使用 ACL2s 自动评阅自动机

Ankit Kumar、Andrew Walter 和 Panagiotis Manolios东北大学

{ankitk,atwalter,pete}@ccs.neu.edu

几乎所有计算机科学专业都要求学生修读计算理论（ToC）课程，课程涵盖有限自动机、下推自动机和图灵机等各种计算模型。计算理论课程通常布置需要用纸笔作答的作业。批改这类作业需要时间，因此学生通常要在完成作业一周多后才能收到反馈。我们介绍自动机自动检查器（A2C），这是一个开源库，可用于构造可执行的自动机，其定义方式仿照标准教科书中的定义 [28]。借助语义等价性检查、性质和测试用例，这类构造很容易推理。教师可以方便地用自己的构造来指定答案。A2C 能检查学生答案与教师答案在语义上是否等价，并立即生成可供采取行动的反馈，帮助学生更好地理解课程内容。学生可以下载 A2C 并在本地使用，也可以将其集成到 Gradescope 等学习管理系统（LMS）中，以自动评阅学生提交的答案并生成反馈。A2C 基于 ACL2s 交互式定理证明器，该证明器提供了陈述、证明和证伪性质的高级方法。由于反馈是自动生成的，A2C 可以大规模部署，并集成到大规模开放在线课程中。

1 引言

在计算理论（ToC）课程中，学生会学习计算模型，包括确定性有限自动机（确定有限自动机）、下推自动机和图灵机。在构造自动机时，学生可以从即时的自动反馈中受益。我们介绍自动机自动检查器（A2C） [1], ——一个基于 ACL2s 定理证明器 [4, 12, 7]的开源库。为实现上述目的，A2C 提供了便捷的形式，用于定义、执行、测试自动机并对其进行推理。

使用 A2C 定义的自动机不仅可以执行，还是一种可供推理的计算形式模型。教师可以使用 ACL2s 的功能来指定并检查这些模型的性质，也可以运用 ACL2s 定理证明器的全部能力来实现自定义检查。视觉化表示或基于 XML 的表示通常不具备这一优势，例如 Automata Tutor V3 [13] 或 JFLAP [25]中使用的表示。此外，借助功能强大的定理证明器，教师可以测试自己的构造、证明性质，甚至添加新的理论或模型。A2C 内置了对学生提交的自动机进行评分并将结果输出为 JSON 的支持，因此易于集成到现有的学习管理系统中。它开箱即支持 Gradescope，这是一种广泛使用的在线评分系统。

学生向 Gradescope 提交自动机构造后，系统会检查其与教师提供的解答是否一致且在语义上等价。学生会立即收到反馈，例如，

J. Marcos、W. Neuper 和 P. Quaresma（编）：教育软件定理证明组件 2022（ThEdu'22），EPTCS 375，2023年，第 77–91 页，
doi:10.4204/EPTCS.375.7

© Kumar、Walter 和 Manolios 本作品采用知识共享署名许可协议授权。

如果系统判定其解答有误，就会提供一些输入，指出其自动机错误分类了哪些输入。这比 JFLAP [25] 和 OpenFLAP [22]等现有工具有了显著改进，因为这些工具不支持个性化、高质量的反馈。学生可以根据收到的反馈修改并重新提交自动机。

A2C 目前在同一个软件包中支持 3 种计算模型——确定有限自动机、下推自动机和图灵机。这三种模型均支持输入验证、等价性测试和基于属性的测试。与 Automata Tutor V3 或 JFLAP 不同，我们为所有受支持的计算模型提供统一测试接口；而 Automata Tutor V3 和 JFLAP 的测试能力在很大程度上取决于所检查的自动机类型。

A2C 构建于 ACL2s 之上，因此可以使用多种实用功能，例如功能强大的宏系统，可用于定义便于声明自动机的形式（我们的形式忠实反映了相应的教科书 [28]描述）；Defdata 框架可将自动机组件定义为类型；定理证明可用于证明类型等价性；基于属性的测试可用于检查自动机构造之间的等价性，并测试此类构造是否满足特定性质；此外，反例生成可在自动机构造不满足性质时提供有用的反馈。由于 A2C 是开源的，因此可以轻松扩展以支持其他模型或理论，例如 ACL2s 内置支持的递归函数理论。

我们的贡献。 我们通过 A2C 做出了以下贡献：(i) 一个用于构建可执行的确定有限自动机、下推自动机和图灵机模型的开源库，(ii) 能够利用性质和测试用例对构建的自动机进行推理，(iii) 即时生成反馈，引导学生正确构建，(iv) 能够在本地运行，或在 Gradescope 等兼容的学习管理系统上运行，以及 (v) 只需具备少量 ACL2s 使用经验，即可扩展支持更多计算模型。据我们所知，目前没有其他工具具备以上所有功能。

论文概要。 第 2 节介绍自动机自动评分领域的相关工作。第 3 节说明我们选择 ACL2s 定理证明器的原因。第 4 节展示 A2C 针对学生提交生成的各类实用反馈。第 5 节介绍该系统的实现考量。第 6 节展示系统架构，并详细说明与 Gradescope 的集成。第 7 节讨论我们使用 A2C 自动评定（自动评分）作业和考试的经验。第 8 节列出 A2C 的一些局限，第 9 节则作出总结。

2 相关工作

目前已有多种工具和技术，可用于自动评分或为学生提交的自动机提供反馈。为简洁起见，我们将介绍一些最常用的工具，并简要探讨使用专用算法进行自动机等价性检查的可能性。

JFLAP（Java形式语言与自动机软件包）是一款用于形式语言的可视化教学工具。除了支持确定有限自动机、下推自动机和图灵机外，它还支持其他几种计算模型和解析算法。JFLAP 没有内置评分功能（因为它旨在供学生自行运行，以完成作业解答），不过已有一些 JFLAP 扩展程序尝试添加此类功能 [27, 24]。这两种扩展程序都只支持确定有限自动机，而 Shekhar等人 的工作只是通过有界检查字符串，来检验学生提交的内容是否与教师的解答等价。这两种扩展程序都无法与外部学习管理系统集成。JFLAP 声称自己是开源软件，但我们未能找到从其官方网站获取 JFLAP 源代码的方法。

OpenFLAP 是 FLAP 的较新版本，也是 OpenDSA 的一个组成部分 [26], 。OpenDSA 是一个开源、交互式电子教材系统。OpenFLAP 支持自动评分的确定有限自动机、下推自动机和图灵机构造练习，但对学生提交内容的评估仅取决于其是否通过教师提供的具体测试用例。OpenDSA 可以与 Canvas 学习管理系统集成。

Automata Tutor v3 [13] 是一个闭源在线平台，可自动为自动机评分并提供反馈。它已用于在 30 多所大学中教授数千名学生，学生和教师的满意度都很高 [13], ，这凸显了在本科计算理论课程中采用自动评分系统的益处。除其他几种计算模型外，它还支持确定有限自动机、下推自动机和图灵机。教师可以在 Automata Tutor 中创建包含自动机构造练习的作业。随后，学生使用 Automata Tutor 的界面以图形方式构造相应的自动机。Automata Tutor 会自动为练习评分；对于确定有限自动机，它还会提供旨在帮助学生理解其解答错误之处的“描述性提示” [14]。对于下推自动机和图灵机，Automata Tutor 只会提供反例（如果能够找到）。这些反例通过测试随机生成的字符串来生成，字符串长度不超过可配置的上限，并受特定资源限制约束 [13]。像 Automata Tutor v3 这样的闭源单体平台有一个缺点：当出现技术问题或进行维护而无法使用时，便无法可靠地使用它；我们上次检查时就是如此，当时是 12th 2022 年 11 月。

检查图灵机或（非确定性）下推自动机的等价性是不可判定的，但检查确定有限自动机的等价性是可判定的。现有算法可以检查等价性并生成见证（即两个确定有限自动机不等价的反例，例如，一个确定有限自动机接受而另一个拒绝的单词），其最坏情况下的运行时间复杂度相对于两个确定有限自动机的状态总数而言“近乎线性” [24]。未来若能为 A2C 增添一个用于检查确定有限自动机等价性的完整过程，将会很有帮助。

3 ACL2 Sedan

A2C 是用 ACL2 Sedan（ACL2s）编写的 [4, 12, 7], ，后者是应用式 Common Lisp 的计算逻辑（ACL2）的扩展 [2, 15]。ACL2 是一个工业级系统，支持集成建模、仿真和归纳推理。它源自 Boyer-Moore 定理证明器系列，能够运用带有数学归纳法的一阶逻辑进行推理。ACL2s 在 ACL2 的基础上增加了自动化和易用功能，例如先进的数据定义框架 Defdata、基于调用上下文图 [20] 和序数 [17, 18, 19], 的强大终止性分析、用于定理证明的基于性质的建模与推理框架、用于为无效性质生成反例的 *cgen*框架 [8, 9, 11, 31]，以及对系统编程的支持。系统编程是一项相对较新的能力，允许用户构建以 ACL2s 为关键组件的形式化方法赋能工具；这项能力已应用于涉及游戏化验证、教育、证明检查、与外部定理证明器对接以及安全的项目 [32, 29, 30]。

A2C 利用所有这些功能，在同一系统中支持编写可执行自动机构造、使用性质对其进行推理，并通过反例获得有用的反馈。

4 示例

在介绍系统的细节之前，我们将通过几个示例，展示 A2C 为学生提供的各种反馈。这些示例题也作为编程作业发布在 Gradescope 上，可按照 [1] 中提供的说明访问。

4.1 检查确定有限自动机

考虑这样一道作业题：要求学生构造一个确定有限自动机，使其能够识别由$\{0,1\}^*$组成且包含奇数个 1 的字符串。确定有限自动机 M是一个五元组（Q Σ,δ,q_0,F），其中 Q是一个有限状态集， Σ是一个字母表， δ: Q $\times\Sigma\to Q$是一个转移函数，$q_0\in Q$是一个初始状态，而 $F\subseteq Q$是一个有限接受状态集。假设学生提交了如下所示的 ACL2s 表单：

```
(gen-dfa
 :name     student-dfa
 :states   (e1 e2 o1 o2)
 :alphabet (0)
 :start    e1
 :accept   (o1 o2)
 :transition-fun (((e1 0) . e1) ((e1 2) . o1) ((e2 0) . e2) ((e2 2) . e2)
                 ((o1 0) . o2) ((o1 2) . e2) ((o2 0) . o1) ((o2 2) . e1)))
```

`gen-dfa` 是一个用于生成 DFA 构造的数据定义的宏。ACL2s 拥有强大的宏系统，可用于创建与书中自动机等价的表示。熟悉相关知识的用户可以定义自己的宏和语法。上面的 `gen-dfa` 表单定义了一个名为`student-dfa`的 DFA。表单中各组成部分的顺序无关紧要，例如，交换 `:states` 和 `:accept` 的顺序会得到相同的 DFA。读取 `gen-dfa` 表单时，A2C 会执行若干项检查，包括确保：(1) 名称是新的；(2) DFA 的所有组成部分均已提供；(3) 各组成部分格式正确，例如，转移函数是从相应定义域到相应陪域的全函数。

根据上述表单，A2C 报告称 `:transition-fun`(δ) 不是一个定义域为$Q\times\Sigma$的函数。这是因为上述表单中指定的 `:alphabet` (Σ)包含元素`0`，而`:transition-fun` 的定义域似乎是按照另一个字母表构造的，即该字母表同时包含`0` 和`2` 这两个元素。收到此反馈后，学生可以如下更新 `:alphabet` 。

```
:alphabet       (0 2)
```

更新后的表单通过了系统的所有检查，这表明学生提交的表单表示一个有效的确定有限自动机。接下来需要检查的是，该构造是否正确。正确自动机的规范是由教师提供的解答：

```
(gen-dfa
 :name           instructor-dfa
 :states         (even odd)
 :alphabet       (0 1)
 :start          even
 :accept         (odd)
 :transition-fun (((even 0) . even) ((even 1) . odd) ((odd 0) . odd) ((odd 1) . even)))
```

正确性测试可归结为测试学生的解答 `student-dfa` 与教师的解答 `instructor-dfa`之间是否等价。检查自动机的等价性还包括检查

字母表是否等价。因此，在这个运行示例中，A2C 尝试证明两个字母表的等价性，但失败了，并生成了以下反馈：

```
Incorrect alphabet provided.
```

这是因为学生的字母表与教师的 DFA 中提供的字母表不匹配。学生修正了字母表，并相应地更新了转移函数。

```
:alphabet (0 1)
:transition-fun (((e1 0) . e1) ((e1 1) . o1) ((e2 0) . e2) ((e2 1) . e2)
                 ((o1 0) . o2) ((o1 1) . e2) ((o2 0) . o1) ((o2 1) . e1))
```

不过， `student-dfa` 仍然不正确，正如 A2C 在以下反馈中指出的那样 ：

```
Transition function error. The following words are misclassified:
('(0 1 1 1) '(1 1 1 0) '(1 1 1))
```

反馈中的每个列表都表示一个被 `student-dfa` 错误分类的字符串。通过这些反馈，学生对自己的构造有了新的认识，并可以像下方所示，在生成的字符串上运行其 DFA：

```
(run-dfa student-dfa '(0 1 1 1))
```

The `run-dfa` form 在输入单词 `'(0 1 1 1)` 上运行 `student-dfa`，生成以下输出：

```
E2
```

这有助于学生意识到，从 `e2` 出发的任何转移都会返回 `e2`。学生按如下所示修正了这一错误：

```
:transition-fun (((e1 0) . e1) ((e1 1) . o1) ((e2 0) . e2) ((e2 1) . o2)
                 ((o1 0) . o2) ((o1 1) . e2) ((o2 0) . o1) ((o2 1) . e1))
```

更新后的解答终于被自动评分器接受，输出如下：

```
student-dfa is correct.
```

4.2 检查下推自动机

考虑另一道作业题，要求学生构造一个能够识别语言 $\{0^n 1^n \mid n \geq 0\}$ 的下推自动机。回想一下，下推自动机是一个六元组（Q Σ，Γ，δ，q_0,F），其中 Q 是有限状态集，Σ 是输入字母表，Γ 是栈字母表，δ: $Q \times (\Sigma \cup \{\varepsilon\}) \times (\Gamma \cup \{\varepsilon\}) \to \mathscr{P}(Q \times (\Gamma \cup \{\varepsilon\}))$ 是转移函数，$q_0 \in Q$ 是初始状态，而 $F \subseteq Q$ 是有限个接受状态组成的集合。

请看一位学生针对这道题提交的答案：

```
(gen-pda
 :name student-pda
 :states (q1 q2 q3)
 :alphabet (0 1)
 :stack-alphabet (0 z)
 :start-state q1
 :accept-states (q3)
 :transition-fun (((q1 0 :e) . ((q1 0)))
                  ((q1 1  0) . ((q2 :e)))
                  ((q2 1  0) . ((q2 :e)))
                  ((q2 :e z) . ((q3 :e)))))
```

上方所示的 `gen-pda` 表单定义了一个名称为 `student-pda` 的下推自动机。与前面介绍的 `gen-dfa`表单类似，此表单中各组成部分的顺序无关紧要。我们用 `:e` 表示

空字 ε。我们也可以使用表示 ε 的 Unicode 字符，但为了避免学生遇到因缺乏 Unicode 支持而产生的潜在问题，我们没有这样做。只要求输入 ASCII 字符，使 A2C 能够支持在线课程；在这些课程中，学生可能会使用各种开发环境编写自己的解答。有关状态、输入字母表和栈字母表有效性的检查，大多与确定有限自动机的检查类似。虽然我们不要求为状态、字母表和栈符号的每种可能组合都提供转移（默认情况下，我们假定其对应空集），但我们确实会检查初始状态在空字 `:e` 上是否存在转移。这是添加栈底符号所必需的。上方所示的提交未通过此项测试，因此学生会收到以下反馈：

```
Starting transition from (Q1 :e :e) missing from the transition function.
```

T通过添加一个新的初始状态 q0 并按所示更新转移函数来修正这一问题 n:

```
(gen-pda
 :name student-pda
 :states (q0 q1 q2 q3)
 :alphabet (0 1)
 :stack-alphabet (0 z)
 :start-state q0
 :accept-states (q3)
 :transition-fun (((q0 :e :e) . ((q1 z)))
                  ((q1 0  :e) . ((q1 0)))
                  ((q1 1   0) . ((q2 :e)))
                  ((q2 1   0) . ((q2 :e)))
                  ((q2 :e  z) . ((q3 :e)))))
```

更新后的解答通过了针对有效下推自动机的所有检查。现在该检查它是否符合题目规范了。在检查提交的下推自动机是否正确后，自动评分器报告

```
Transition function error. The following words were misclassified :
(:e)
```

这条反馈表明， `student-pda` 未能正确判定输入单词 ε。实际上，这个单词应该被接受，但 `student-pda` 并未接受它。可以通过修改接受状态集合来解决这个问题：

```
:accept-states (q0 q3)
```

或者修改 转移函数，使其允许从初始状态到接受状态之间存在一个 ε 转移 e:

```
:transition-fun (((q0 :e :e) . ((q1 z) (q3 :e)))
                 ((q1 0  :e) . ((q1 0)))
                 ((q1 1   0) . ((q2 :e)))
                 ((q2 1   0) . ((q2 :e)))
                 ((q2 :e  z) . ((q3 :e))))
```

这个更新后的解答终于在提交后获得了接受。

```
student-pda is correct.
```

4.3 检查图灵机

考虑这样一个问题：学生需要提交一台图灵机，使其将纸带上的 0 翻转为 1，反之亦然。图灵机定义为一个七元组（Q Σ,Γ,δ,q_0,$q$accept,$q$reject），其中 Q 是有限状态集， Σ 是

输入字母表，Γ是纸带字母表，δ：$Q \times \Gamma \to Q \times \Gamma \times \{L,R\}$是转移函数，$q_0 \in Q$是初始状态，$q$accept 是接受状态，$q$reject 是拒绝状态。我们要求图灵机在给定输入 w 上的运行结束时，读写头位于纸带上输出的末尾。运行结束后，我们忽略读写头右侧出现的任何符号。此外，我们还会删去输出末尾与读写头之间出现的所有空白符号。

假设某位学生针对上述问题提交了以下图灵机：

```
(gen-tm
 :name student-tm
 :states (q0 q1 q2 q3)
 :alphabet (0 1)
 :tape-alphabet (0 1)
 :start-state q0
 :accept-state q1
 :reject-state q2
 :transition-fun (((q0 1) . (q0 0 R))
                  ((q0 0) . (q0 1 L))
                  ((q0 nil) . (q3 nil R))
                  ((q3 nil) . (q1 nil L))))
```

上面显示的 `gen-tm` 表单定义了一台名称为 `student-tm` 的图灵机。与之前见过的 `gen-`表单类似，此表单中各组成部分的顺序无关紧要。提交此表单后，自动评分器给出了以下反馈：

```
Blank tape symbol nil missing from tape-alphabet.
```

空白带符号用关键字 `nil` 表示。`tape-alphabet` 的组成部分必须包含空白带符号 `nil`，且输入字母表必须是纸带字母表的子集。通过修改 `tape-alphabet` 即可解决此问题：

```
:tape-alphabet (0 1 nil)
```

编辑并重新提交后，自动评分器给出了更多反馈：

```
Incorrect output produced when running submitted TM on the following words :
('(1 0) '(0) '(0 1 1))
```

学生在反馈中提供的字符串上运行图灵机后，发现了转移函数中的一个错误，并按如下方式进行了更正：

```
:transition-fun (((q0 1) . (q0 0 R))
                 ((q0 0) . (q0 1 R))
                 ((q0 nil) . (q3 nil R))
                 ((q3 nil) . (q1 nil L)))
```

更新后的解答终于通过了自动评分器的检查。

5 实现

如前所述，我们的工具基于 ACL2s 定理证明器 [12, 4], , 它是 ACL2 [15,2] 的扩展，包含一种编程语言、该语言的逻辑以及一个交互式定理证明器。

5.1 解答格式

A2C 提供了一种定义明确的输入格式，用于描述自动机。采用这种格式的解答文件由一个或多个 `gen-x` 形式组成，其中 x 可以是 `dfa`、 `pda` 或 `tm`。每种形式都对相应的自动机提供了声明式描述。该描述忠实于其教科书定义 [28]。因此，这种输入格式易于自然地讲解和理解。

5.2 验证自动机

A2C 会验证表单 `gen-x`，检查以下条件是否全部成立 :

- 是否提供了自动机的所有必需组成部分,
- 初始状态是否属于给定状态,
- 接受状态集合是状态集合的子集,
- 转移函数的定义域类型正确,
- 转移函数的陪域类型正确,
- 针对下推自动机和图灵机分别进行的其他特定于模型的检查，例如，空白带符号不会出现在图灵机的字母表中，但应出现在带字母表中。

5.3 检查正确性

A2C 使用 Defdata 数据定义框架，将自动机描述转换为相应的状态、字母表、转移函数定义，以及使其可执行的函数。

对于基于属性的测试，教师可以使用 `property` forms（既用于测试，也用于定理证明），也可以使用 `test?` forms（仅用于测试）。基于属性的测试依赖 cgen 框架来生成针对无效属性的反例。A2C 利用 `interface` [32]library，通过 `test?` forms 向 ACL2s 查询，以测试等价性并提取反例。

Cgen 框架： 使用 ACL2s 的 `test?` forms 进行基于属性的测试依赖于 cgen。cgen 是一个反例生成框架，它以协同的方式结合了定理证明与测试。该框架相当成熟，已广泛应用于多个项目，包括工业项目。 [23, 21] 它提供了许多配置选项，并基于若干算法和思想，相关工作中对此均有介绍。简而言之，它运用一系列算法和定理证明器的全部能力来简化猜想，并将其分解为子目标；例如，它可能证明某个猜想在状态空间的某些无限区域内成立，从而将这些区域排除在后续考察范围之外。它还会在定理证明过程的关键环节采用一系列测试方法。测试与底层的 ACL2s 类型系统相集成，以生成给定类型的随机元素，并可选择满足某些约束。该反例生成框架甚至能够提升 ACL2s 证明定理的能力；例如，它可能发现泛化步骤会使给定猜想不成立，此时便会迫使定理证明器回溯，从而避免某些失败，并有可能最终得到证明。基于属性的测试所生成的测试数量可以配置。在本科生级别的计算机理论课程中，我们对 `test?`forms 使用了默认的测试用例数量，即 1000 个，并且据我们所知，没有出现任何假阳性。

等价性测试 价态： 要测试两个自动机是否等价，我们首先检查它们的字母表是否匹配 :

```
(defdata-equal instructor-x-alphabet student-x-alphabet)
```

这里， `instructor-x-alphabet` 和 `student-x-alphabet` 是以下定义：根据 `x` 是 `dfa`、 `pda` 还是 `tm`之一，分别由 `gen-dfa`、`gen-pda` 或 `gen-tm`生成，用于定义教师和学生自动机的字母表。 `defdata-equal` 用于检查两种数据类型是否相同。如果不同，defdata 框架 [10] 会生成一个属于其中一种数据类型、但不属于另一种的数据值。在这种情况下，如果学生和教师的字母表不同，Defdata 就会返回一个属于其中一个字母表、但不属于另一个的符号。要完成等价性检查，我们还需检查以下性质。

```
(test?
  (=> (instructor-x-wordp w)
      (== (accept-x w *student-x*)
          (accept-x w *instructor-x*))))
```

其中 `x` 是以下选项之一： `dfa`， `pda` 或 `tm`。

请注意，每当 `gen-x` 表单成功通过验证，并为给定构造 `student-x`的每个组成部分生成数据定义时，它都会在后台生成一个常量 `*student-x*`，用于引用由给定组成部分构成的列表。(`accept-x *student-x* w`) 和 (`accept-x*instructor-x* w`) 表单分别检查学生的自动机和教师的自动机是否接受类型为 `instructor-x-word` 的单词 `w`。此 `test?` 表单生成的任何反例都表明，恰好只有其中一个自动机接受该单词。下推自动机和图灵机在输入上允许运行的步数有上限，且可由教师配置。对于图灵机，我们使用另一种等价性测试，检查两台图灵机的输出是否一致。

```
(test?
   (=> (instructor-tm-wordp w)
       (== (remove-final-nils (left-of-head (run-tm w *student-tm*)))
           (remove-final-nils (left-of-head (run-tm w *instructor-tm*))))))
```

其中 `left-of-head` 的输出是 `run-tm` 读写头左侧的纸带部分，我们期望输出位于此处。 `remove-final-nils` 会移除输出末尾与读写头之间的空白符号。

自动机的构 造通过函数执行，其定义取决于自动机 `x` ：

- 给定初始状态$q0$、转移函数 δ和单词w， `run-dfa` 会返回状态$s \in Q$。从$q0$ 和w中的第一个字母开始，它会查询 δ 以获取一个新状态。重复此过程，直到w 中的所有字母都已处理完毕，并到达s 。由于 δ是完备的（$Q \times \Sigma$ 中的所有对都属于 δ的定义域），且w 的长度有限，因此 `run-dfa` 会终止。

- 考虑一个元组$t \in Q$ {v7>，它分别表示到达的状态、栈中的内容以及尚待处理的输入单词后缀。我们称之为执行元组。

  `run-pda` 是一个函数，接受一个界限（自然数）、一个下推自动机和一组执行元组，并返回布尔值：t 或 nil。 `run-pda` 的执行轨迹是一棵树，其中每个节点都是一个执行元组。这棵树的根是 (q0, ε, w)，即由初始状态、空栈和输入单词组成的元组。在执行的每一步，都可能生成树的新叶节点。回想一下， δ: Q $\times (\Sigma \cup \{\varepsilon\}) \times (\Gamma \cup \{\varepsilon\}) \rightarrow \mathscr{P}$(Q $\times (\Gamma \cup \{\varepsilon\})$)。对于叶节点 l $=$(q$_l$, (t . . .), (c . . .))，其中 q$_l \in$Q，t 是其栈顶，c 是剩余单词的首字母，我们有以下元组：(q$_l$,t, c)、(q$_l$,ε, c)、(q$_l$,t ε) 和 (q$_l$,ε,ε)。如果这些元组中任意一个存在于 δ 的定义域中，

我们就称叶节点 l 为活动的，并且可以使用 δ 生成 l 的子节点。如果这些元组都不存在于 δ 的定义域中，l 则仍为叶节点。

当执行元组的形式为 $(q^f, \ldots, \varepsilon)$，且$q_f \in F$时，该执行元组被接受。 `run-pda` 会一直执行，直到生成一个可接受的执行元组（此时返回 t），或者树的所有活动叶节点的深度都大于n（此时返回 nil）。

下推自动机固有的非确定性，加上可能进行 ε–转移的情况，使得无法保证 `run-pda`终止。因此， `run-pda` 的执行需要由一个正整数n限定；该整数是由教师指定的执行树最大深度。

- 给定一个图灵机 $(Q\ \Sigma, \Gamma, \delta, q0, q\text{accept}, q\text{reject})$ 和一个单词 w， `run-tm`会返回一个元组 $(Q \times \Gamma^* \times \Gamma^*)$，其第一个元素是图灵机的当前状态，第二个和第三个元素分别是读写头左侧和右侧的纸带内容。请注意，读写头左侧的纸带符号以逆序存储。通过移除读写头左侧纸带的第一个元素，并将其附加到读写头右侧纸带的开头，来模拟图灵机的读写头沿纸带向左移动。

  由于图灵机的停机问题不可判定，因此我们需要设定图灵机运行的最大步数，达到该步数后便停止运行。

属性测试：由于检查下推自动机和图灵机的等价性是不可判定的，因此教师可以准备一组性质，用于检查学生构造的正确性。例如，在我们贯穿全文的确定有限自动机示例中，

```
(property no-odd1s-in-ww (w :instructor-dfa-word)
  :proofs? nil
  (! (accept-dfa *student-dfa* (append w w))))
```

它会检查 *`student-dfa`* 是否不接受形如 `ww` 的字符串。这是因为这类字符串中的 `1` 的数量不可能是奇数。这类性质代表了一组无限的测试用例，可用于评估提交内容。当然，也可以定义基表达式等单元测试。ACL2s 中的属性形式可以配置为执行所需数量的测试。在我们的示例中，我们使用 `:proofs? nil` 在此属性中局部关闭定理证明，以便专注于测试并希望找到反例。也可以像下面这样全局禁用所有属性的定理证明：

```
(set-acl2s-property-table-testing? t)
(set-acl2s-property-table-proofs? nil)
```

现在定理　　证明已被禁用，并且已全局启用对属性形式的测试，我们有

```
(property accept-w->accept-0w1 (w :instructor-pda-word)
  (=> (accept-pda *student-pda* w)
      (accept-pda *student-pda* (app '(0) w '(1)))))
```

该属性检查：如果单词w 被 *`student-pda`* 接受，那么 $0w1$ 也会被接受。

最后，对于图灵机示例，我们可以使用以下属性来测试学生提交的作业（在其通过验证后）：

```
(property involution (w :student-tm-word)
  (== (remove-final-nils (left-of-head (run-tm
        (remove-final-nils (left-of-head (run-tm w *student-tm*))))))
      w))
```

它会检查：如果对输入单词w 运行 *`student-tm`*，得到的结果是将 0 和 1 互换后的w'，那么再次对w' 运行 *`student-tm`* 后，应输出w。

请注意，每个属性都是用 ACL2s 语言以可执行自动机的形式进行程序化描述的，因此创建测试用例轻而易举。

单元测试： 教师还可以使用 `check=` 形式，针对特定测试用例检查自己构造的内容。例如，在我们前面的示例中，

```
(check= (accept-pda *instructor-pda* '(0 0 0 1 1 1)) t)
```

检查 `*instructor-pda*` 是否接受 `'(0 0 0 1 1 1)`。类似地，

```
(check= (remove-final-nils (left-of-head (run-tm '(1 0 1 1 1 0 1 0) *instructor-tm*)))
   '(0 1 0 0 0 1 0 1))
```

检查在输入 `'(1 0 1 1 1 0 1 0 1 0)` 上运行 `*instructor-tm*` 时，是否会将 0 和 1 翻转。

6 系统描述

A2C 可以打包为可执行文件，并部署在任何能够运行该可执行文件来处理学生提交内容、并利用生成的输出为作业评分的在线评分平台上。

6.1 ACL2s 与外部工具

如第 3 节所述，ACL2s 在 ACL2 定理证明器的基础上扩展了多项自动化功能和便于用户使用的特性。ACL2 提供了功能齐全的编程语言，因此无需修改或了解 ACL2 的内部机制，就能在其中实现并集成额外功能。ACL2 构建于 Common Lisp 之上，其设计使用户能够编写与 ACL2 代码交互的 Common Lisp 代码，反之亦然。这也意味着，将外部工具集成到 ACL2 中相对容易。

ACL2s 提供了一项功能，允许用户将正在运行的 ACL2s 进程的状态保存为可执行文件。我们利用这项功能，为教师希望自动评分的每份作业生成一个可执行文件。此类可执行文件的状态包含我们编写的库，这些库提供了测试学生构造所需的形式，并支持与 Gradescope 集成；此外，还包含教师用于评分的任何自动机版本。



图 1：自动评分 ToC 作业的技术栈

6.2 库

我们为计算理论课程开发的基于 Gradescope 的自动评分器使用以下库：

- `A2C`：用于定义、验证、运行用户提供的自动机并检查其等价性的形式
- `gradescope-acl2s`：通过生成包含自动评分提交的分数和反馈的 JSON 文件，与 Gradescope 交互。

- `interface`： 提供 ACL2s 定理证明和反例生成功能的接口。

这些库旨在 A2C 以外的其他场景中重复使用。学生可以在本地使用 A2C 库，在提交前检查自己的自动机。例如，有兴趣的学生可以定义一个自动机，然后使用 ACL2s 检查具体的测试用例，以及他们认为该自动机应满足的性质。教师在开发解答自动机时，也可能会发现这些库有助于检查自己的工作。定义性质需要具备使用 ACL2s 定理证明系统的经验。东北大学的计算机科学一年级学生通常会在逻辑与计算课程中学习 ACL2s。

6.3 Gradescope 集成

Gradescope 是多所大学使用的学习管理系统。它提供自动评分功能，并且可以配置为允许学生多次提交同一份作业，并在每次提交后查看自动评分器的反馈。

Gradescope 自动评分器是一个 ZIP 压缩包，其中包含用于接收学生提交内容并生成 JSON 文件的代码；这些 JSON 文件记录每份自动评分解答所得的分数和反馈。学生提交的内容会在 Docker 容器中运行，这些容器按照自动评分器 ZIP 压缩包中提供的规范进行设置。由于 Gradescope 的 Docker 容器默认不支持 ACL2s，我们使用自定义 Docker 镜像 [3]。在大多数情况下，教师只需使用我们提供的示例之一，根据需要更新题目和解答，即可发布自动评分作业。

我们的自动评分器可以进行调整，以支持任何能够通过 Docker 镜像或可执行文件进行自动评分的学习管理系统。为此，需要编写一个与 `gradescope-acl2s` 类似的库，用于处理该学习管理系统所需的各种输入/输出。

7 经验与观察

我们在 Gradescope 上部署了 A2C，为东北大学一门本科级别的计算理论课程中的作业进行自动评分，该课程约有 50 名学生。在发布相应作业之前，我们先向学生介绍了用于指定每个自动机的输入格式。

7.1 易用性

使用 A2C 设置自动评分作业很容易。解答也很容易指定，因为只需使用教师的解答即可。期末考试中有一部分题目采用了自动评分。例如，期末考试中有一道类似这样的问题：“编写一台图灵机，在输入纸带上从左数第 3 次出现符号 # 后插入一个 1。”针对这道题提交的图灵机相当复杂（也就是说，它们有多个状态和转移）。自动评分帮助我们节省了约 16 小时的助教评分时间。此外，通常每三十名学生中就有一人会提出复议申请，这还需要额外花费 2 小时。因此，自动评分对学生也很有帮助。

7.2 轶事证据

我们观察到，与人工评分作业相比：(1) 自动评分作业的重新提交次数显著更多；(2) 学生在自动评分作业中获得的成绩更高；

(3) 学生对自动评分的反馈极为积极，因为即时反馈让学生能够发现简单的错误，并帮助他们更好地理解课程材料。平均而言，超过 95% 的学生在自动评分题目中获得了满分，而在人工评分题目中获得满分的学生不到 20%。

8 局限性

我们的方法有一些局限性，下面将对此进行讨论。首先，我们的工具不提供用于构建或检查自动机的图形界面。尽管我们用于提交自动机的格式与 Sipser 经典教材中使用的格式相似，[28], 学生仍可能需要学习如何读写 S 表达式，才能编写自动机描述。这在我们的课堂上不是问题，因为大多数学生已经上过“逻辑与计算”课程，并在其中接触了 ACL2s 编程。不过，如果学生不理解 S 表达式，使用我们的工具前可能需要接受一些额外培训。

为我们的工具添加自动机可视化功能相对容易——由于我们的工具是用 ACL2s 编写的，因此很容易开发能够生成适用于可视化工具的输出的代码。添加用于构建自动机的用户界面可能会稍微困难一些，而且无法与 Gradescope 集成。

虽然 A2C 不要求用户具备任何 ACL2s 使用经验，但某些功能确实需要 ACL2s 使用经验。具体而言，要扩展我们的工具，用户需要编写 ACL2s 函数。用户也可能需要对 ACL2s 有一定了解，才能使用属性系统。

此外，即使自动机等价问题是可判定的，我们用于检查自动机等价性的方法也并不完备，这是因为所有受支持的计算模型都采用了统一测试接口。换句话说，我们的工具可能会产生假阳性，例如，它可能会报告两个自动机等价，而实际上它们并不等价。在计算理论课程中使用我们的工具时，我们抽查了工具给出的结果，没有发现任何问题。学生所犯的错误可能通常很容易利用，也就是说，很容易找到反例。我们预计，对于更复杂的自动机，以及那些仅在少数字符串的分类上存在差异的自动机，工具更容易产生假阳性。为缓解不完备性问题，使用我们工具的教师可以提供测试用例和性质，以检查自动机等价性检查器可能遗漏的棘手错误。如果教师具备一定的 ACL2s 专业知识，还可以运用各种技术提升 ACL2s 找到不等价见证的能力。不过，在大多数情况下，添加更多性质就足够了。

总的来说，我们认为以不完备性换取易用性是值得的，因为这能让教师和助教有更多时间与学生一起钻研课程中真正有趣的部分。

9 结论与未来工作

我们介绍 A2C，这是一个基于 ACL2s 定理证明器的库，用于检查学生构造的自动机，并向学生提供自动反馈。这些构造是可执行的，能够生成形式模型，以便在定理证明器的环境中进行推理；具体做法包括检查其与教师构造的等价性，以及测试性质和测试用例。它能即时提供有用的反馈。A2C 既可在本地运行，也可在兼容的学习管理系统上运行。它已集成到 Gradescope 中，用于评阅作业

和考试。这些功能使 A2C 非常适合应用于大规模开放在线课程，因为这类课程的师生比通常过低，教师难以提供及时、优质且个性化的反馈。我们认为，基于定理证明器的反馈生成方法也可用于逻辑和离散数学等其他学科中的非编程作业。

未来，我们希望添加自动机可视化生成功能，让学生的学习更加直观有效。我们还希望为可判定问题添加判定过程，例如检查确定有限自动机的等价性。不过，由于 A2C 是开源的，最终用户也可以实现这类扩展。

致谢

我们衷心感谢 Mirek Riedewald 和 Jason Hemann，他们分别于 2020 年秋季和 2021年秋季担任东北大学计算理论课程的教师。

参考文献

[1] A2C：自动机自动检查器。 https://github.com/ankitku/A2C.[2] ACL2 网页。 https://www.cs.utexas.edu/users/moore/acl2/.[3] ACL2s Docker 镜像。 https://hub.docker.com/r/atwalter/acl2s_gradescope_autograder/tags.[4] ACL2s 网页。 http://acl2s.ccs.neu.edu/acl2s/doc/.[5] Gradescope。 https://www.gradescope.com.[6] Rajeev Alur、Loris D'Antoni、Sumit Gulwani 和 Dileep Kini（2013）：确定有限自动机构造的自动评分。载于：国际人工智能联合会议（IJCAI）。参见 http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6759.[7] Harsh Chamarthi、Peter C. Dillinger、Panagiotis Manolios 和 Daron Vroon（2011）："ACL2" Sedan 定理证明系统。载于：系统构造与分析工具和算法（TACAS），doi:10.1007/978-3-642-19835-9 27。_[8] Harsh Raju Chamarthi（2016）：交互式非定理证伪。博士论文，东北大学，doi:10.17760/D20467205。[9] Harsh Raju Chamarthi、Dillinger Peter C.、Matt Kaufmann 和 Panagiotis Manolios（2011）：集成测试与交互式定理证明。载于：ACL2 定理证明器及其应用国际研讨会（ACL2），doi:10.4204/EPTCS.70.1。[10]Harsh Raju Chamarthi、Dillinger Peter C. 和 Panagiotis Manolios（2014）：ACL2 Sedan 中的数据定义。载于：ACL2 定理证明器及其应用国际研讨会（ACL2），doi:10.4204/EPTCS.152.3。[11] Harsh Raju Chamarthi 和 Panagiotis Manolios（2011）：使用交互式定理证明器进行自动化规范分析。载于：计算机辅助设计形式化方法国际会议（FMCAD）。参见 http://dl.acm.org/citation.cfm?id=2157665。[12] Peter C. Dillinger、Panagiotis Manolios、Daron Vroon 和 J Strother Moore（2007）：ACL2s："ACL2 Sedan"。载于：国际软件工程会议（ICSE），doi:10.1016/j.entcs.2006.09.018。[13] Loris D'Antoni、Martin Helfrich、Jan Kretinsky、Emanuel Ramneantu 和 Maximilian Weininger（2020）：Automata Tutor v3。载于：计算机辅助验证国际会议（CAV），doi:10.1007/978-3-030-53291-8 1。_[14] Loris D'Antoni、Dileep Kini、Rajeev Alur、Sumit Gulwani、Mahesh Viswanathan 和 Björn Hartmann（2015）：自动反馈如何帮助学生构造自动机？载于：ACM 人机交互汇刊（TOCHI），doi:10.1145/2723163。

[15] Matt Kaufmann、Panagiotis Manolios 和 J Strother Moore（2012）：计算机辅助推理：一种方法。Springer US。参见 https://link.springer.com/book/10.1007/978-1-4615-4449-4。[16] Matt Kaufmann 和 J Strother Moore：ACL2：Nqthm 的工业级版本。载于：计算机保障会议（COMPASS），doi:10.1109/CMPASS.1996.507872。[17] Panagiotis Manolios 和 Daron Vroon（2003）：序数算术算法。载于：第 19 届国际自动演绎会议（CADE），doi:10.1007/978-3-540-45085-6_19。[18] Panagiotis Manolios 和 Daron Vroon（2004）：将序数算术推理整合到 ACL2 中。载于：计算机辅助设计中的形式化方法（FMCAD），doi:10.1007/978-3-540-30494-4_7。[19] Panagiotis Manolios 和 Daron Vroon（2005）：序数算术：算法与机械化。自动推理期刊，doi:10.1007/s10817-005-9023-9。[20] Panagiotis Manolios 和 Daron Vroon（2006）：基于调用上下文图的终止性分析。载于：计算机辅助验证（CAV），doi:10.1007/11817963_36。[21] Craig McMillan、Andy Crapo、Michael Durling、Meng Li、Abha Moitra、Panagiotis Manolios、Mark Stephens 和 Daniel Russell（2019）：通过使用 ASSERTTM 进行验证与确认，提高系统与软件开发的开发保障水平。技术报告，SAE 技术论文，doi:10.4271/2019-01-1370。[22] Mostafa Mohammed、Clifford A. Shaffer 和 Susan H. Rodger（2021）：通过可视化和自动评分练习教授形式语言。载于：ACM 计算机科学教育技术研讨会（SIGCSE TS），doi:10.1145/3408877.3432398。[23] Abha Moitra、Kit Siu、Andrew Crapo、Harsh Chamarthi、Michael Durling、Meng Li、Han Yu、Panagiotis Manolios 和 Michael Meiners（2018）：迈向完整且无冲突的需求开发。载于：2018 年 IEEE 第 26 届国际需求工程会议（RE），doi:10.1109/RE.2018.00036。[24] Daphne Norton（2009）：有限自动机等价性测试算法及用于 JFLAP 的评分工具。参见 https://scholarworks.rit.edu/theses/6939。[25] Susan H Rodger 和 Thomas W Finley（2006）：JFLAP：交互式形式语言与自动机软件包，计算机科学教育版。Jones and Bartlett Publishers, Inc.。参见 https://www.jflap.org/jflapbook/jflapbook2006.pdf。[26] Clifford A. Shaffer（2016）：OpenDSA：面向计算机科学课程的交互式电子教材。载于：ACM 计算科学教育技术研讨会（SIGCSE TS），doi:10.1145/2839509.2850505。[27] Vinay S. Shekhar、Anant Agarwalla、Akshay Agarwal、B. Nitish 和 Viraj Kumar（2014）：通过自动机构造问题和自动反馈增强 JFLAP。载于：当代计算国际会议（IC3），doi:10.1109/IC3.2014.6897141。[28] Michael Sipser（2013）：计算理论导论，第三版。Course Technology。[29] Andrew Walter、Seth Cooper 和 Panagiotis Manolios（2021）：用于循环不变式发现游戏化的推理引擎。arXiv 预印本 arXiv:2109.01121，doi:10.48550/arXiv.2109.01121。[30] Andrew T Walter、Benjamin Boskin、Seth Cooper 和 Panagiotis Manolios（2019）：代码循环不变式发现的游戏化。载于：AAAI 人类计算与众包会议论文集，doi:10.1609/hcomp.v7i1.5277。[31] Andrew T Walter、David Greve 和 Panagiotis Manolios（2022）：带约束的枚举数据类型。载于：计算机辅助设计形式化方法会议（FMCAD 2022），doi:10.34727/2022/isbn.978-3-85448-053-2 25。_[32] Andrew T. Walter 和 Panagiotis Manolios（2022）：ACL2s 系统编程。载于：ACL2 定理证明器及其应用国际研讨会（ACL2），doi:10.4204/EPTCS.359.12。