****

**学生研究计划(SRP)项目验收**

**项目成员个人结题报告**

|  |  |
| --- | --- |
| **参加项目名称：** | **编译原理课程在线教学辅助系统的研发** |
| **参加项目编号：** | **X201910561494** |
| **参加起止时间：** | **2019年 3 月至 2020 年 3 月** |
| **指导教师姓名：** | **徐扬** |
| **指导教师所在学院：** | **软件学院** |
| **学生姓名：** | **成子谦** |
| **学生学号：** | **201730681303** |
| **学生手机号：** | **17620720853** |
| **学生所在学院：** | **软件学院** |
| **学生所学专业：** | **软件工程** |
| **填表日期** | **2020 年 4 月** |

|  |  |
| --- | --- |
| **教务处** | **制** |
| **2020年3月** |

1. **文献调研**
   1. Web应用的可行实现方案

目前正处于互联网技术高速发展的阶段，自从基于B/S架构的web应用广泛被使用以来，web应用的开发技术就一直引领潮流、推陈出新。从传统的前端语言+后端语言的直接交互，到前端+后端框架技术的应用，web应用开发技术顺应时代的潮流与开发人员的要求，经历了一次又一次的变革。

为了扩展前端工程师的技术栈、使得更多的前端工程师能轻松接触后端业务，Node.js这一JavaScript运行环境顺应而生。Node.js既迎合了前端工程师的技术栈、降低了后端技术的上手难度，同时又具备运行速度快、异步IO（非阻塞IO）的优点，使得它得以轻量和高效，非常适合在分布式设备上运行数据密集型的实时应用。Node.js对一些特殊用例进行优化，并提供替代的API，使得其所采用的V8引擎在非浏览器环境下运行得更好。

本次SRP项目主要技术栈为JavaScript，其中，后端部分使用了Node.js技术。这也是目前比较成熟的web应用实现方案之一。

* 1. 正则表达式、NFA、DFA的转化

这一部分知识主要参考《Compilers: Principles, Techniques, & Tools》一书，这也是学院《编译原理》课程相关教材。

正则表达式，可以认为是对一组字符串集合的描述。有限状态机也可以用来描述字符串集合。并且，有限状态机是可以“执行”的，可以用来对输入的字符串进行检测。如果最终匹配，也就意味着输入的字符串和正则表达式匹配。所以，编程语言中的正则表达式，一般是通过有限状态机来实现。正则表达式匹配字符串的过程，可以分解为：1、正则表达式转换为等价的有限状态机。2、有限状态机输入字符串执行。

NFA和DFA的区别在于，NFA是“不确定有限状态机”，而DFA是“确定有限状态机”，“确定”意即在一个状态下，输入一个符号，一定是转换到确定的状态，没有其他的可能性。NFA在进行正则表达式匹配时，由于其状态的不确定性，所以其需要在执行过程中尝试所有的可能结果。当一条路径发生匹配失败后，需要进行回溯操作，并切换另一条路径并尝试匹配，直到匹配成功或所有路径都被尝试过。由于DFA消除了这种不确定性，故可以预见，DFA执行性能应该比NFA要好。

NFA可以转化为等价的DFA，也就是说，正则表达式可以用DFA实现，从而获得优于NFA的执行性能。但是从NFA到DFA的转化过程，会消耗更多资源，甚至最终得到的DFA会产生大量空间开销。而且DFA与NFA相比，在实现一些正则表达式的特性时会更复杂，成本更高。所以当前的许多编程语言，其正则表达式引擎为NFA模式。

从正则表达式到NFA的转换，主要依赖Thompson算法。虽然Thompson算法并非是最高效的算法，但其易于理解、实现简单，且具有很高的实用性。

Thompson算法使用最基本的两种转换：普通转换和epsilon转换。普通转换需要输入字符；而epsilon转换无需输入，即可从一种状态转移到另一种状态。而正则表达式中的各种运算，可以通过上述两种转换实现。

在正则表达式转换为NFA之前，我们需要把正则表达式转换为后缀表达式，以体现符号的优先级。可以通过调度场算法来实现这一功能。通过对后缀表达式进行迭代，并识别其中的运算符来执行，就可以对后缀表达式进行求解。对于正则表达式来说，则是在将其变为后缀表达式之后，通过解析的过程来进一步构建，并得到最终的NFA。

基于后缀表达式创建NFA，是一个由简单的 NFA 进行不断组合得到复杂 NFA 的过程。在构造最终 NFA 过程中，每次会创建 NFA 的片段。但不管 NFA 片段内部是怎样复杂，它都只有一个开始状态，一个最终状态。处理的过程大致如下：

1. 创建用于记录NFA片段的栈stack。
2. 依次读取输入的后缀表达式的每个字符c。
3. 如果c是运算符，从stack出栈所需数目的NFA片段，构建新的NFA片段并入栈。
4. 如果c是普通字符，创建新的状态，并构建只包含此状态的NFA并入栈。
5. 返回栈顶的NFA片段，即为最终结果。
6. **研究方案**
   1. 已有项目的研究

研究团队接手了一个具有一定完成度的web应用（即本SRP项目所涉编译原理课程在线教学辅助系统），前端使用Vue.js框架搭建，后端使用express框架搭建。

项目设计功能为：根据学生输入的正则表达式，构造出相应的NFA和DFA，并完成词法分析、语法分析、语义分析等相关工作。

项目已完成部分为：正则表达式转换、词法分析部分（但均存在部分bug需要修改）。语法分析、语义分析功能尚未开发，需要在之后的迭代版本中予以实现。

研究团队确定了本次SRP项目的主要研究内容：

1. 在不更改web应用框架结构的前提下，继续维护原有的项目。
2. 完善web应用词法分析部分，修正存在的bug。
3. 提出新的功能，并实现之。
   1. 提出可行改进方案

研究团队经过一星期的熟悉、使用项目流程后，与老师开会探讨，对项目现有的问题提出了以下几点看法：

1. 目前，web应用前端页面仍然存在部分bug，具体表现为：当用户进行了不正确的输入时，NFA显示出错。这使得web应用鲁棒性降低。
2. 词法分析部分功能单调，老师认为难以满足教学需求。要求研究团队开发更多功能，加强web应用的互动性，使得学生更容易理解词法分析的具体流程，提高学生的上课体验与老师的教学体验。
3. 研究团队成员根据个人使用体验，提出了开发“连线练习”的功能：在输入正则表达式并生成NFA、DFA后，要求学生手动构造NFA或DFA，并验证学生的答案是否正确，提高学生对NFA、DFA构造过程的理解。

研究团队认为，“连线练习”不仅开发难度不大，且具有一定的教学意义。老师同意了研究团队的意见，认为此项方案可以实行。

1. **成员分工**
   1. 前端部分

Web应用前端部分由刘生乾、潘墉圳、邝碧霞、李梓然同学负责。前端组组长为刘生乾。

* 1. 后端部分

Web应用后端部分由成子谦、赛炜城、刘润生同学负责。后端组组长为成子谦。

* 1. 接口设计

接口设计部分由邝碧霞、成子谦同学负责。

1. **个人主要工作**
   1. 后端逻辑设计

NFA合法性判定算法如下：

后端从前端得到传入的学生构造的NFA，构造字符表并给节点重新排序后，从源点出发，迭代遍历全图并构建DFA，再与原正则表达式所对应的DFA进行一一比对，看是否完全相同。若相同，则学生所构建的NFA合法，反之不合法。

这样做是因为可以从一个正则表达式构建若干可能NFA，但一个正则表达式只对应一个DFA。通过把NFA转换为DFA，再利用DFA的唯一性，可以对NFA的合法性进行判定。虽然从NFA到DFA的转换开销较大，但由于该应用的实际使用场景较简单，用户输入的正则表达式并不会造成巨大压力，响应速度依然能维持在较快水平，故可以接受此种做法。

* 1. 接口设计

前后端交互通过Ajax技术实现。前端通过传入学生构造的NFA（具体表现为图的节点及边），通过回调函数（与提交按钮绑定）从后端得到返回结果，并对学生进行结果反馈。

* 1. 软件著作权

目前项目以及完成，仍处于软件著作权申请流程中，需等待下一步消息。

1. **更多研究**
   1. 对下一步工作的设想

目前，本项目“词法分析”功能已经接近尾声。可以考虑继续完善该功能，或考虑进行“语法分析”部分的迭代开发。

对于“词法分析”功能，可考虑进行如下优化：

1. 语法高亮。对学生输入的正则表达式，实时予以语法高亮，提高交互体验。
2. “连线练习”绘图逻辑优化。应支持重复操作，且生成节点时应避免节点间覆盖问题。
3. Tex（一个先进的词法分析工具）的使用。

对于“语法分析”功能，可从如下角度入手：

1. 上下文无关文法的解释与构造（最左推导、最右推导等）。
2. 下推自动机的构造。
3. Top-down / Bottom-up分析法。

然而“语法分析”为《编译原理》这一课程的教学难点。预计开发时间较长、流程较复杂，需要做出更详细的规划。

1. **感想与小结**

通过本次SRP项目，我第一次接触到web应用的开发、维护流程，对web应用开发技术有了更深一步的了解，学习到Node.js、git、图论相关算法以及前后端接口设计等技术，意识到一个好的项目离不开大胆的构想、详细的构思、规范的管理和谨慎的测试，让我对项目开发的具体流程有了更深的理解。