Complier_Lab1_Report

161250055 李安迪

Motivation / Aim

通过编写REs->NFAs, NFAs->NFA, NFA->DFA, DFA->DFAmin的代码，回顾课上所讲的词法分析过程。

Content Description

本实验提交包含以下内容：

- maven包，主要包含：
  - src/main/java/Lexical 中的拟lex程序
  - src/main/resources中的词法定义文件`xxrule.txt`和对应的输入文件`xx.txt`
  - lex生成的`xx_generationfile`文件
  - 输出文件`xx_res.txt`
  - 实验报告 (pdf)

Ideas / Methods

进行词法分析有两种实现方法：

1. 首先自己确定一些类别的单词对应的正规表达式REs，然后,手工将这些REs变为小的NFAs，再将小的NFA手工合并为一个大的NFA，接着，将大的NFA手动确定化、最小化。最后，基于手工构造出来的具有最少状态的DFA，手工编写相应的词法分析程序。后面再通过一个程序片段作为手工编写的词法分析程序的输入，输出相应程序片段中的单词Token序列。
2. 首先构造一个以.l文件为输入，词法分析程序为输出的词法分析程序的生成程序。该词法分析程序的生成程序就是类似课堂上所讲的Lex。Lex的核心程序就是将REs转换为具有最少状态的DFA的程序。然后，利用所编写的能生成词法分析程序的程序，生成相应的词法分析程序（对应此词法分析程序的RE是在相应的.l文件中定义）。最后，验证所生成的词法分析程序。

本实验简单模拟第二种实现方法，包含以下步骤：

1. 从 `rule` 词法描述文件中读取 token-name 和 RE
2. 从 RE 生成 NFA
3. 合并多个 NFA 为一个大 NFA
4. 从大 NFA 生成 DFA
5. 合并等效的 DFA 以节约空间
6. 从 DFA 生成词法分析 Java 程序

Assumptions

- 正则表达式支持以下运算符：
  - `()` 括号
  - `|` 表示“或”关系

    - `*` 表示出现 0 或多次
- 不支持对空格，和含转义字符的字符匹配
- `rule` 文件中的正则表达式，越靠后则优先级越低

Related FA descriptions

`rule` 文件由两部分组成：

- 首行：允许出现的字符
- 其他行：RE所对应的token-name 需要匹配的 RE

示例：

```
ab
single (a|b)*abb
```

Description of important Data Structures

以下列出lex部分主要数据结构。

```
/**正则表达式**/
public class RE {
    public String content;
}
```

```
/**NFA节点**/
public class NFANode implements Serializable {
    /**是否被访问过，遍历时使用**/
    boolean visited = false;
    /**由出边数目决定的节点状态
    1代表只有一条出边，此时char为出边上的字母，
    0代表有两条epi出边
    -1代表终结态
    2表示有多条epi出边
    -2表示未指定其状态
    **/
    private int state = -2;
    /**出边所指向的其他节点**/
    private ArrayList<FANode> outnodes = new ArrayList<>();
    /**出边字符，由于thompson算法的特殊性，每个节点如果有非\epsilon的出边，则该节点只有一条出边**/
    private char c = '\0';
    /**指向的终态**/
    private FANode endAt;
    /**若节点为终态，节点名**/
    private String name = "";
}
```

```
/**DFA状态**/
public class Dstate implements Serializable {
    /**标记在DFA最小化最后一次分划中该状态所属状态组在状态组列表中的位置**/
    private int tag;
    /**\epsilon闭包**/
    private List<NFANode> epiclosure = new ArrayList<>();
    /**状态转换Dtrans**/
    private Map<Character, Dstate> Dtrans = new HashMap<>();
     /**若状态为终态，状态名**/
    private String name = "";
}
```

Description of core Algorithms

- RE 到 NFA 使用课上讲授的 Thompson算法
- NFA 到 DFA 使用子集构造法
- DFA 的简化通过状态最小化算法

Use cases on running

单一测例：

```
abb
```

处理结果：

```
<single,abb>
```

对应`rule`：

```
ab
single (a|b)*abb
```

初级测例：

```
abba
```

处理结果：

```
<2,abb>
<1,a>
```

对应`rule`：

```
ab
1 a
2 abb
3 a*b
```

正常测例：

```
package Lexical;

public class Input {
    int i = 0;
    i += 1;
}
}
```

处理结果：

```
<reservedWords,package>
<variable,Lexical>
<end,;>
<reservedWords,public>
<reservedWords,class>
<variable,Input>
<block,{>
<reservedWords,int>
<variable,i>
<operator,=>
<number,0>
<end,;>
<variable,i>
<operator,+=>
<number,1>
<end,;>
<block,}>
```

异常测例：

```
package Lexical;\

public class Input {
    int i = 0;
    i += 1;
}
}
```

处理结果：

```
<reservedWords,package>
<variable,Lexical>
<end,;>
no matched pattern with char \at line 1
```

对应`rule`：

```
abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789+-=;{}
reservedWords (package)|(public)|(private)|(protected)|(class)|(int)
variable
(a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z|A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|
R|S|T|U|V|W|X|Y|Z)
(a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z|A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|
R|S|T|U|V|W|X|Y|Z|0|1|2|3|4|5|6|7|8|9)*
number (1|2|3|4|5|6|7|8|9)*(0|1|2|3|4|5|6|7|8|9)
operator =|(+=)|(-=)
end ;
block {|}
```

Problems occurred and related solutions

- 问题：

  使用ObjectInputStream出现java.io.StreamCorruptedException: invalid stream header EFBFBDEF 异常

  解决方法：使用二进制流初始化对象流，使用Base64做二进制流的encode和decode

- 问题：

  使用thompson算法连接两个小自动机时，如果按照龙书上将前一个小自动机的末状态和后一个小自动机的起始状态重合，工作量较大。

  解决方法：直接连接两个小自动机。

Your feelings and comments

- lex部分最好不用持久化文件的形式存储状态转换图，而是直接存成代码模板。
- 想要实现一个实际可用且高效的词法分析程序实在太难了，还有很多东西需要深入理解和课后学习。
- 数据结构和算法应该学得再好一点，算法写的有点啰嗦。在设计时要多权衡“空间换时间”和”时间换空间“的策略。
- 即使并不是工程项目，也应该先做一些整体的设计再开始写代码。