# 虚拟仿真实验一 正则表达式到NFA

1. **实验目的**

* 熟悉编译原理虚拟仿真实验环境的基本使用方法。
* 掌握正则表达式到NFA的转换。

1. **实验内容**
   1. **编译原理虚拟仿真实验软件**

编译原理虚拟仿真实验软件是基于VSCode编程环境进行高度定制化的版本，方便教师和学生使用。在定制版的VSCode编程环境中集成了Make构建工具、GCC编译器、GDB调试器等必要工具，同时包含了一些有用的VSCode插件，其中最重要就是可以将编译原理的概念使用图形化功能渲染到窗口上的虚拟仿真插件。

编译原理虚拟仿真实验软件可以免去学生手工构建实验环境所带来的学习成本，使学生可以将主要精力放在对编译原理的分析与理解上，并且通过虚拟仿真手段让学生对枯燥、难懂的编译原理知识可以有一个直观的、形象的认识。

**安装和启动**

学生首先需要将编译原理虚拟仿真实验软件的压缩包解压缩到64位Windows 7或Windows 10本地磁盘的一个目录中（例如D:\vscode，注意目录路径不要包含中文字符或者空格），然后，学生就可以双击D:\vscode\code.exe文件启动VSCode了。

**窗口布局**

软件启动后的窗口布局由下面的若干元素组成：

* 编辑器：这是主要的代码编辑区域，可以多列或者多行的打开多个编辑器。
* 侧边栏：位于左侧的侧边栏包含了文件资源管理器、文件搜索、源代码版本管理、调试与运行、插件等基本视图。
* 活动栏：位于侧边栏的左侧，可以方便的让用户在不同的视图之间进行切换。
* 状态栏：位于底部的状态栏用于显示当前打开文件的光标位置、编码格式等信息。
* 面板：编辑器的下方可以展示不同的面板，包括显示输出信息的面板、显示调试信息的面板、显示错误信息的面板和集成终端。面板也可以被移动到编辑器的右侧。

**打开虚拟仿真可视化窗口**

选择VSCode的“View”菜单中的“Command Palette...”或者按Ctrl+Shift+P快捷键，在VSCode的顶部中间位置打开命令面板，输入命令“Lab: New Visualizer View”后按回车,会在右侧打开数据可视化窗口，在后面的演示过程中，学生就可以使用此窗口查看虚拟仿真图形了。

* 1. **实验项目文件夹**

本实验提供了配套的实验源代码文件，这些文件放置在文件夹“NFA”中。

**打开实验项目文件夹**

选择VSCode的“File”菜单中的“Open Folder...”菜单项，在打开的窗口中选中本实验对应的文件夹“NFA”，然后点击“选择文件夹”按钮，VSCode 会自动打开项目文件夹。文件夹中的文件会在左侧的“文件管理器”窗口中列出。

**阅读实验源代码**

该实验包含了三个头文件“RegexpToNFA.h”、“RegexpToPost.h”、“NFAFragmentStack.h”和三个C源文件“main.c”、“RegexpToPost.c”、“NFAFragmentStack.c”。下面对这些文件的主要内容、结构和作用进行说明：

**main.c文件**

在左侧的“文件管理器”窗口中双击“main.c”打开此文件。此文件主要包含了以下内容：

1. 在文件的开始位置，使用预处理命令包含了RegexpToNFA.h、RegexpToPost.h和NFAFragmentStack.h文件。
2. 定义了main函数。在其中实现了栈的初始化。然后，调用了re2post函数，将正则表达式转换到解析树的后序序列。最后，调用post2nfa函数，将解析树的后序序列转换到NFA。
3. 在main函数的后面，定义了函数CreateNFAState和MakeNFAFragment，这两个函数分别是用来创建一个新的NFA状态和构造一个新的Fragment。接着定义了函数post2nfa，关于此函数的功能、参数和返回值，可以参见其注释。在这个函数中用‘$’表示空转换，此函数的函数体还不完整，留给学生完成。

**RegexpToPost.c文件**

在“文件管理器”窗口中双击“RegexpToPost.c”打开此文件。此文件主要包含了以下内容：

1. 在文件的开始位置，使用预处理命令包含了RegexpToPost.h文件。
2. 定义了re2post函数，此函数主要功能是将正则表达式转换成为解析树的后序序列形式。

**NFAFragmentStack.c文件**

在“文件管理器”窗口中双击“NFAFragmentStack.c”打开此文件。此文件主要包含了以下内容：

1. 在文件的开始位置，使用预处理命令包含了NFAFragmentStack.h文件。
2. 定义了与栈相关的操作函数。在构造NFA的过程中，这个栈主要用来放置NFA片段。

**RegexpToNFA.h文件**

在“文件管理器”窗口中双击“RegexpToNFA.h”打开此文件。此文件主要包含了以下内容：

1. 包含用到的C标准库头文件。目前只包含了标准输入输出头文件“stdio.h”。
2. 包含其他模块的头文件。目前没有其他模块的头文件需要被包含。
3. 定义了与NFA相关的数据结构，包括NFA状态NFAState和NFA片段NFAFragment。具体内容可参见下面的两个表格。

|  |  |
| --- | --- |
| **NFAState的域** | **说明** |
| Transform | 状态间转换的标识。用 '$' 表示 'ε-转换'。 |
| Next1和Next2 | 用于指向下一个状态。由于一个NFA状态可以存在多个转换，而在本程序中使用的是类似于二叉树的储存结构，每一个状态最多只有两个转换，所以，这里定义两个指针就足够了。当NFA只有一个转换时，优先使用Next1，Next2赋值为NULL。 |
| Name | 状态名称。使用整数表示（从1开始），根据调用CreateNFAState函数的顺序依次增加。 |
| AcceptFlag | 是否为接受状态的标志。1 表示是接受状态 0 表示非接受状态。 |

|  |  |
| --- | --- |
| **NFAFragment的域** | **说明** |
| StartState | NFAFragment的开始状态。 |
| AcceptState | NFAFragment的接受状态。在构造NFA的过程中总是在NFA的开始状态和接受状态上进行操作，所以用开始状态和接受状态表示一个NFA片段就足够了。 |

1. 声明函数和全局变量。

**RegexpToPost.h文件**

在“文件管理器”窗口中双击“RegexpToPost.h”打开此文件。此文件主要包含了以下内容：

1. 包含其他模块的头文件。目前只包含了头文件“RegexpToNFA.h”。
2. 声明函数。为了使程序模块化，所以将re2post函数声明包含在一个头文件中再将此头文件包含到“main.c”中。

**NFAFragmentStack.h文件**

在“文件管理器”窗口中双击“NFAFragmentStack.h”打开此文件。此文件主要包含了以下内容：

1. 包含其他模块的头文件。目前只包含了头文件“RegexpToNFA.h”。
2. 定义重要的数据结构。定义了与栈相关的数据结构。
3. 声明函数。声明了与栈相关的操作函数。

|  |
| --- |
| 提示：请学生认真理解这部分内容，其他实验题目中的源代码文件也严格遵守这些约定，如无特殊情况将不再进行如此详细的说明。 |

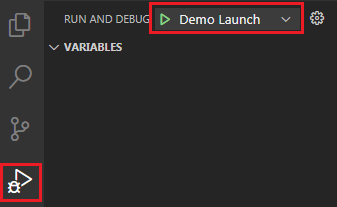
* 1. **使用虚拟仿真功能进行演示**

在学生编写代码之前，可以使用虚拟仿真功能查看程序的执行过程，并在“数据可视化”窗口中以图形化的方式查看内存中的数据变化情况。

按照下面的步骤演示项目：

1. 在左侧“活动栏”中点击“Run and Debug”按钮，在侧边栏中打开了Run and Debug窗口，在顶部选择“Demo Lauch”项，如图1所示。

|  |
| --- |
| 提示：在Run and Debug顶部的下拉框中，如果选择Demo Lauch，表示运行的是演示程序main.demo，如果选择Lauch，表示运行的是学生编写的程序main.c。 |

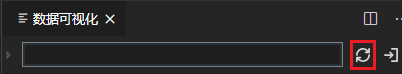


**图1：Run and Debug**

1. 选择“Run”菜单中的“Start Debugging”或者按F5快捷键，启动调试，自动打开了main.demo文件，并中断执行。中断所在的行会使用特殊颜色标识出来，表示程序在对应的行停止运行了（注意，此时中断所在的行还没有运行，表示的含义是接下来会运行中断所在的行）。学生应该注意到，在main.demo文件中提供了C语言代码和伪代码混合的内容，目的是将程序执行的流程演示出来。

|  |
| --- |
| 注意：不要修改main.demo文件中的内容！ |

1. 在“数据可视化”窗口中，点击“刷新”按钮，可以查看正则表达式等相关的图形化内容。



**图4：刷新“数据可视化”窗口**

1. 按F5继续调试，可以查看“数据可视化”窗口的变化以及main.demo中执行的过程。
2. 在“数据可视化”窗口，由于正则表达式转NFA包含2幅图（默认显示的是#1的图形），在顶部的文本框中输入“#1”按回车，显示正则表达式和NFA；在文本框中输入“#2”按回车，显示正则表达式的解析树。由于正则表达式的解析数是固定不会变化的，所以学生可以在查看#2图后切换会#1图继续后面的演示过程。



**图5：“数据可视化”窗口内容**

1. 学生可以按F5继续调试，在调试的过程中，每执行main.demo文件中的一行代码后，仔细观察“数据可视化”窗口内容所发生的变化，例如构造单字符 NFA 片段，构造连接NFA片段等，理解NFA片段构造的执行过程。
2. 待按F5一直调试到post2nfa函数最后的return语句时，就可以在数据可视化窗口的最下方看到构造完成的NFA图形了。继续按F5调试过程就运行完毕了，此时在VSCode底部的TERMINAL窗口中还会以文本形式打印输出NFA，这个工作是通过在main函数的最后调用OutputResult函数来完成的。

**更改演示用的正则表达式**

在演示过程结束后，学生可以更改演示用的正则表达式，查看不同正则表达式转换到NFA的过程。本实验使用的正则表达式包含在input1.txt～input8.txt共8文件中，演示过程默认使用的是input1.txt文件中的正则表达式。

更改演示用的正则表达式的方法是，在左侧的“文件管理器窗口”中，展开.vscode文件夹,然后双击lauch.json文件打开，在"name"为 "Demo Launch"的配置部分“args”的值为["<", "input1.txt"]，表示在调试演示程序时将标准输入重定向到了input1.txt文件，所以在main函数开始位置的scanf函数就从该文件中读取正则表达式的字符串了。学生将input1.txt替换为其他input文件的名称，保存lauch.json文件后即可按F5启动查看其它正则表达式转换为NFA的演示过程了。



**图3：修改重定向输入文件**

* 1. **编写代码**

之前演示流程使用的是main.demo文件中的内容。学生如果要自己实现类似的功能，需要修改main.c文件，文件main.c中的post2nfa的函数体还不完整，需要学生补充完整。要求学生编写的代码能够正确处理input1.txt～input8.txt文件中的所有正则表达式。

**提示：**

* 在main.demo文件中的post2nfa函数中，可以方便的查看该函数的演示流程。
* 参考演示流程中构造NFA片段的描述，其中给出了构造单字符NFA片段和连接NFA片段的源代码，这两步操作可以完成对input1.txt 中正则表达式到NFA的转换（转换后的NFA如图8所示），学生可以一边调试演示程序一边理解源代码的执行过程，在此基础上完成其他形式NFA片段的构造。
* 对于问号的NFA片段的构造（如图11所示），原则上也可以将状态1作为开始状态，状态2作为接受状态，并通过ε-转换来表达接受状态为空的情况。但是如果存在一个或多个NFA片段与问号NFA片段相连接的情况，对于上述的情况就不适用了。因为在本程序中使用的是类似于二叉树的存储结构，一个状态最多只有两个转换指针，所以，必须在原来的基础上再添加两个状态作为开始状态和接受状态。

表示开始状态；表示一般状态；表示接受状态；

**图6：NFA状态图例。**



**图7：表示单字符的NFA片段。**



**图8：表示连接的NFA片段（对应input1.txt中的正则表达式）。**



**图9：表示选择的NFA片段（对应input2.txt中的正则表达式）。**



**图10：表示星号的NFA片段（对应input3.txt中的正则表达式）。**



**图11：表示问号的NFA片段（对应input4.txt中的正则表达式）。**



**图12：表示加号的NFA片段（对应input5.txt中的正则表达式）**

**生成项目**

学生编写完毕代码后，需要选择“Terminal”菜单中的“Run Build Task...”或者按Ctrl+Shift+B快捷键，在打开的列表中选择“生成项目”，如果存在语法错误会在底部的TERMINAL窗口和PROBLEMS窗口中进行提示，学生可以根据这些提示信息修改语法错误，直到可以正确生成项目。生成项目成功后，在资源管理器中可以查看已生成的app.exe文件。

|  |
| --- |
| 提示：生成项目的过程，就是使用项目中的makefile文件将项目所包含的每个C源代码文件（.c文件）编译为一个对象文件（.o文件），然后再将多个对象文件链接为一个目标文件（.exe文件）的过程。以本实验为例，成功生成项目后，默认会在本项目文件夹中生成 “main.o”文件“RegexpToPost.o”文件“NFAFragmentStack.o”文件和 “app.exe”文件。 |

**更改运行的正则表达式**

学生可以更改正则表达式，确保编写的程序可以正确处理不同的正则表达式。本实验使用的正则表达式包含在input1.txt～input8.txt共8文件中，默认使用的是input1.txt文件中的正则表达式。

更改正则表达式的方法是，在左侧的“文件管理器窗口”中，展开.vscode文件夹,然后双击lauch.json文件打开，在"name"为 " Launch"的配置部分“args”的值为["<", "input1.txt"]，表示在运行学生编写的程序时将标准输入重定向到了input1.txt文件，所以在main函数中就从该文件中读取正则表达式的字符串了。学生将input1.txt替换为其他input文件的名称，保存该文件后即可。

**启动运行**

按照下面的步骤运行学生编写的源代码：

1. 在“活动栏”的左侧点击“Run and Debug”按钮，在侧边栏中打开了Run and Debug，在顶部选择“Lauch”项。这样才会运行学生编写的源代码，而不是演示用的main.demo文件。
2. 启动调试的方法是选择“Run”菜单中的“Start Debugging”或者按F5快捷键。启动调试后， 开始运行学生编写的源代码。
3. 程序运行完毕后，会将生成的NFA输出到TERMINAL窗口中。学生可以将自己程序输出的NFA，与演示程序输出的NFA进行比较，如果完全一致，说明学生编写的源代码是正确的，否则学生需要继续修改源代码，直到所有的正则表达式都可以正常处理。在output1～output8.txt共8文件中包含了使用对应input文件中的正则表达式生成的NFA，学生也可以使用output文件中的内容与自己编写的程序生成的NFA进行比较，确保自己编写的程序生成的正则表达式是正确的。
4. 在运行学生编写的源代码时，学生也可以在源代码中添加断点，并进行单步调试，在调试的过程中也可以刷新“数据可视化”窗口，查看图形化数据，通过与演示过程中的图形化数据进行比对，也可以帮助学生发现自己编写的代码中的错误。