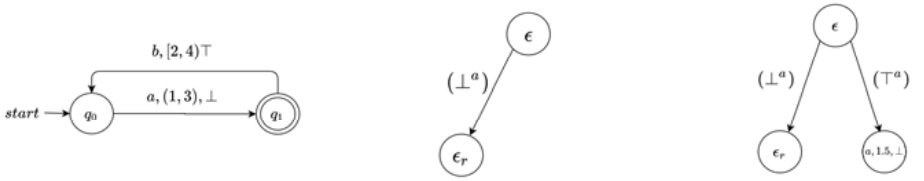
筛分操作 *sift* 又称分类操作, 是逻辑时间分类树中的一个重要操作. 给定一个逻辑重置时间字$\gamma_r$, $sift(\gamma_r)$ 会返回逻辑时间分类树中一个与之对应的叶节点. 筛分操作 $sift(\gamma_r)$ 按照如下过程执行.

1. 将分类树的根节点设置为当前节点.
2. 把当前节点用 $F$ 中间节点 $d$ 来表示, 之后, 根据函数 $\pi(\gamma_r,d)$ 得到一个布尔向量$\kappa$. 然后根据父节点 $d$ 和布尔向量$\kappa$从分类树的迁移集合 $\boldsymbol{R}$ 中寻找对应的子节点, 再把子节点设为当前节点. 重复进行此步骤, 直到当前节点是一个叶节点或者是空节点.
3. 如果当前节点是一个叶节点, 则将此叶节点返回, 筛分操作结束.
4. 如果当前节点是一个空节点, 则说明分类树中没有与$\gamma_r$ 对应的叶节点, 则在该位置生成一个新的叶节点$\gamma_r$, 并构建出相应的迁移关系 $(d,\kappa,\gamma_r)$, 其中, $\kappa=\pi(\gamma_r,d)$, 再将新的叶节点$\gamma_r$返回.

**定理 1(逻辑时间分类树的确定性)**. 给出一个逻辑重置时间字$\gamma_r$, 通过筛分操作 $sift(\gamma_r)$只能到达唯一的一个叶节点.

证明: 假设存在一个逻辑重置时间字$\gamma_r$, 且 $sift(\gamma_r)$可以到达不同的叶节点, 那么一定存在一个中间节点 $d$, 函数 $\pi(\gamma_r,d)$ 会获得两种结果, 也就是说对逻辑时间字 $\Pi_{\{1,2\}}(\gamma_r)\cdot d$ 进行成员查询会获得两个结果, 显然这是不成立的. 因此, 给出一个逻辑重置时间字$\gamma_r$, 通过筛分操作 $sift(\gamma_r)$只能到达唯一的一个叶节点. □

图 1 给出了一个 *sift* 操作的例子, 对于图 1(a)所示的 DOTA $A$ 和图 1(b)所示的分类树$\mathcal{T}$. 接下来我们以逻辑重置时间字 $(a,1.0,\top)$ 和 $(a,1.5,\perp)$ 为例对 *sift* 操作进行说明. 对于逻辑重置时间字 $(a,1.0,\top)$, 我们首先在根节点$\epsilon$处进行 $\pi((a,1.0,\top),\epsilon)$ 操作得到结果$\perp^a$, 然后向下迁移到$\epsilon$的子节点$\epsilon_r$, 因为$\epsilon_r$ 是一个叶节点, 所以将$\epsilon_r$ 作为结果返回, 即 $sift((a,1.0,\top))=\epsilon_r$. 对于逻辑重置时间字 $(a,1.5,\perp)$, 我们在根节点$\epsilon$处进行 $\pi((a,1.5,\perp),\epsilon)$ 操作得到的结果是$\top^a$, 之后向下迁移到了一个空节点, 因此我们创建新的叶节点$(a,1.5,\perp)$, 从而分类树$\mathcal{T}$转化成了新的分类树$\mathcal{T}'$, 如图 1(c)所示, 因此 $sift((a,1.5,\perp))=(a,1.5,\perp)$.




(a) $\mathcal{A}$ (b) $\mathcal{T}$ (c) $\mathcal{T}'$

图 1 筛分操作示例

### 2.3 叶节点迁移构建

为了避免每次构建假设自动机都重新计算相应的迁移, 我们在分类树中增加叶节点迁移这一元素. 在分类树中, 叶节点对应假设 DOTA 的位置, 叶节点迁移对应假设 DOTA 的相应迁移. 本节将介绍逻辑时间分类树中叶节点迁移的构建算法.

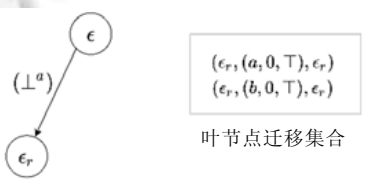
对于逻辑时间分类树中的叶节点$\gamma_r$ 以及逻辑时间动作$(\sigma,t)$. 我们用 $tran(\gamma_r,(\sigma,t))$ 来表示从叶节点$\gamma_r$ 出发, 执行逻辑时间动作$(\sigma,t)$的迁移. $tran(\gamma_r,(\sigma,t))$ 的计算方式如下.

(1) 对逻辑时间字 $\Pi_{\{1,2\}}(\gamma_r)\cdot(\sigma,t)$进行成员查询, 得到一个逻辑重置时间字 $\gamma_r\cdot(\sigma,t,b)$, 其中, $b\in\mathbb{B}$, 表示执行逻辑时间动作$(\sigma,t)$时是否发生时钟重置.

(2) 根据筛分操作得到目标叶节点 $sift(\gamma_r\cdot(\sigma,t,b))$, 即得到叶节点迁移 $tran(\gamma_r,(\sigma,t))=(\gamma_r,(\sigma,t,b),sift(\gamma_r\cdot(\sigma,t,b)))$.

新节点迁移构建操作 $nto(l)$: 每当逻辑时间分类树上增加一个新的叶节点 $l\in L$ 时, 都需要根据字母表$\Sigma$构建从 $l$ 出发的叶节点迁移, 即对于字母表$\Sigma$中的任意一个动作$\sigma\in\Sigma$, 构造出一个对应的逻辑时间动作$(\sigma,0)$, 然后按照上述叶节点迁移构建的方法, 构建出从叶节点 $l$ 出发执行逻辑时间动作$(\sigma,0)$的叶节点迁移. 我们使用 $nto(l)$表示上述操作.

如图 2 所示, 我们给出了一个叶节点迁移构建的例子. 当我们在分类树中新增了一个叶节点$\epsilon_r$时, 需要对其进行 $nto(\epsilon_r)$ 操作, 具体来说, 字母表$\Sigma$中存在 $a$ 和 $b$ 两个动作, 对于动作 $a$, 创建相应的叶节点迁移 $tran(\epsilon_r,(a,0))=(\epsilon_r,(a,0,\top),\epsilon_r)$, 同样对于动作 $b$, 我们也创建了对应的叶节点迁移 $tran(\epsilon_r,(b,0))=(\epsilon_r,(b,0,\top),\epsilon_r)$, 如图 2(b)所示.




(a) $\mathcal{T}$ (b) $E$

图 2 叶节点迁移构建示例

### 2.4 叶节点分裂

在逻辑时间分类树中, 有两种方式可以创建新的叶节点, 一种是对逻辑重置时间字$\gamma_r$进行筛分操作并最终到达一个空节点, 此时创建一个新的叶节点$\gamma_r$; 另外一种是本节将引入的叶节点分裂.

假设逻辑时间分类树$\mathcal{T}$中存在一个叶节点$\gamma_r$, 且存在一个逻辑重置时间字$\beta_r$满足$\gamma_r$=*sift*($\beta_r$), 也就是说, 在分类树$\mathcal{T}$中, $\gamma_r$和$\beta_r$被分类到同一个叶节点$\gamma_r$上. 此时如果存在一个逻辑时间字 $d$, 满足 $\pi(\gamma_r,d)\neq\pi(\beta_r,d)$, 则说明 $d$ 可以对$\gamma_r$ 和$\beta_r$ 进行区分. 当出现这种情况时, 需要对分类树$\mathcal{T}$的叶节点$\gamma_r$ 进行分裂以得到新的叶节点, 我们用 *split*($\gamma_r$,$\beta_r$,$d$)来表示对叶节点$\gamma_r$进行分裂, *split*($\gamma_r$,$\beta_r$,$d$)操作按照如下方式进行操作.

1. 用 $d_0$ 表示叶节点$\gamma_r$的父节点, 相应的迁移关系为 $(d_0,\kappa,\gamma_r)$.
2. 新建一个中间节点 $d$ 代替$\gamma_r$ 作为 $d_0$ 的子节点, 将迁移关系 $(d_0,\kappa,\gamma_r)$ 改为 $(d_0,\kappa,d)$.
3. 将叶节点$\gamma_r$作为 $d$ 的子节点, 并构建对应的迁移关系 $(d,\pi(\gamma_r,d),\gamma_r)$.
4. 新建叶节点$\beta_r$, 将其作为中间节点 $d$ 的子节点, 并构建对应的迁移关系 $(d,\pi(\beta_r,d),\beta_r)$, 此时叶节点$\gamma_r$分裂成了叶节点$\gamma_r$和$\beta_r$.
5. 在叶节点$\gamma_r$ 分裂之后, 分类树$\mathcal{T}$中的叶节点迁移也需要进行相应的修改. 对于$\mathcal{T}$中所有指向叶节点$\gamma_r$的叶节点迁移 $(u,\boldsymbol{\sigma}_r,\gamma_r)\in\boldsymbol{E}$, $u\in L$, $\boldsymbol{\sigma}_r\in\boldsymbol{\Sigma}_r$, 都需要重新计算目标叶节点, 转变为 $(u,\boldsymbol{\sigma}_r,sift(u\cdot\boldsymbol{\sigma}_r))$.

此时在逻辑时间分类树$\mathcal{T}$中, 叶节点$\gamma_r$分裂成两个叶节点$\gamma_r$和$\beta_r$, 并可通过中间节点$d$区分$\gamma_r$和$\beta_r$且原来指向$\gamma_r$的叶节点迁移都进行了相应的修改.

## 3 改进的 DOTA 学习算法

基于第 2 节提出的逻辑时间分类树, 我们设计了改进的 DOTA 学习算法, 见算法 1. 在该算法中, 学生首先构建一个初始化的逻辑时间分类树, 然后根据初始化的分类树构建出一个对应的假设. 此时通过等价查询判断此假设和目标自动机$\mathcal{A}$是否等价, 如果两者等价, 则说明假设是正确的, 算法终止. 否则, 老师会给出一个逻辑重置时间字作为反例 *ctx*, 学生根据 *ctx* 对分类树进行相应的修改, 直至构建出新的假设. 在反例处理过程中, 一个反例可能携带多个可以对分类树进行修改的信息, 但是我们的处理算法只对反例中出现的第一个信息进行处理, 这会丢失一些信息, 从而导致等价查询次数的增加. 为了有效地利用反例信息, 减少等价查询的次数, 我们在进行等价查询之前, 首先判断反例是否可以复用, 即判断上次等价查询产生的反例 *ctx* 是否可以作为新假设的反例. 如果仍是反例, 直接使用上次等价查询产生的反例对分类树进行假设修改. 否则, 重新进行等价查询. 重复上述操作, 直到等价查询不再产生反例, 即构建的假设和目标语言等价.

**算法 1**. 改进的 DOTA 学习算法.

输入: 字母表Σ, 成员查询$\mathcal{Q}_s$, 等价查询$\mathcal{E}$;

输出: 假设自动机$\mathcal{H}$.

```
1.  T ← initialize()
2.  H ← buildHypothesis(T)
3.  ctx ← E(H)
4.  While ctx ≠ null  do
5.      T ← processCounterExample(ctx)
6.      H ← buildHypothesis(T)
7.      While MQ_A(Π_{1,2}(ctx)) ≠ MQ_H(Π_{1,2}(ctx))
8.          T ← processCounterExample(ctx)
9.          H ← buildHypothesis(T)
10.     End while
11.     ctx ← E(H)
```

12. **End while**

13. **return** $\mathcal{T}$

### 3.1 逻辑时间分类树的初始化

本节介绍逻辑时间分类树的初始化算法. 假设目标 DOTA 的字母表为Σ, 学生首先创建一个只有根节点的逻辑时间分类树 $\mathcal{T}_1=(\Sigma,V_1,L_1,\boldsymbol{R}_1,\boldsymbol{E}_1)$, 其中, $V_1=\{\epsilon\}$, $L_1$, $\boldsymbol{R}_1$ 和 $\boldsymbol{E}_1$ 都为空集. 然后学生根据筛分操作 $sift(\epsilon_r)$ 将叶节点 $\epsilon_r$ 加入到 $\mathcal{T}_1$ 中(因为此时不存在叶节点, 那么筛分操作一定会生成一个新的叶节点 $\epsilon_r$). 当叶节点 $\epsilon_r$ 创建好之后, 我们通过 $nto(l)$操作创建从 $\epsilon_r$ 出发的叶节点迁移. 在此过程中可能产生新的叶节点, 当不再有新的叶节点生成时, 初始化完成.

我们以图 3(a)描述的 DOTA $A$ 为例进行逻辑时间分类树的初始化操作, 得到了图 3(b)所示的分类树$\mathcal{T}_1$.


(a) 目标 DOTA $A$ (b) 初始化逻辑时间分类树$\mathcal{T}_1$ (c) 假设 DFA $\mathcal{M}_1$ (d) 假设 DOTA $\mathcal{H}_1$

图 3

### 3.2 构造假设DOTA

给定逻辑时间分类树 $\mathcal{T}=(\Sigma,V,L,\boldsymbol{R},\boldsymbol{E})$, 当$\mathcal{T}$初始化完成或者根据反例修改完成之后, 我们可以根据$\mathcal{T}$构造假设 DOTA $\mathcal{H}$. 在构建$\mathcal{H}$时, 我们首先通过如下转换, 构造$\mathcal{T}$对应的 DFA $M=(Q_M,q_M^0,\Sigma_M,F_M,E_M)$.

- $Q_M=\{q_l|l\in L\}$, 表示 DFA 的状态的有穷集合;
- $q_M^0=q_{\epsilon_r}$, 其中, $\epsilon_r\in L$, 表示 DFA 的初始状态;
- $\Sigma_M=\{\gamma_r\,|\,(l_1,\gamma_r,l_2)\in\boldsymbol{E}\wedge l_1\in L\wedge l_2\in L\wedge\gamma_r\in\boldsymbol{\Sigma_r}\}$, 是 DFA 的字母表, 对于一个 $\gamma_r\in\Sigma_M$, 表示一个逻辑重置时间字;
- $F_M=\{q_l\,|\,\lambda_{\mathcal{A}}(l)=\top^a,l\in L\}$, 是 DFA 接收状态的集合;
- $E_M=\{(q_{l1},\gamma_r,q_{l2})|q_{l1}\in Q_M\wedge q_{l2}\in Q_M\wedge\gamma_r\in\Sigma_M\}$, 表示 DFA 的迁移集合.

当得到 DFA $M$ 之后, 我们根据文献[17]所提出的划分函数, 划分函数可以将时间点映射成不同的时间区间, 从而将该猜想 DFA $M$ 转换成 DOTA $\mathcal{H}$. 所以通过如上操作, 分类树的叶节点对应于$\mathcal{H}$的一个位置, 叶节点迁移对应于$\mathcal{H}$的相应迁移. 以图 3(b)描述的逻辑时间分类树$\mathcal{T}_1$为例, 我们可构建出图 3(c)所示的假设 DFA $M_1$ 和图 3(d)所示的假设 DOTA $\mathcal{H}_1$.

### 3.3 反例处理

参考成员查询的定义, 给出一个逻辑时间字$\gamma$, 我们用 $MQ_{\mathcal{H}}(\gamma)=(\xi_{\mathcal{H}}(\gamma),\lambda_{\mathcal{H}}(\gamma))$ 表示将逻辑时间字$\gamma$在假设自动机$\mathcal{H}$中进行查询的结果, 其中, $\xi_{\mathcal{H}}(\gamma)$ 表示$\gamma$在假设$\mathcal{H}$中对应的逻辑重置时间字, $\lambda_{\mathcal{H}}(\gamma)\in\mathbb{B}$ 表示 $\xi_{\mathcal{H}}(\gamma)$ 是否属于 $\mathcal{L}_{lr}(\mathcal{H})$.

对于一个反例 $ctx=(\sigma_1,t_1,b_1)(\sigma_2,t_2,b_2)...(\sigma_n,t_n,b_n)$, 我们使用 $ctx[:i]=(\sigma_1,t_1,b_1)(\sigma_2,t_2,b_2)...(\sigma_i,t_i,b_i)$ 表示反例 $ctx$ 第 $i$ 个(包括第 $i$ 个位置)的前缀, 使用 $ctx[i:]=(\sigma_i,t_i,b_i)(\sigma_{i+1},t_{i+1},b_{i+1})...(\sigma_n,t_n,b_n)$ 表示反例 $ctx$ 的第 $i$ 个位置之后(包括第 $i$ 个位置)的后缀, 使用 $ctx[i]=(\sigma_i,t_i,b_i)$ 表示反例 $ctx$ 的第 $i$ 个位置的逻辑重置时间动作. 对于一个 DOTA $M$ 和一个逻辑重置时间字$\gamma_r$, 我们用 $M[\gamma_r]$表示 $M$ 执行$\gamma_r$]到达的位置.

在 DOTA 的模型学习算法中, 我们希望构造出一个与目标 DOTA $\mathcal{A}$具有相同逻辑重置时间语言的假设自动机$\mathcal{H}$. 因此当等价查询给出一个逻辑重置时间字表示的反例 $ctx$ 时, 一定满足 $MQ_{\mathcal{A}}(\Pi_{\{1,2\}}(ctx))\neq$

$MQ_{\mathcal{H}}\left(\Pi_{\{1,2\}}(ctx)\right)$, 此时存在两种情况, 第 1 种情况是反例对应的逻辑时间字 $\Pi_{\{1,2\}}(ctx)$ 在$\mathcal{H}$和$\mathcal{A}$中的重置信息不同, 即 $\xi_{\mathcal{A}}(ctx)\neq\xi_{\mathcal{H}}(ctx)$. 第 2 种情况是反例 *ctx* 在$\mathcal{H}$和$\mathcal{A}$中的接收结果不同, 即 $\lambda_{\mathcal{A}}(ctx)\neq\lambda_{\mathcal{H}}(ctx)$.

为了对反例进行处理, 我们参考了文献[21,22]中对反例进行分解的算法, 该算法已经应用到 DFA, 符号自动机等自动机的模型学习算法中[8,19]. 但因为这些自动机的语言中不包含时钟重置信息, 其反例处理算法仅考虑了反例在假设自动机和目标自动机中的接收结果不同(上述第 2 种情况), 没有考虑到时钟重置信息的影响(上述第 1 种情况). 因此我们对该算法进行相应的扩展, 提出了一种新的反例处理算法, 使其能够对反例中时钟重置的错误也进行处理, 如算法 2 所示.

在反例处理算法中, 对于反例 *ctx* 的每一个下标 $i\in[1,|ctx|]$, 我们依次通过前缀分析(prefixAnalyse)和后缀分析(suffixAnalyse)两个子过程判断 *i* 是否是一个错误下标 *EI*. 当我们通过前缀分析或者后缀分析找到 *EI* 时, 再根据发现错误下标 *EI* 的不同子过程, 采用不同的算法对逻辑时间分类树进行修改.

**前缀分析**. 前缀分析用来判断反例的前缀 $ctx[:i]$ 对应的逻辑时间字 $\Pi_{\{1,2\}}(ctx[:i])$ 在$\mathcal{H}$和$\mathcal{A}$中的对应的逻辑重置时间字是否相同, 如果不相同, 即 $\xi_{\mathcal{A}}\left(\Pi_{\{1,2\}}(ctx[:i])\right)\neq\xi_{\mathcal{H}}\left(\Pi_{\{1,2\}}(ctx[:i])\right)$, 那么我们得到了一个错误下标 *EI*=*i*, 然后根据 *EI* 对分类树进行相应的修改(见反例处理 1). 否则, 我们对下标 *i* 进行后缀分析.

**反例处理 1**. 当前缀分析中发现了错误下标 *EI* 时, 令 $q_u=\mathcal{H}[ctx[:EI-1]]$ 表示假设$\mathcal{H}$执行 $ctx[:EI-1]$ 到达的位置, *u* 是其对应的叶节点. 此时我们在分类树中增加一条从叶节点 *u* 出发执行 *ctx*[*EI*]的叶节点迁移 $(u,ctx[EI],sift(u\cdot ctx[EI]))$, 其中逻辑重置时间动作 *ctx*[*EI*]中包含了正确的时钟重置信息, 且 *ctx*[*EI*]中的时间点可以对时间约束进一步划分. 在此过程中, 如果通过筛分(*sift*)操作生成了新的叶节点 *l*, 我们需要 *nto*(*l*)操作建立相关迁移(对应于算法 2 中的 makeComplete 操作).

**算法 2**. 反例处理算法.

输入: 反例 *ctx*;

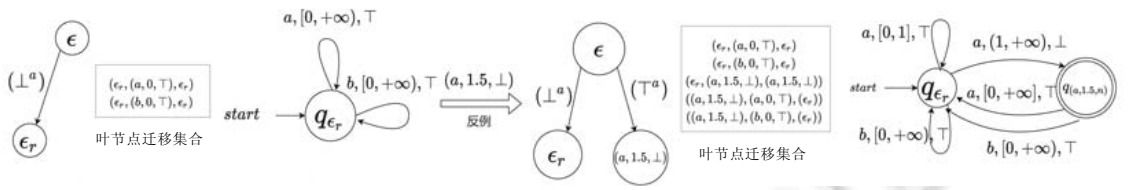
输出: 逻辑时间分类树$\mathcal{T}$.

```
1.  i=1
2.  For  i ≤ |ctx| do
3.      If  prefixAnalyse(ctx,i) = true  then
4.          EI ← i
5.          T.E.add(tran(u,Π_{1,2}(ctx[EI])))
6.          makeComplete(T)
7.          return T
8.      Else if  suffixAnalyse(ctx,i) = true  then
9.          EI ← i
10.         q_u ← H[:EI−1]
11.         q_l ← H[:EI]
12.         u' ← sift(u·ctx[EI])
13.         If  u' ≠ l  then
14.             T.E.add(tran(u,Π_{1,2}(ctx[EI])))
15.         else
16.             T.split(u',u·ctx[EI],ctx[EI+1:])
17.         End if
18.         makeComplete(T)
19.         return T
20.     End if
21.     i=i+1
```

22. **End for**

**前缀分析示例**. 以图 3(a)所示的 DOTA 为目标进行学习, 图 4 给出前缀分析得到错误下标并进行处理的例子. 其中图 4(a)是初始化得到的分类树$\mathcal{T}$, 图 4(b)是其对应的假设$\mathcal{H}$. 当我们对$\mathcal{H}$进行等价查询得到反例$(a,1.5,\bot)$时, 需要对该反例进行前缀分析. 当下标 $i$=1 时, 发现$\xi_{\mathcal{A}}((a,1.5))\neq\xi_{\mathcal{H}}((a,1.5))$, 因此得到错误下标 $EI$=1. 接下来我们根据 $EI$ 对分类树$\mathcal{T}$进行相应的修改, 首先计算出在假设$\mathcal{H}$中执行$(a,1.5,\top)[:EI-1]$到达位置$q_{\epsilon_r}$对应的叶节点为$\epsilon_r$, 然后在$\mathcal{T}$中添加一条新的叶节点迁移$(\epsilon_r,(a,1.5,\top),sift((a,1.5,\top)))$, 又因$sift((a,1.5,\top))$产生了新的叶节点$(a,1.5,\top)$, 我们执行$nto((a,1.5,\top))$操作创建了如下两条叶节点迁移: $((a,1.5,\top),(a,0,\top),\overline{\epsilon_r})$和$((a,1.5,\top),(b,0,\top),\overline{\epsilon_r})$. 通过上述操作, 我们得到了如图 4(c)所示的分类树$\mathcal{T}'$, 并根据$\mathcal{T}'$构建了假设$\mathcal{H}'$.

**后缀分析**. 当对下标 $i$ 进行前缀分析没有发现错误时, 此时 $i$ 满足$\xi_{\mathcal{A}}\left(\Pi_{\{1,2\}}\left(ctx[:i]\right)\right)=\xi_{\mathcal{H}}\left(\Pi_{\{1,2\}}\left(ctx[:i]\right)\right)$, 也就是说逻辑重置时间字$ctx[:i]$在$\mathcal{H}$中是可执行的. 我们用$q_l$表示假设$\mathcal{H}$执行$ctx[:i]$到达的位置, $q_l$对应的叶节点为$l$. 此时$l$与$ctx[:i]$对应于$\mathcal{H}$中的同一个位置. 然后我们为$l$与$ctx[:i]$接上同一个后缀$\Pi_{\{1,2\}}\left(ctx[i+1:]\right)$, 在老师中进行成员查询$MQ_{\mathcal{A}}\left(\Pi_{\{1,2\}}\left(l\cdot ctx[i+1:]\right)\right)$和$MQ_{\mathcal{A}}\left(\Pi_{\{1,2\}}(ctx[:i]\cdot ctx[i+1:]\right)$, 得到不同前缀下的逻辑时间字$\Pi_{\{1,2\}}\left(ctx[i+1:]\right)$的重置信息和接收结果, 分别为$\pi\left(l,\Pi_{\{1,2\}}\left(ctx[i+1:]\right)\right)$和$\pi\left(ctx[:i],\Pi_{\{1,2\}}\left(ctx[i+1:]\right)\right)$. 如果二者不相等, 则说明$\Pi_{\{1,2\}}\left(ctx[i+1:]\right)$可以对逻辑重置时间字 $l$ 或$ctx[:i]$进行区分, 即 $l$ 和$ctx[:i]$应该对应于$\mathcal{H}$中的不同位置. 此时我们得到一个错误下标 $EI$=$i$, 然后根据 $EI$ 对分类树进行相应的修改(见反例处理 2). 否则, 令 $i$=$i$+1, 再次进行前缀分析.




(a) $\mathcal{T}$ (b) $\mathcal{H}$ (c) $\mathcal{T}'$ (d) $\mathcal{H}'$

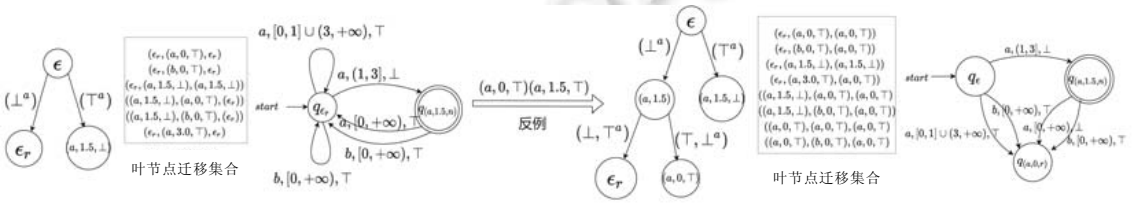
图 4 前缀分析示例

**反例处理 2**. 当通过后缀分析发现错误下标 $EI$=时, 说明在假设$\mathcal{H}$中逻辑重置时间字$ctx[:EI]$和 $l$(参考第 2.1 节对应于逻辑重置时间字)不应该对应于同一个位置. 此时存在两种可能原因(时间区间划分的精度不够或存在未发现的新位置), 需要进一步分析. 令$q_u$表示假设$\mathcal{H}$执行$ctx[:EI-1]$到达的位置, 其对应的叶节点为 $u$. 在分类树中计算叶节点 $u$ 执行动作 $ctx[EI]$到达的叶节点$u'=sift(u\cdot ctx[EI])$.

1. 如果$u'\neq l$, 则在分类树中$u\cdot ctx[EI]$与 $u'$对应于不同的叶节点, 这说明了在$\mathcal{H}$中从位置 $q_u$ 执行 $ctx[EI]$到达位置 $q_l$ 的迁移是一条错误的迁移, 原因是该迁移上时间区间划分的粒度较粗. 对于这种情况, 我们在分类树中添加新的叶节点迁移$(u,ctx[EI],u')$, 其中, $ctx[EI]$包含的时间点可以进一步划分时间区间.
2. 如果$u'=l$, 则在分类树中 $u\cdot ctx[EI]$与 $u'$对应于同一个叶节点, 又从后缀分析可知$\pi(u\cdot ctx[EI],\Pi_{\{1,2\}}\left(ctx[EI+1:]\right))\neq\pi\left(u',\Pi_{\{1,2\}}\left(ctx[EI+1:]\right)\right)$, 这满足了叶节点分裂的条件(也说明了存在一个未发现的新位置). 我们将叶节点 $u'$分裂成叶节点 $u'$和一个新的叶节点 $u\cdot ctx[EI]$, 同时新建一个中间节点$\Pi_{\{1,2\}}\left(ctx[EI+1:]\right)$作为叶节点 $u'$和 $u\cdot ctx[EI]$ $u\cdot ctx[EI]$的父节点, 再对原先指向 $u'$的叶节点迁移重新分配目标叶节点.

同样在反例处理 2 的过程中, 如果通过筛分(*sift*)操作或叶节点分裂操作产生了新的叶节点, 我们需要为其执行 *nto* 操作(对应于算法 2 中的 *makeComplete* 操作).

**后缀分析示例**. 同样以图 3(a)所示的目标 DOTA 进行学习, 图 5 给出后缀分析得到错误下标并进行处理的例子. 其中, 图 5(a)是学习过程中的一个逻辑时间分类树$\mathcal{T}$, 图 5(b)是根据$\mathcal{T}$构建的假设$\mathcal{H}$. 对假设$\mathcal{H}$进行等价查询得到反例$(a,0,\top)(a,1.5,\bot)$. 我们后缀分析发现当 *EI*=1 是一个错误下标: 假设$\mathcal{H}$中执行$(a,0,\top)$到达的位置$q_{\epsilon_r}$对应的叶节点为$\epsilon_r$, 显然$\pi(\epsilon_r,(a,1.5))\neq\pi((a,0,\top),(a,1.5))$. 接下来我们根据 *EI*=1 进一步判断错误原因, 并对分类树$\mathcal{T}$进行相应修改. 根据 $sift((a,0,\top))=\epsilon_r$ 可知在分类树$\mathcal{T}$中$\epsilon_r$和$(a,0,\top)$对应于同一个叶节点, 因此错误原因是存在未发现的位置, 此时我们将叶节点$\epsilon_r$分裂成$\epsilon_r$和$(a,0,\top)$, 它们的父节点为$(a,1.5)$. 然后重新计算指向叶节点$\epsilon_r$的叶节点迁移, 并为叶节点$(a,0,\top)$创建相应的初始叶节点迁移. 我们得到如图 5(c)所示的分类树$\mathcal{T}'$以及图 5(d)所示的假设$\mathcal{H}'$.




(a) $\mathcal{T}$ (b) $\mathcal{H}$ (c) $\mathcal{T}'$ (d) $\mathcal{H}'$

图 5 后缀分析示例

**定理 2**. 给出一个反例, 该反例一定存在一个错误下标可以被反例处理算法发现.

证明: 我们用 *ctx* 表示等价查询给出的反例, 那么 *ctx* 要么满足$\xi_{\mathcal{A}}\left(\Pi_{\{1,2\}}(ctx)\right)\neq\xi_{\mathcal{H}}\left(\Pi_{\{1,2\}}(ctx)\right)$, 要么满足$\lambda_{\mathcal{A}}\left(\Pi_{\{1,2\}}(ctx)\right)\neq\lambda_{\mathcal{H}}\left(\Pi_{\{1,2\}}(ctx)\right)$. 当$\xi_{\mathcal{A}}\left(\Pi_{\{1,2\}}(ctx)\right)\neq\xi_{\mathcal{H}}\left(\Pi_{\{1,2\}}(ctx)\right)$时, 显然通过前缀分析一定能找到一个错误下标. 当$\xi_{\mathcal{A}}\left(\Pi_{\{1,2\}}(ctx)\right)=\xi_{\mathcal{H}}\left(\Pi_{\{1,2\}}(ctx)\right)$时, $ctx$ 必定满足$\lambda_{\mathcal{A}}\left(\Pi_{\{1,2\}}(ctx)\right)\neq\lambda_{\mathcal{H}}\left(\Pi_{\{1,2\}}(ctx)\right)$, 据此可以推出$\pi\left([ctx]_{\mathcal{H}},\epsilon\right)\neq\pi(ctx,\epsilon)$, 也就说明了通过后缀分析一定能找到一个错误下标. 因此给出一个反例, 该反例一定存在一个错误下标可以被反例处理算法发现. □

### 3.4 反例复用

上述针对反例的前缀后缀进行分析的方法, 使得我们可以不用保存反例的所有前缀. 相比基于观察表的反例处理算法, 这种处理算法可以减少大量的冗余信息. 但仍存在一个问题: 由于在反例处理过程中, 只对反例中出现的第 1 个错误下标进行处理, 当某一个反例中存在多个错误下标, 则会丢失一些关键信息, 从而导致等价查询次数的增加. 为了解决此问题, 我们对反例进行了复用. 如前所述, 在对反例进行处理并构建出一个新的假设 DOTA 之后, 将再次判断这个反例是否能作为新构建的假设 DOTA 的反例. 如此以来, 利用反例中所有的有效信息, 可以避免大量冗余信息的处理.

### 3.5 算法分析

给定一个 DOTA $\mathcal{A}$, 令其节点数$|Q|=n$, 字母表大小$|\Sigma|=m$, 时间约束中出现的最大常数为 $g$, 反例的最大长度为 $z$. 考虑到时间自动机的时钟约束的端点值一定是整数, 我们使用符号化状态的概念: 一个 DOTA 的符号化状态是一个二元组$(q,\theta)$, 其中, $q$ 是 DOTA 的一个位置, $\theta$是一个区间, 表示 DOTA 时钟 $x$ 的时钟约束. 其中, $\theta$要么形如$[i,i]$表示一个整数点, 要么形如$(i,i+1)$表示两个相邻整数值之间的区间. 对于 DOTA 的任意一个节点 $q$, 其符号状态共有 $g$+1 个形如$(q,[i,i])$的符号化状态和 $g$+1 个形如$(q,(i,i+1))$的符号化状态, 因此一个 DOTA 最多存在 $2n(g+1)$个符号化状态. 从 DOTA 的任意一个位置出发, 对于每一个动作 $a\in\Sigma$, 迁移的个数最

多为 $2n(g+1)$, 因此一个 DOTA 的迁移的个数最多为 $4mn^2(g+1)^2$.

**定理 3**. 基于逻辑时间分类树的 DOTA 学习算法是可以终止的.

证明: 在基于逻辑时间分类树的学习算法中, 每次对反例进行处理, 都会至少增加一条迁移(增加位置时至少会增加 $m$ 条迁移). 在最坏的情况下, 我们认为每次反例处理只增加一条迁移. 而目标自动机 DOTA 的符号化状态和迁移的个数都是有穷的, 因此本算法一定会终止. □

**定理 4**. 基于逻辑时间分类树的 DOTA 学习算法的等价查询的最坏复杂度为 $\mathcal{O}(mn^2g)$, 成员查询的最坏复杂度为 $\mathcal{O}(mn^3g^3z)$.

证明: 如上所述, 一个 DOTA 的迁移的个数最多为 $4mn^2(g+1)^2$. 因此在最坏情况下, 学生最多会进行 $4mn^2(g+1)^2$ 次等价查询, 所以等级查询的最坏复杂度为 $\mathcal{O}(mn^2g^2)$. 根据等价查询的次数可知, 在学习过程中最多会给出 $4mn^2(g+1)^2$ 个反例, 对于其中的每一个反例, 学生都会对其进行前缀分析和后缀分析, 分析过程产生的成员查询次数最多为 $z$ 个. 在得到错误下标之后, 学生需要对反例进行处理, 此时会使用筛分(*sift*)操作. 因为 DOTA 的符号状态的个数最多为 $2n(g+1)$, 因此在分类树中最多存在 $2n(g+1)$ 个叶节点, 可以推出分类树中最多存在 $2n(g+1)$ 个中间节点, 因此进行一次筛分(*sift*)操作, 学生最多进行 $2n(g+1)$ 次成员查询, 所以成员查询的最坏复杂度为 $\mathcal{O}(mn^3g^3z)$. □

从算法的复杂度来看, 虽然我们提出的基于逻辑时间分类树的学习算法与基于逻辑时间观察表的学习算法具有相同的最坏时间复杂度, 但是因为我们提出的算法可以很大程度地减少内存空间, 降低成员查询的次数, 同时可以更快地构建自动机. 因此在实际的学习场景中, 我们提出的学习算法的具有更好的性能.

## 4 学习算法示例

为了更清楚地描述我们提出的学习算法, 本节将以图 3(a)中的 DOTA $\mathcal{A}$为例进行学习, $\mathcal{A}$的初始位置为 $q_0$, 接收位置为 $q_1$, 字母表为 $\Sigma=\{a,b\}$. 图 6 给出了本次学习过程中逻辑时间分类树的变化情况, 图 7 则描述了相应的假设 DFA 和 DOTA 的变化情况.

学习算法的第 1 步是逻辑时间分类树的初始化, 学生通过初始化操作, 得到了图 6(a)所示的初始分类树 $\mathcal{T}_1$, $\mathcal{T}_1$ 只有两个节点: 根节点 $\epsilon$ 和叶节点 $\epsilon_r$, $\mathcal{T}_1$ 的叶节点迁移集合中包含两条叶节点迁移. 接着, 我们根据$\mathcal{T}_1$, 构建假设 DFA $\mathcal{M}_1$(如图 7(a)所示), 以及对应的假设 DOTA $\mathcal{H}_1$(如图 7(b)所示). 然后通过等价查询询问假设 $\mathcal{H}_1$ 和目标 DOTA $\mathcal{A}$是否等价. 当得到反例 $(a,1.5,\bot)$ 时, 我们通过前缀分析中获得错误下标 $EI$=1, 因此在分类树中添加相关的迁移 $(\epsilon_r,(a,1.5,\bot),sift(a,1.5,\bot))$. 该过程的筛分操作 $sift(a,1.5,\bot)$ 产生了新的叶节点 $(a,1.5,\bot)$, 从而需要构建新的叶节点的初始叶节点迁移. 在完成 $nto((a,1.5,\bot))$ 操作后, 得到分类树$\mathcal{T}_2$(如图 6(b)所示). 再根据$\mathcal{T}_2$ 构建了相应的假设 DFA $\mathcal{M}_2$(如图 7(c)所示)和假设$\mathcal{H}_2$(如图 7(d)所示). 再次对$\mathcal{H}_2$ 进行等价查询, 得到反例 $(a,3.0,\top)$并 进行分析. 在前缀分析中发现错误下标 $EI$=1, 因此在分类树中增加新的叶节点迁移 $(\epsilon_r,(a,3.0,\top),sift(a,3.0,\top))$, 进而得到新的分类树$\mathcal{T}_3$(如图 6(c)所示), 并根据$\mathcal{T}_3$ 构建了假设 DFA $\mathcal{M}_3$(如图 7(e)所示)和假设$\mathcal{H}_3$(如图 7(f)所示). 继续对$\mathcal{H}_3$ 进行等价查询, 得到反例 $(a,0.0,\top)(a,1.5,\top)$ 并在后缀分析中发现了错误下标 $EI$=1. 进一步分析发现在$\mathcal{T}_3$ 中叶节点 $\epsilon_r$ 需要分裂, 因此将叶节点 $\epsilon_r$ 分裂为叶节点 $\epsilon_r$ 以及一个新的叶节点 $(a,0.0,\top)$, 并增加新的中间节点 $(a,1.5)$, 叶节点 $\epsilon_r$ 以及 $(a,0.0,\top)$ 都是中间节点 $(a,1.5)$ 的子节点. 通过对叶节点 $(a,0.0,\top)$进行 $nto(a,0.0,\top)$. 操作, 得到分类树$\mathcal{T}_4$(如图 6(d)所示), 并构建了相应的 DFA $\mathcal{M}_4$(如图 7(g)所示)和假设$\mathcal{H}_4$(如图 7(h)所示). 对$\mathcal{H}_4$ 继行等价查询, 得到新的反例 $(a,1.5,\bot)(b,2.0,\top)(a,1.5,\bot)$, 并通过后缀分析发现错误下标 $EI$=2, 进一步分析发现是由于迁移出错, 因此学生在分类树的叶节点迁移集合中增加相关的迁移 $((a,1.5,\bot),(b,2.0,\top),sift((a,1.5,\bot),(b,2.0,\top)))$. 由此得到了分类树$\mathcal{T}_5$(如图 6(e)所示), 对应的 DFA 为$\mathcal{M}_5$

(如图 7(i)所示), DOTA 为$\mathcal{H}_5$(如图 7(j)所示). 学习者通过对$\mathcal{H}_5$ 进行等价查询, 再次得到一个反例 $(a,1.5,\perp)(b,4.0,\top)(a,1.5,\perp)$. 对该反例进行分析, 仍然是后缀分析发现了 *EI*=2, 原因同样是迁移出错, 因此学生在分类树中增加相应的叶节点迁移$\big((a,1.5,\perp),(b,4,\top),(a,0,\top)\big)$, 得到了逻辑时间分类树$\mathcal{T}_6$(如图 6(f)所示), 生成对应的假设 DFA $\mathcal{M}_6$(如图 7(k)所示)和 COTA $\mathcal{H}_6$(如图 7(l)所示). 对$\mathcal{H}_6$进行等价查询, 此时没有反例产生, 即 $\mathcal{L}(\mathcal{H}_6)=\mathcal{L}(\mathcal{A})$, 说明了$\mathcal{H}_6$和 DOTA $\mathcal{A}$等价, 学习流程结束.

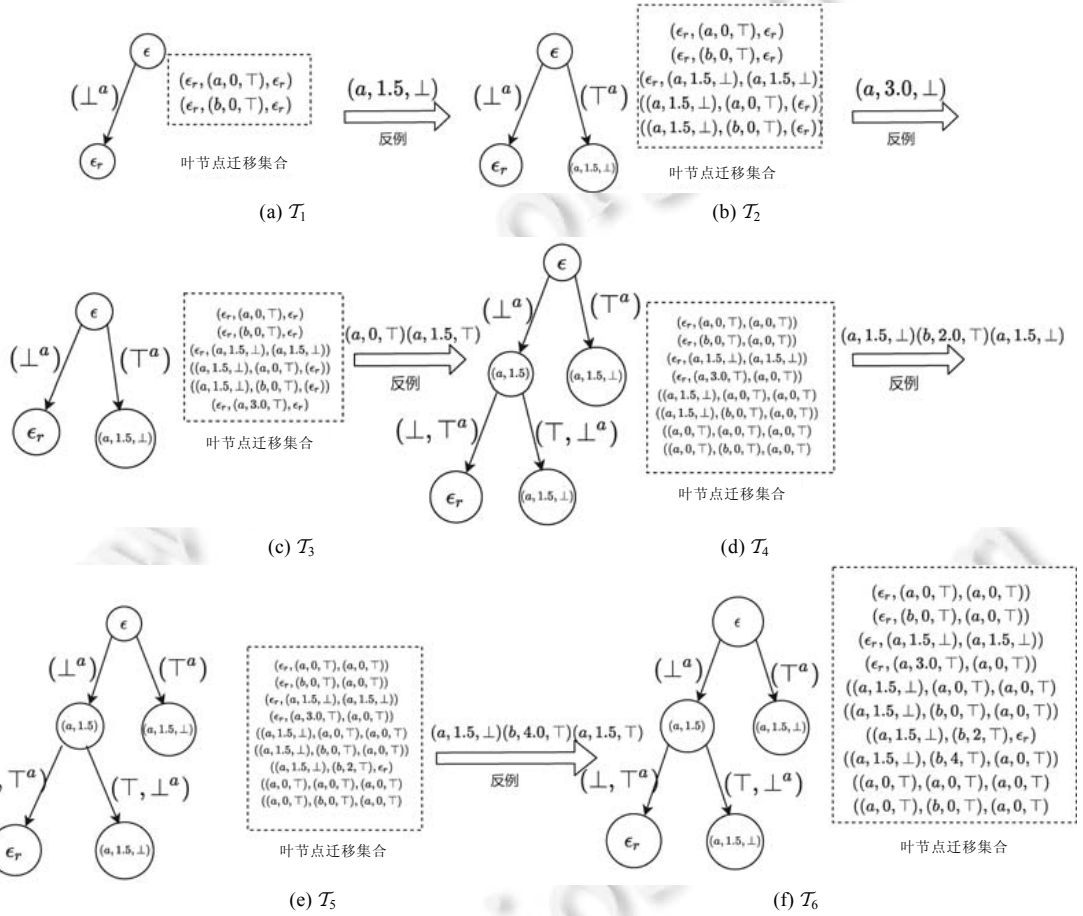


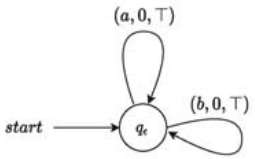

图 6 学习示例产生的分类树

## 5 实验与结果分析

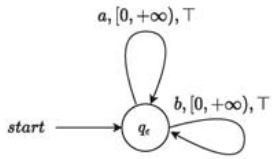
我们实现了本文提出的改进的 DOTA 学习算法. 为了比较我们的算法和目前基于观察表的算法[13], 针对多组随机生成的 DOTA 模型, 我们分别使用两种算法学习, 记录两种算法的学习性能数据(成员查询次数, 等价查询次数和学习耗时). 所有实验都是在同一台 Surface Laptop2 的 PC 机上进行的, 处理器为 Intel(R) Core (TM) i5-8250U CPU @ 1.60 GHz 1.80 GHz, 运行内存为 8.00 GB RAM.

我们根据 DOTA 的字母表大小, 位置数以及时间区间划分数的不同, 一共随机生成了 13 组 DOTA 模型, 每一组又分别有 15 个 DOTA 模型. 为了避免随机算法产生的极端例子的干扰, 在统计过程中, 我们舍弃了每一组中学习耗时最长和最短的 5 个例子(以改进算法学习耗时为标准), 并对剩下的 5 个 DOTA 模型的学习性能进行统计求平均值. 实验结果见表 1–表 3. 表中的例子编号由 3 个数字组成, 分别表示 DOTA 模型的位置数 *n*, 字母表的大小 *m* 以及 DOTA 时间区间的划分个数 *h*. 表 1–表 3 分别记录了固定 *n*、*m*、*h* 中的两个参数, 对

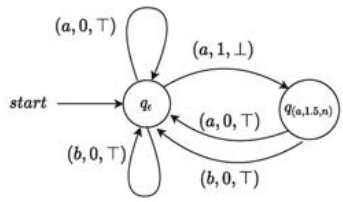
另外一个参数进行改变而得到的实验结果, 以此来观察改变不同变量对于两种学习算法性能的影响. 图 8–图 10 以折线图的方式对表 1–表 3 的数据进行了描述. 其中, 方形点为逻辑时间观察表的算法, 圆形点为逻辑时间分类树改进算法.



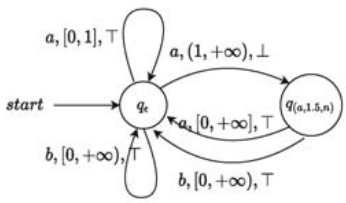

(a) $\mathcal{M}_1$



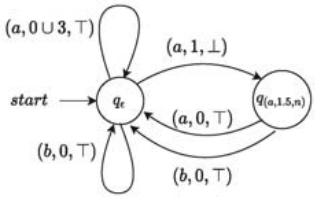

(b) $\mathcal{H}_1$



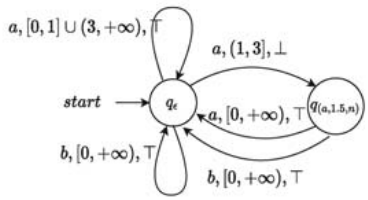

(c) $\mathcal{M}_1$



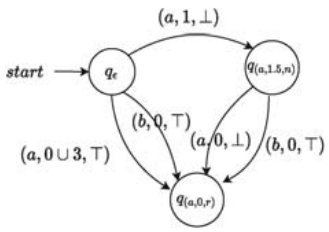

(d) $\mathcal{H}_2$



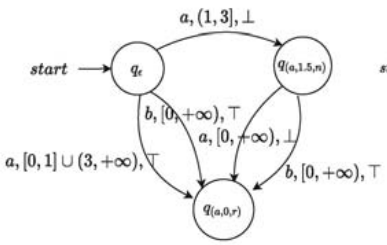

(e) $\mathcal{M}_3$



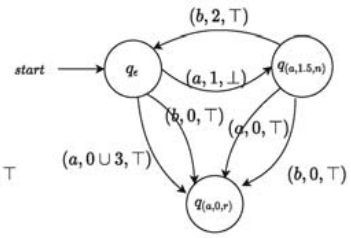

(f) $\mathcal{H}_3$



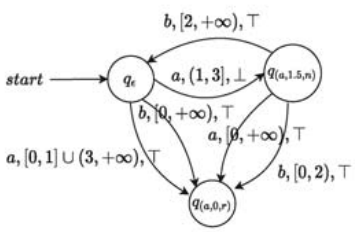

(g) $\mathcal{M}_4$



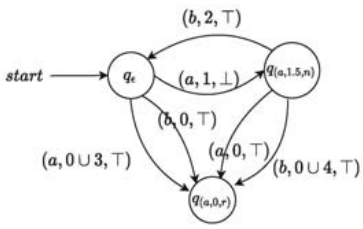

(h) $\mathcal{H}_4$



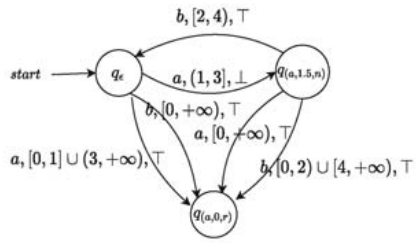

(i) $\mathcal{M}_5$




(j) $\mathcal{H}_5$




(k) $\mathcal{M}_6$




(l) $\mathcal{H}_6$

图 7　学习示例产生的假设 DFA 和假设 DOTA

表 1 改变位置数 $n\in\{5,7,9,11,13\}$且固定字母表大小 $m$=4 和时间区间划分数 $h$=3 的学习结果

| 例子编号 | 逻辑时间观察表 | | | 逻辑时间分类树 | | |
|---|---|---|---|---|---|---|
| | 成员查询次数 | 等价查询次数 | 耗时(s) | 成员查询次数 | 等价查询次数 | 耗时(s) |
| 4-4-3 | 291.8 | 33 | 0.769 | 152.4 | 30.8 | 0.036 |
| 6-4-3 | 740 | 50.2 | 5.566 | 329.6 | 46.6 | 0.163 |
| 8-4-3 | 1 292.4 | 64.4 | 18.197 | 515.2 | 60.4 | 0.452 |
| 10-4-3 | 1 822.4 | 82.2 | 51.004 | 620.6 | 77.2 | 1.321 |
| 12-4-3 | 2 647.4 | 95.4 | 117.744 | 823.8 | 90.4 | 2.676 |

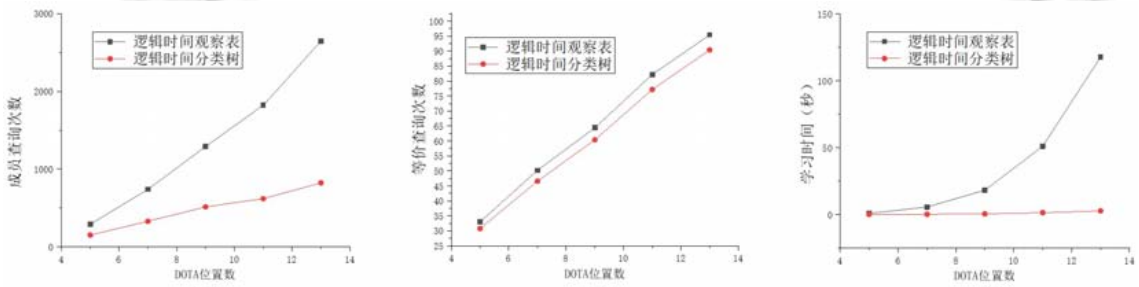
表 2 改变字母表大小 $m\in\{2,3,4,5,6\}$且固定位置数 $n$=9 和时间区间划分数 $h$=3 的学习结果

| 例子编号 | 逻辑时间观察表 | | | 逻辑时间分类树 | | |
|---|---|---|---|---|---|---|
| | 成员查询次数 | 等价查询次数 | 耗时(s) | 成员查询次数 | 等价查询次数 | 耗时(s) |
| 8-2-3 | 454.2 | 26 | 1.282 | 150.6 | 22 | 0.034 |
| 8-3-3 | 897 | 48 | 6.719 | 306 | 44.2 | 0.182 |
| 8-4-3 | 1 292.4 | 64.4 | 18.197 | 515.2 | 60.4 | 0.452 |
| 8-5-3 | 1 724.6 | 80 | 42.828 | 684 | 76.2 | 0.931 |
| 8-6-3 | 1 541.4 | 98.8 | 60.498 | 647.4 | 93.4 | 1.830 |

表 3 改变时间区间划分数 $h\in\{2,3,4,5,6\}$且固定位置数 $n$=9 和字母表大小 $m$=4 的学习结果

| 例子编号 | 逻辑时间观察表 | | | 逻辑时间分类树 | | |
|---|---|---|---|---|---|---|
| | 成员查询次数 | 等价查询次数 | 耗时(s) | 成员查询次数 | 等价查询次数 | 耗时(s) |
| 8-4-2 | 874.4 | 34.6 | 3.574 | 281.2 | 29.8 | 0.068 |
| 8-4-3 | 1 292.4 | 64.4 | 18.197 | 515.2 | 60.4 | 0.452 |
| 8-4-4 | 1 446.2 | 96.6 | 28.434 | 550 | 93.4 | 1.897 |
| 8-4-5 | 1737 | 127.8 | 72.614 | 682.2 | 126.4 | 5.356 |
| 8-4-6 | 2 278.8 | 158 | 144.210 | 894.4 | 155.4 | 13.328 |

图 8 以折线图的形式展示了表 1 的数据, 描述了随着 DOTA 位置数增加, 两种算法的性能表现.




(a) (b) (c)

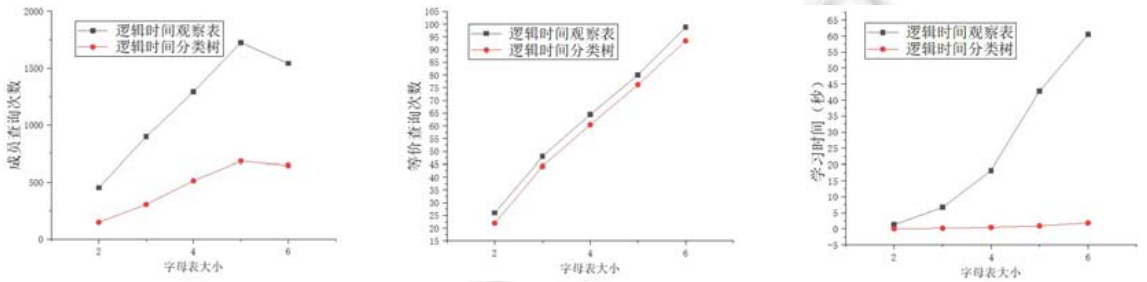
图 8 随 DOTA 的位置数增加, 两种学习算法性能数据的变化趋势

图 8(a)描述了两种学习算法的成员查询次数与 DOTA 的节点数的关系, 可以看出成员查询数量都随着 DOTA 的位置数的增长而增长. 但是改进算法的成员查询次数要比原有算法要低得多, 仅为原有算法的三分之一左右. 图 8(b)描述了两种算法的等价查询数量随着位置数增加的变化趋势, 两种算法的等价查询趋势基本相同, 其中改进算法的等价查询次数有稍许降低. 这是因为我们通过反例复用减少了改进算法等价查询的数量. 图 8(c)描述了学习时间随着位置数增加而产生的变化, 可以看出原有算法的学习时间随着位置数的增加增长的很快, 而改进算法的学习时间始终维持在一个较低的且平稳的水平. 在位置数达到了 12 个的时候, 改进算法只需要 2.676 s 就可以学习成功, 而原有算法则需要 117.744 s, 改进算法学习速度提高了 44 倍.

图 9 描述了随着 DOTA 字母表大小的增加, 两种算法的性能表现. 从图 9(a)中可以看出, 随着字母表大小的增加, 成员查询总体上是逐渐增加的. 但是在字母表大小达到 6 个的时候, 原有算法的成员查询有一个降低的趋势, 经过对相关模型进行分析, 我们发现对该组 DOTA 模型进行学习时, 等价查询产生的反例长度较短, 相应地反例前缀的数量就较少, 因此原有算法的成员查询数量得到了大幅度降低. 而反例长度对于改进算法的影响较小, 所以其成员查询的数量保持了一个平稳的趋势. 图 9(b)描述了等价查询次数随着字母表大小变

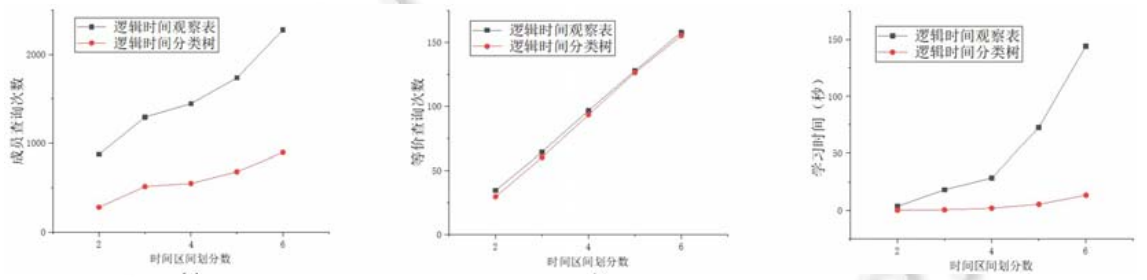
化的趋势, 显然改进算法的等价查询次数略小于基于观察表的学习算法. 图 9(c)描述了学习耗时随着字母表大小的变化而发生的变化, 可以看出改进算法在学习耗时上表现仍然优越.

图 10 描述了随着 DOTA 时间区间划分数的增加, 两种算法的性能表现. 可以看出随着时间区间划分数的增加, 两种算法的成员查询, 等价查询和学习耗时都会随之增加, 但相较于原有算法, 改进算法具有更好的表现, 成员查询的次数仅为原有算法的 35%左右, 等价查询略有降低, 学习耗时大幅度地降低.




(a) (b) (c)

图 9 随 DOTA 字母表大小的增加,两种学习算法性能数据的变化趋势




(a) (b) (c)

图 10 随 DOTA 时间区间划分数的增加, 两种学习算法性能数据的变化趋势

综上所述, 无论是在哪一种情况下, 改进算法的性能表现都有明显提升. 在成员查询方面, 改进算法的成员查询次数仅为原有算法的 35%左右. 降低的主要原因是逻辑时间分类树不需要处理反例的每一个前缀, 尤其是在处理较长反例的时候, 可以避免处理更多的冗余信息. 在等价查询方面, 两种算法的等价查询次数相近, 但是改进算法的等价查询次数总是要低一些, 这是因为在改进算法中使用了反例复用的技术, 使得每一个反例都能够得到有效利用, 从而降低了等价查询次数. 在学习耗时方面, 从实验中可以看出, 在较复杂的 DOTA 的模型学习中, 改进算法的学习速度比原有算法提高了 45 倍. 这种学习速度提升的原因, 除了成员查询和等价查询的优化, 还有一个原因是逻辑时间分类树可以更高效地构建自动机.

## 6 相关工作

模型学习(model learning)通过软件和硬件组件的输入输出观察, 建立状态机模型[23]. 对于实时系统的模型学习, 国际上已有一些比较好的研究结果. 主要分为被动(passive)和主动(active)学习两种形式. 被动学习一般指从给定的输入输出集合中学习出系统的形式模型，该模型只要求与给定的集合行为一致/主动学习一般指学习者可以根据需要主动选择观察系统的输入输出行为. 对于实时自动机(real-timed automata, RTA), Verwer 等人根据状态合并的思想提出了相应的被动学习算法[24]. 在此基础上, Verwer 等人又提出了一种单时钟时间自动机的被动学习算法[25]. 但是, 被动学习不能保证学习出的形式模型的正确性, 因此其无法在实时系统的模型检验中使用. 为了解决这个问题, 在符号自动机的模型学习[6−8]的启发下, 安杰等人通过在 $L^*$算法中引入

了时间信息, 提出了实时观察表的概念. 基于实时观察表, 安杰等人提出了实时自动机的模型学习算法, 并证明算法的正确性以及多项式复杂度. 在此基础上, 安杰等人将实时观察表扩展成逻辑时间观察表, 并提出了现有第 1 个确定性单时钟时间自动机的主动模型学习算法. 在实际系统中, DOTA 的模型学习中的等价查询往往通过测试的方式进行, 很难保证其正确性. 沈炜等人[18]将 PAC 理论应用到 DOTA 的等价查询算法中, 为在实际系统中学习 DOTA 提供了理论基础. 但正如本文所介绍的, 基于观察表的学习总是存在冗余查询的问题, 并且修改观察表的相关操作非常耗时, 随着模型规模的增加尤为明显, 因此本文与前述工作最大的不同之处是, 运用时间分类树取代了观察表, 从而解决了以上两个问题. 对于其他实时系统形式模型的学习, 也已有一些工作. Grinchtein 等人提出了事件记录自动机(event-recording automata)的主动学习[26]. 进一步, Henry 等人考虑了一种特殊形式的事件记录自动机的主动模型学习[27]. 最近, Vaandrager 等人提出了对于带一个定时器的 Mealy 机(mealy machine with one timer)的主动模型学习算法[28].

基于分类树结构的自动机学习, 最早由 Isberner[19]在其 TTT 算法中提出, 其依旧遵循着 $L^*$框架下的学习者-老师模型来学习一个 DFA, 不同的是, 其中的学习者使用一种分类树的结构代替了观察表结构存储从老师得到的信息, 并实验证明了分类树的结构相比于观察表具有更高的反例处理效率和更少的成员查询次数, 从而拥有显著高于观察表的学习速度使得其可以适用于学习更大的模型. Argyros 基于 Isberner 的研究拓展了 TTT 的树形结构提出针对符号自动机的模型学习算法, 实验证明其仍然具有优秀的表现. 本文将其运用到时间自动机的学习, 证明了其在时间自动机的模型学习中的有效性.

## 7 总结和展望

针对目前 DOTA 学习的效率问题, 本文提出了逻辑时间分类树, 并设计基于逻辑时间分类树的 DOTA 的模型学习算法, 进而代码实现. 通过相关实验, 我们说明了该算法的优越性, 在成员查询, 等价查询以及学习时间方面, 改进算法都有比较好的效果. 未来工作可以探讨多个时钟的确定性时间自动机的学习算法.

**References**:


[1] Angluin D. Learning regular sets from queries and counterexamples. Information & Computation, 1987, 75(2): 87−106.

[2] Aarts F, Kuppens H, Tretmans GJ, *et al*. Improving active mealy machine learning for protocol conformance testing. Machine Learning, 2014, 96(1-2): 189−224.

[3] Shahbaz M, Groz R. Inferring mealy machines. In: Proc. of the 2nd World Congress on Formal Methods. 2009. 207−222.

[4] Isberner M, Howar F, Steffen B. The open-source learnlib—A framework for active automata learning. In: Proc. of the 27th Int'l Conf. on Computer Aided Verification (CAV). 2015. 487−495.

[5] Bollig B, Habermehl P, Kern C, *et al*. Angluin-style learning of NFA. In: Proc. of the 21st Int'l Joint Conf. on Artificial Intelligence (IJCAI). 2009. 1004−1009.

[6] Drews S, D'Antoni L. Learning symbolic automata. In: Proc. of the 23rd Int'l Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS). 2017. 173−189.

[7] Maler O, Mens I. Learning regular languages over large alphabets. In: Proc. of the 20th Int'l Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS). 2014. 485−499.

[8] Argyros G, D'Ani L. The learnability of symbolic automata. In: Proc. of the 30th Int'l Conf. on Computer Aided Verification (CAV). 2018. 427−445.

[9] Howar F, Steffen B, Jonsson B, *et al*. Inferring canonical register automata. In: Proc. of the 13th Int'l Conf. on Verification, Model Checking, and Abstract Interpretation (VMCAI). 2012. 251−266.

[10] Cassel S, Howar F, Jonsson B, *et al*. Active learning for extended finite state machines. Formal Aspects of Computing, 2016, 28(2): 233−263.

[11] Aarts F, Fiterau-Brostean P, Kuppens H, *et al*. Learning register automata with fresh value generation. In: Proc. of the 12th Int'l Colloquium on Theoretical Aspects of Computing (ICTAC). 2015. 165−183.

[12] Farzan A, Chen Y, Clarke EM, *et al*. Extending automated compositional verification to the full class of omega-regular languages. In: Proc. of the Int'l Conf. on Tools & Algorithms for the Construction & Analysis of Systems. 2009.

[13] Li Y, Chen YF, Zhang L, *et al*. A novel learning algorithm for Büchi automata based on family of DFAs and classification trees. Information and Computation, 2020(2), 104678.

[14] Tappler M, Aichernig BK, Bacci G, *et al*. L*-based learning of markov decision processes (Extended Version). In: Proc. of the Formal Methods-The Next 30 Years-Third World Congress (FM 2019). LNCS 11800, 2019. 651−669.

[15] Wang J, Zhan N, Feng X, *et al*. Overview of formal methods. Ruan Jian Xue Bao/Journal of Software, 2019, 30(1): 33−61 (in Chinese with English abstract). http://www.jos.org.cn/9825-1000/5652.htm [doi: 10.13328/j.cnki.jos.005652].

[16] An J, Wang L, Zhan B, *et al*. Learning real-time automata. SCIENCE CHINA Information Sciences, 2020, 10.
[17] An J, Chen M, Zhan B, *et al*. Learning one-clock timed automata. In: Proc. of the Int'l Conf. on Tools and Algorithms for the Construction and Analysis of Systems. Cham: Springer, 2020. 444−462.
[18] Shen W, An J, Zhan B, *et al*. PAC learning of deterministic one-clock timed automata. In: Proc. of the Int'l Conf. on Formal Engineering Methods. Cham: Springer, 2020. 129−146.
[19] Isberner M, Howar F, Steffen B. The TTT algorithm: A redundancy-free approach to active automata learning. In: Proc. of the 5th Int'l Conf. on Runtime Verification (RV). 2014. 307−322.
[20] Grinchtein O, Jonsson B, Pettersson P. Inference of eventrecording automata using timed decision trees. In: Proc. of the Int'l Conf. on Concur-concurrency Theory. Berlin, Heidelberg: Springer, 2006.
[21] Rivest RL, Schapire RE. Inference of finite automata using homing sequences. Information & Computation, 1993, 103(2): 299−347.
[22] Argyros G, Stais I, Kiayias A, *et al*. Back in black: Towards formal, black box analysis of sanitizers and filters. In: Proc. of the 2016 IEEE Symp. on Security and Privacy (SP). IEEE, 2016.
[23] Vaandrager FW. Model learning. Communications of the ACM, 2017, 60(2): 86−95.
[24] Verwer S, de Weerdt M, Witteveen C. Efficiently identifying deterministic real-time automata from labeled data. Machine Learning, 2012, 86(3): 295−333.
[25] Verwer S, de Weerdt M, Witteveen C. The efficiency of identifying timed automata and the power of clocks. Information & Computation, 2011, 209(3): 606−625.
[26] Grinchtein O, Jonsson B, Leucker M. Learning of event-recording automata. Theoretical Computer Science, 2010, 411(47): 4029−4054.
[27] Henry L, Jéron T, Markey N. Active learning of timed automata with unobservable resets. In: Proc. of the 18th Int'l Conf. on Formal Modeling and Analysis of Timed Systems (FORMATS). 2020. 144−160.
[28] Vaandrager FW, Bloem R, Ebrahimi M. Learning mealy machines with one timer. In: Proc. of the 15th Int'l Conf. on Language and Automata Theory and Applications (LATA). 2021. 157−170.

**附中文参考文献**:

[15] 王戟, 詹乃军, 冯新宇, 刘志明. 形式化方法概貌. 软件学报, 2019, 30(1): 33−61. http://www.jos.org.cn/1000-9825/5652.htm [doi: 10.13328/j.cnki.jos.005652]

**米钧日**(1995－), 男, 硕士生, 主要研究领域为组合验证, 时间自动机理论, 模型学习.

**安杰**(1990－), 男, 博士, 主要研究领域为形式化方法, 机器学习, 模型学习, 系统识别.

**张苗苗**(1971－), 女, 博士, 研究员, 博士生导师, 主要研究领域为智能系统学习, 人工智能的形式化验证, 模型学习和验证.

**杜博闻**(1991－), 男, 博士生, 主要研究领域为人工智能, 智能系统, 移动计算与软件工程.