将自动机理论变为一门实践课程

Susan H. Rodger
Computer Science
Duke University
Durham, NC 27708

Bart Bressler
Computer Science
Duke University
Durham, NC 27708

Thomas Finley
Computer Science
Cornell University
Ithaca, NY 14853

Stephen Reading
Computer Science
Duke University
Durham, NC 27708

rodger@cs.duke.edu
翻译：161810120 李皓琨

摘要

我们提出了一种在形式语言与自动机课程中通过实践解决课程问题的方法。通过使用JFLAP，学生们可以轻松解决各种非常繁琐的问题，而这些问题通常需要通过纸和笔来进行繁复的运算才能被解决。通过与更为传统的理论问题相结合，学生们可以在自动机理论这个主题上学习更为广泛的知识，解决更为广泛的问题。因此，学生们可以使用JFLAP 在计算过程和视觉反馈上探索形式语言与自动机的概念，并且在理论学习上不会依赖 JFLAP。此外，我们为 JFLAP 提出了一个全新的特性：带子程序的图灵机模型。带子程序的图灵机模型通过使用其他图灵机作为自己的一个模块从而帮助人们更方便地构建更复杂的图灵机模型。

分类

F.4.3[计算理论] 数理逻辑与形式语言
D.1.7[软件] 编程技术、视觉编程

通用术语

理论

关键词

JFLAP，自动机，下推自动机，图灵机，文法，SLR 文法分析，LL 文法分析，L-系统

1. 介绍

传统上，形式语言与自动机(FLA)课程的书面作业有两种类型：证明题和构建模型的练习题。而第二种类型的问题仅限于一些比较小的例子。即使是构建一个只有八个状态的中等大小的自动机模型，学生也不愿意在这种模型上费劲去手动测试，因为这种测试方式非常的乏味。对这种问题进行评分也非常耗时，而且很容易判断失误。

所以我们提出了一种形式语言与自动机理论课程的实践方式，学生通过这种方式，可以使用 JFLAP 工具在计算过程和视觉反馈上更方便地探索形式语言与自动机理论的诸多概念。当然我们并不是提倡在课程中取消证明题类型的练习，而是通过这种实践探索的方式来补充课题内容。例如证明这样一个问题：如果 L 文法是正则的，那么 L^R 文法(包含所有 L 文法的字符串的倒序字符串的一种语言)也是正则的。这是课程中常见的一种证明题类型。对于一些学生来说，在证明这个题目之前，先将这个例子可视化，会对证明这个例子有所帮助。学生们可以从这些正则文法 L 开始，为它们构建一个确定有限自动机(DFA)，然后将这个 DFA 转换为 L^R 的 DFA。他们必须为这两个 DFA 创建测试数据以确认 DFA 是正确的。这种方法让形式语言与自动机的课程与其他的一些计算机专业课的教学方法更加一致，很多计算机专业课是要求学生动手做实验，通过构建，调试以及测试的方法来学习。

其他人也对形式语言与自动机课程采取了较为类似的实践方式，但只关注其中的

较少的一部分课题。Turing's World[1]中使用的方式让学生可以自行创建图灵机以及自动机并进行实验。这篇文章关注的重点是图灵机以及子状态机。Taylor[8]使用软件 Deus Er Machina 让用户可以进行图灵机、有限自动机、下推自动机以及其他几种类型的自动机的实验。Forlan[7]是一个与 Standard ML 结合使用的一个工具集，用于创建和实验有限自动机、正则表达式和正则文法。Language Emulator[9]是一个包含了很多常规的形式语言的工具集，其中包括 Moore 机和 Mealy 机，以及各种形式语言之间的转译方式。Grinder[4]开发了 FSA 模拟器用于进行有限状态自动机的实验。[3]是 Webworks 的一部分，这是一本涵盖了自动机理论的很多内容的超文本网页书籍，它包含了文本、音频、图片、插画、幻灯片、视频剪辑以及一些自主学习模型。

在本文中，我们提出了 JFLAP 的概述，然后给出几个例子用于说明如何在计算过程和视觉反馈上使用 JFLAP 来深入学习形式语言与自动机的概念。接着，我们展示了 JFLAP 的新特性，包括使用子程序图灵机构建更加复杂的图灵机模型。一个构建好的图灵机可以在其他的图灵机模型中作为子程序来重新命名和复用。最后，我们评估了 JFLAP 在全世界各地的使用情况，并对未来的工作进行展望。

2. JFLAP 概述

JFLAP[5,6,2]是一个教学工具，用于创建并实验几种不同类型的非确定性自动机、文法、正则表达式、L-系统，以及不同结构之间的转换。通过 JFLAP 用户可以构建有限自动机(FA)、下推自动机(PDA)或者多带图灵机(TM)，并且可以观测这些模型在不同输入下的模拟情况。用户也可以输入正则文法、上下文无关文法(CFG)或者无限制文法并观察该文法对字符串的解析过程以及作为结果显示的语法树。

JFLAP 允许不同的形式语言和自动机之间互相转换。它可以将 NFA 转换为 DFA 再转换为最简 DFA，也可以在 NFA 和正则文法之间进行互相转换，也可以在 NFA 和正则表达式之间进行互相转换。它可以将不确定性 PDA(NPDA)转换到 CFG，或者将 CFG 转换为 NPDA，也可以将 CFG 转换到乔姆斯基范式，同时删除 lambda 生成式，单一生成式以及其他无用生成式。它可以将 CFG 转换为 LL(1)或者 SLR(1)的解析表并且用这个表解析文法中的字符串。最后，它还可以创建 L-系统，这是一种特别的文法形式，专门用于建模植物的生长和分形的生成。

3. 使用 JFLAP 解决问题

前一节描述了 JFLAP 中自动机与文法的构造与测试功能，以及它的形式转换功能。这些功能本身允许用户可以用比手工更方便的方式来构建和测试自动机。

现在我们描述其他几种可以用 JFLAP 解决的问题，而在纸上手动解决这些问题非常费劲。

3.1 有限自动机的比较

给定两个不同的有限自动机，确定它们是否等价，如果不等价，则表明它们不接受相同的文法。学生可以从现有的文件中获得两个有限自动机，也可以直接用 JFLAP 构建它们。同时学生必须编写一组良好的测试数据，将它们输入字符串并运行模拟器进行验证。JFLAP 允许多开窗口来同时测试不同的输入数据。或者学生们还可以简化这两个有限自动机从而对比它们的结果。最后 JFLAP 的等价性比较将确认这两个有限自动机是否等价。如果这两个有限自动机不等价，则学生只需要确定一个特别的字符串，这个字符串在其中一个有限自动机中被接受，而另外一个有限自动机不接受。

3.2 正则表达式的比较

给定两个正则表达式，确定它们是否等价。在 JFLAP 中我们无法为一个正则表达式测试字符串。但是，我们可以将正则表达式转换成一个等价的有限自动机，并且运行一

系列测试字符串，进行类似上面一部分提到的两个有限自动机的比较。

3.3 逆向工程-DFA 到 NFA

这个问题展示了我们对 NFA 如何转换为 DFA 的理解，但是这是个逆向的过程。学生将 NFA 交给 JFLAP 转换并获得了一个 DFA。这个 DFA 的每个状态都标记上了来自 NFA 的状态数。这个问题的关键是如何创建原始的 NFA。有一种假设是，原始的 NFA 没有任何的 lambda 转换。一旦构建出来原始的 NFA，学生就可以使用与 DFA 的等价性比较功能来确定他们是不是构建了正确的 NFA。

3.4 创建基于文法属性的自动机

JFLAP 可以用于构造说明文法属性的样例。例如，给定两个自动机，要求构建一个表示这两个自动机的并集的自动机。若这两个自动机已经被构建出来了。则使用“合并自动机”功能，两个自动机就会被放在同一个编辑界面中，并且其中一个自动机的初始状态会被删除，因为只有一个状态可以作为初始状态。接着，用户就可以对其进行连接和修改。在这个例子中，一个新的初始状态将会被创建，并且 lambda 产生式会从新的初始状态中被添加到先前的每个初始状态中。这样用户就可以在多个输入数据上测试新构造出的自动机了。

一个更加复杂的例子是考虑一个被称为 SwapFirstLast(L)的属性，它接受 L 文法中每个字符串的第一个字符，并且与这个字符串的最后一个字符进行交换。学生需要证明如果 L 是正则的，那么 SwapFirstLast(L)也是正则的。首先，他们需要使用 JFLAP 将文法 L 构造为一个简单的 DFA M，接着将其转换为文法 SwapFirstLast(L)的 DFA M2。其次，他们需要不借助 JFLAP，从形式上证明文法 SwapFirstLast(L)是正则的。

3.5 确定可区分的状态

JFLAP 提供的转换方式之中有一个是将 DFA 转换为最简状态的 DFA。这个 JFLAP 的算法最开始会假设所有的终止状态都是不可区分的，而且所有的非终结状态也都是不可区分的，并且将这些不可区分的状态分为两组，一种是终止状态，一种是非终止状态。然后这个算法会尝试确定每个集合中的一些状态是否可以区分，从而将一个集合分解成两个或者多个集合。如果用户怀疑同一个集合中的两个状态是可以区分的，那么他们可以自行修改 DFA，使得每个状态变成初始状态（在不同的操作中），并且确定一个字符串，这个字符串可以被修改过的 DFA 接受，而不是另外一个 DFA 接受。假如存在这样的一个字符串，那么这些状态就是可以区分的，需要放在不同的集合中。

3.6 不确定性探索

大多数学生都习惯于按照顺序去进行思考。然而当给出一个只能不确定地去解决的问题的时候，学生们就会深陷其中苦苦挣扎。确定一个字符串是否为回文串这个问题可以用不确定性 PDA(NPDA)来解决，而不是使用一个确定性 PDA(DPDA)来解决。按照先前的习惯，学生们会使用惯性思维，寻找字符串的中间部分，接着确定两边的部分是否都相同。但是这种算法并不能使用 DPDA 来实现。学生们可以使用 JFLAP 来为这个问题建立一个 NPDA，然后观察这种非确定性的思维是如何解决问题的。当使用 JFLAP 进行模拟的时候，每个当前的运行状态和运行过程都会被展示出来。对于一个有效的输入串，只要其中一个状态和过程匹配到了字符串的最中间部分，那么接下来的模拟就会持续匹配左右的两个部分，直到能判断接受这个字符串为止。

3.7 文法的指数级增长

JFLAP 可以展示一个文法是如何指数级扩展的。比如说，给予学生们左侧的这个包含 lambda 产生式的文法，并且要求他们将其转换为一个等价的且不包含 lambda 产生

式或者单一产生式的文法，生成的 CFG 显示在右侧。接着他们被要求比较这两个字符串匹配的文法。对于输入串 aaababaabbb，左侧的文法会使用更长的时候来接受，并且在生成树中有一共 13286 个节点。而右侧的这个文法只在生成树中生成了 335 个节点并且快速接受了输入串。

$S \to aB$
$S \to Ba$
$B \to aBb$
$B \to BB$
$B \to bBa$
$B \to \lambda$

$S \to aB$
$S \to Ba$
$S \to a$
$B \to aBb$
$B \to BB$
$B \to bBa$
$B \to ab$
$B \to ba$

类似的例子也可以使用无限制文法来显示。这些文法的左侧会有更多的产生式，这些产生式会让解析过程变得更为迅速。

3.8 确定一个语法对应的文法

JFLAP 可以用于确定一个给定的 CFG 的文法类型。在这种方法中，我们可以用多个输入数据来测试 CFG。而在另外一种方法中，用户可以将问题细分为更小的子问题来解决。对于每个变量，输入它对应的产生式来确定这个变量的生成能力（可以用临时的终结符来替换其他的变量）。例如，考虑如下的七个产生式的语法。用户可以在一个新的语法编辑窗口输入所有的 B 产生式，用一个小 s 来表示 S（否则就无法进行产生式展开）。接下来用户可以使用字符串集合{ b , ab , bs , aabs , absa , ...}并且确定 B⇒a*b(λ +S)a*。只输入 S 产生式并且用一个特别的终结符来表示变量 B，那么用户可以确定 S⇒a*bBa*。将它们综合起来，S⇒a*ba*b(λ +S)a*或(a*ba*b)*a*。接着用户就可以用一组测试字符串集合来测试这个文法。

$S \to aS$　$S \to Sa$　$S \to bB$　$B \to aB$
$B \to Ba$　$B \to bS$　$B \to b$

3.9 深入学习 FOLLOW 集

LL 以及 SLR 语法分析的开始步骤之一是计算语法中每个变量的 FOLLOW 集，变量的 FOLLOW 集是在展开式中所有跟随这个变量的终结符的集合。学生们已经学习到了计算 FOLLOW 集的正确算法，并且大部分人都可以顺利地运用这个算法去推导 FOLLOW 集，但是这不代表我们清楚了学生们是否真正地理解了 FOLLOW 集的概念。对于这个问题，给予学生们一个语法，并且要求学生计算这些产生式的 FIRST 集和 FOLLOW 集。然后，要求学生们把每个产生式展开为只使用 FOLLOW 集中终结符表示的形式，这样就可以看出来 FOLLOW 集中的终结符是紧跟在非终结符后面的。他们可以通过使用 JFLAP 的穷举分析器来解析字符串。每个字符串都会生成一个解析过程，我们可以直接观察到这个解析过程中每个展开式的形式。当做出一些选择时，JFLAP 会根据顺序替换产生式，所以可能需要多个字符串，否则无法找到对应的展开解析式。

3.10 语法分析：两种方法

我们展示了 JFLAP 是如何被广泛用来从两种方式上进行 SLR 语法分析的学习的。类似的方法也适用于 LL 语法分析。给定一个 SLR(1)语法，第一种方法是使用 SLR 语法分析将这个语法转换到 NPDA。所得到的 NPDA 有三个状态并且很有可能是不确定的。学生们用几个不同的输入来运行这个 NPDA，通过在每一步选择 lookahead 并且不使用部分匹配式，从而使运行的结果具有确定性。接下来，学生们可以通过第二种方法来查看语法分析的过程。他们为这个语法构建一个 SLR(1)解析表，并且使用有相同 lookahead 的一组输入来一步步运行语法分析。

3.11 分析非 SLR(1)的语法

使用 JFLAP，即使语法分析表中存在冲突，SLR(1)的解析表仍然是可以被构建出来的。对于每个存在冲突的状态，用户可以选择其中一个。这样用户就可以继续使用解析表来对字符串进行分析。当然并不是所有该

语法中的字符串都是可以被这个经过选择的解析表成功解析的。这里有一个问题可以用来给学生测试他们是否理解了语法分析表。给定一个不是 SLR(1)的 CFG 以及一个给定的输入字符串，试找出一种正确的冲突选择生成的语法分析表，并且这个语法分析表可以正确解析该字符串。

3.12 运行一个通用图灵机

使用 JFLAP 的 3 带图灵机，我们已经构建了一个包含 34 个状态的通用图灵机。通过使用通用图灵机，学生可以用少量的状态转移函数来编辑一个简单的图灵机，每个状态转移函数都是一个长度约为 15 的包含 0 和 1 的字符串。然后，学生可以输入一个包含图灵机和一个输入字符串的测试数据用来模拟，并且观察模拟过程。

3.13 单带图灵机与双带图灵机的对比

学生们将被给予一个语法 $a^n b^n c^n$，并且要求为这个语言使用 JFLAP 构建一个单带图灵机，然后再用 JFLAP 为这个语言构建一个双带的图灵机。然后，学生们将会使用多个不同长度的输入字符串来运行在这两个图灵机上，并且进行对比。在这个例子中，一个高效的单带图灵机将会以 $O(n^2)$ 的时间复杂度运行，而一个高效的双带图灵机则会以 $O(n)$ 的时间复杂度运行。

4. 新特性：子程序模块

JFLAP 的一个最新特性是带有子程序的图灵机。子程序模块也是一种图灵机，其具有特定的用途，那就是被用来作为组件来构建另外一个复杂的图灵机。人们可以使用这种子程序模块来更容易地构建更加复杂的图灵机。

4.1 子程序的创建

想要构建一个子程序模块的话，首先得用状态和状态转移函数先构建一个图灵机，并且保存为一个文件。这个图灵机可以通过选择创建子程序模块功能从而从文件中读取进来，成为一个子程序模块。这个子程序块会被显示为一个方框，并且可以使用状态转移函数来和其他状态进行连接。我们为更方便地连接子程序模块提供了特别的状态转移函数。当然子程序模块也可以使用状态以及子程序模块的组合来构成。

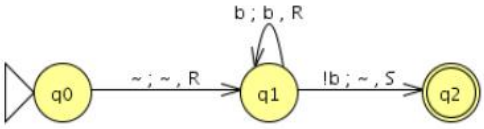

图 1：Rnot_b 的子程序

如果想从一个简单的基础开始，那么就该使用子程序模块。我们列出了可以用于构建复杂图灵机的一个简单图灵机库。这些子程序可以提供给学生使用，或者他们可以自行构建其中的部分图灵机或者构建所有表中的图灵机。

R	向右移动一次
R_a	向右移动一次，持续向右移动直到遇到第一个 a
Rnot_a	向右移动一次，持续向右移动直到遇到一个不是 a 的字符
a	写 a（不移动）
start	模块开始
halt	模块暂停

上表中每一个都代表了一个简单的图灵机。也有类似的图灵机用于向左移动读写头，比如 L,L_a 以及 Lnot_a，也有类似的机器用的是字母表中的其他字母。

子程序模块可以用标准的图灵机状态转移函数来连接。在 JFLAP 中，标准图灵机的状态转移函数 a;b,R 表示读 a 写 b，读写头向右移动。我们已经为现有的状态转移函数以及新的状态转移函数类型创建了特别的符号表。符号~表示忽略读写。状态转移函数~;~,R 表示的是忽略读的符号，忽略写的符号，并且读写头向右移动。而符号!x 在

读这个位置表示的是匹配的终结符是所有不是 x 的终结符。例如，图 1 是一个图灵机，用于向右移动一次，然后继续向右移动直到有一个不是 b 的符号为止。我们给这个图灵机取名为 Rnot_b，并且将使用它作为一个子程序模块。

人们可能希望用两种方式来连接两个子程序模块。第一种方式是通过连接他们来使他们按照顺序执行，用~;~,S(S 代表读写头不移动)。第二种方式是，在处理完第一个子程序后，可能希望根据当前的符号移动到对应的第二个子程序处。例如，a;~,S 表示如果 a 是当前读写头读到的符号，那么就进入该状态转移函数连接的那个子程序，并且读写头不要移动。为了进一步简化两个子程序之间的连接，我们提出了“子程序转移函数创建”功能，该功能使得转移函数只会显示读的内容，并且默认写是~以及读写头移动方式是 S。为了减少除了一个符号外的代码的重复度，我们允许 a1,a2,…an}v 这样的写法。这表示如果读取到其中一个 ai，那么后面出现的所有的 v 都会被 ai 取代。

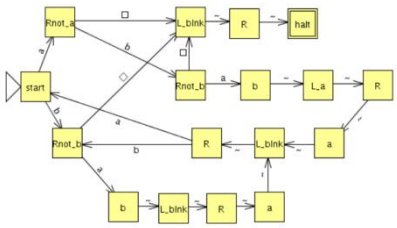

图 2：先输出 a 的图灵机

图 2 展示了一个只用子程序模块构建的图灵机，这个图灵机表示的是函数 $f(w)=w'$，其中 w' 首先列出了 w 中所有 a，然后列出了 w 中所有的 b，例如 f(babba)=aabbb。这个图灵机从一个开始子程序开始执行，这个子程序是一个非常简单的图灵机，只包含一个初始状态和一个结束状态。接着它循环向右移动，找到第一个跟随在 b 后面的 a，并且使用 ab 来替换掉它，然后用 a 替换掉最左边的 b。当所有的 a 都在所有的 b 的左边时，磁头移动到了最左边的符号，并且进入停止子程序。在这个图灵机中，所有的状态转移函数都是用“子程序状态转移函数创建”功能创建的。例如图中 start 块到 Rnot_a 块的状态转移就是用这个功能创建的。这个状态转移函数的意思是“如果读取到当前指向的字符是 a，不要在这个位置写，也不要移动读写头，直接去 Rnot_a 块执行这个块的子程序”。

用子程序模块构建的机器往往是非常复杂的。使用子程序模块构建的带子程序的图灵机也可以被命名并且保存到文件中，并且也可以作为子程序被使用。一旦这个子程序模块被“构建子程序”功能读取，那么这个图灵机就会被复制下来作为子程序副本保存在这个新的图灵机中。如果第二次还是读取的同一个子程序模块，则会使用这个已经作为的副本的老版本图灵机。一旦这个子程序成为了图灵机的一部分，那么它就可以被修改。当选择“属性编辑”功能时，用户可以单击一个子程序块，并且选择“编辑模块”功能对该子程序进行编辑。这个子程序块的图灵机就会出现在一个新的窗口中，并且可以进行修改。值得注意的是：如果在一个图灵机中，这个子程序块有多种用途，修改这个子程序块只会为这个模块创建一个新的子程序块，而不会影响到其他之前相同的模块。最好在图灵机使用该子程序块之前就构建并且测试好这个模块，这样以后就不用再麻烦去修改了。

子程序块的默认名字是读取该模块时候的文件名。也可以通过使用“设置名称”的功能来修改它的名字。

4.2 子程序图灵机的模拟

带子程序图灵机验证输入字符串的模拟器拥有五种功能。第一个功能，“单步执行”，提供了图灵机单步执行的执行路径的状态跟踪。当追踪进子程序块的时候，只要是在子程序的状态中执行，那么该子程序块就会被高亮显示。当选择“状态详情”功能时，视图会被更新，以显示当前执行的子程序的状态转换图，并高亮显示当前执行的状态。选择“退出详情”功能时，将会切换回原本的图灵机，并且高亮显示该子程序块。

第二个功能，“单步直接执行子程序”将会展示原本的图灵机的状态转换图，并且

每个模块都会被认为是模拟中的一个状态直接执行。这样，图灵机就能快速在追踪过程中跳过子程序的内部追踪。如果一个模块表示“向右移动直到遇到空白符”，那么在“一步直接执行”中，磁带读写头将会直接移动到最右侧空白处。接着是三个快速执行的功能选项。快速执行将会接收一个输入字符串，并且直接给出是否接受这个字符串，而不会给出跟踪过程。“多输入执行”以及“多输入执行（转换器）”将会返回多个输入字符串的接受结果，并且不给出对应的执行过程。

4.3 其他新特性

JFLAP 还有其他一些新的特性。其中一个是能更改所有类型的自动机中状态的名字。所有类型的自动机中，默认的状态名字是 qX，并且 X 是从 0 开始的状态数。图 1 就展示了状态 q0,q1,q2 的默认命名。而图 3 展示了一个有四个状态的图灵机，这四个状态被重命名为 start，1,2 和 3。这个图灵机是 $f(w)=w'$ 的转换器，并且 w 必须从一个 a 开始，且至少含有一个 b。这个转换器的输出是一个只含有 b 的字符串，且这个字符串的长度等于输入字符串中，第一组只包含 a 的子字符串的长度。另外一个新的特性是，作为转换器的图灵机可以使用多个输入同时运行。图 4 就显示了对几个输入字符串的模拟情况以及模拟器对这个输入字符串处理后得到的一组输出。第四个输入是无效的，因为它不包含至少一个 b。

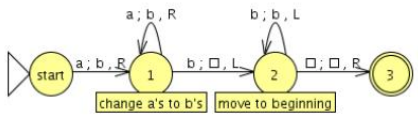

图 3：将第一组 a 全部替换为 b

Input	Output	Result
aaaaab	bbbbb	Accept
aaabab	bbb	Accept
ab	b	Accept
aaa		Reject

图 4：转换器 TM 多输入处理

5. JFLAP 在世界各地的使用情况

自 2003 年 1 月以来，JFLAP 已经被全世界超过 120 个国家下载了超过 25,000 次。在所有用户类型中，29%为本科生，18%的研究生，以及 16%的教职员工。33%的用户是被要求使用 JFLAP 的，而 36%的用户是自愿选择 JFLAP 来使用的。JFLAP 的使用类型中，48%是研究源码，25%是用于家庭作业，15%用于课程和讲座，12%被实验室用于实验。而使用 JFLAP 的理由中，46%是为了参加一门课程，14%是为了在课堂上配合教学使用，6%用于研究任务。这些统计数据合计之所以没有到 100%，是因为有些用户选择不反馈。

6. 结论以及未来工作展望

我们仍然在继续开发 JFLAP，并且为其添加算法，以及寻找在自动机课程中使用 JFLAP 的新方式。我们最近开始了一项为期两年的研究，用于评估 JFLAP 作为学习工具的有效性。十几所大学选择使用 JFLAP 进行自动机课程教学并且参加了这项研究。2005 年 6 月我们举行了为期两天的 JFLAP 研讨会，在此期间我们收到了关于使用的 JFLAP 方面的诸多反馈，因此我们现在有了许多新的改进思路，这些思路将会在未来被计划实施。

7. 参考文献

[1] J. Barwise and J. Etchemendy. Turing’s World 3.0 for the Macintosh. CSLI, Cambridge University Press, 1993.

[2] R. Cavalcante, T. Finley, and S. H. Rodger. A visual and interactive automata theory course with jflap 4.0. In Thirty-fifth SIGCSE Technical Symposium on Computer Science Education, pages 140–144. SIGCSE, March 2004.

[3] J. Cogliati, F. Goosey, M. Grinder, B. Pascoe, R. Ross, and C. Williams. Realizing the promise of visualization in the theory of computing. JERIC, to appear, 2006.

[4] M. T. Grinder. A preliminary empirical evaluation of the effectiveness of a finite state automaton animator. In Thirty-fourth SIGCSE Technical Symposium on Computer Science Education, pages 157–161. SIGCSE, February 2003.
[5] S. H. Rodger. Jflap web site, 2005. www.jflap.org.
[6] S. H. Rodger and T. W. Finley. JFLAP - An Interactive Formal Languages and Automata Package. Jones and Bartlett, Sudbury, MA, 2006.
[7] A. Stoughton. Experimenting with formal languages. In Thirty-sixth SIGCSE Technical Symposium on Computer Science Education, page 566. SIGCSE, February 2005.
[8] R. Taylor. Models of Computation and Formal Languages. Oxford University Press, New York, 1998.
[9] L. F. M. Vieira, M. A. M. Vieira, and N. J. Vieira. Language emulator, a helpful toolkit in the learning process of computer theory. In Thirty-fifth SIGCSE Technical Symposium on Computer Science Education, pages 135–139. SIGCSE, March 2004.