FLA 大作业实验报告

221220092 谷莘

一、PDA 部分

1. 解析器

解析器 loadPDAFromFile 实现的逻辑如下：

从指定文件中读取 PDA 配置，通过文件流读取每一行数据，跳过空行和注释行，并且使用正则表达式和字符串操作解析状态集(Q)、输入符号集(Σ)、栈符号集(Γ)、初始状态(q0)、栈起始符号(z0)和终结状态集(F)（分别对应 states, inputSymbols, stackSymbols, initialState, stackStartSymbol, finalStates, transitions）；

检查每个部分的格式正确性，确保它们被正确解析和存储在 PDA 类的相应属性中。

将转移函数(δ)存储到 PDA 对象的 transitions 向量中。

对不正确的语法格式，我们会给出 syntax error 的报错（支持打印错误位置的模式）：

```
void reportError(const std::string& message, int lineNumber) {
    std::cerr << "Error on line " << lineNumber << ": " << message << std::endl;
    std::exit(1);
}

void judge_(bool condition, const std::string& message, int lineNumber) {
    if (!condition) {
        //reportError(message, lineNumber);
        cerr << "syntax error" << endl;
        ::exit(1);
    }
}
```

2. 模拟器

模拟器用解析器解析出的 PDA 类进行模拟，PDA 类的方法 simulate 的逻辑如下：

使用 simulate 方法接受输入字符串，并根据 PDA 的定义进行处理；

初始化栈，压入栈起始符号；

逐字符读取输入字符串，检查每个字符是否在输入符号集中；

根据当前状态、输入符号和栈顶符号遍历转移函数，寻找匹配的转移；

更新当前状态和栈内容，对ε转移（用"_"表示），对遍历的 index 不增加；

模拟结束后，检查当前状态是否为终结状态，并确认栈是否为空，以判断输入字符串是否被接受；

3. Verbose 模式

实现了加分点 PDA 模拟的 verbose 模式，打印的内容包括当前输入字符、状态和栈内容；在输入非法时，能定位非法输入的位置，效果演示如下：

```
(base) lier@LAPTOP-0IRRTPKQ:/mnt/d/形式语言与自动机/project-2024$ ./bin/fla -v ./pda/case.pda "()"
Input: ()
================== RUN ==================
state : q
stack: Z(
-----------------------------------------
state : q
stack: Z
-----------------------------------------
state : accept
stack:
-----------------------------------------
true
================== END ==================
```

```
(base) lier@LAPTOP-0IRRTPKQ:/mnt/d/形式语言与自动机/project-2024$ ./bin/fla -v ./pda/case.pda "((a))()"
Input: ((a))()
================== ERR ==================
error: 'a' was not declared in the set of input symbols
Input: ((a))()
         ^
================== END ==================
illegal input
```

二、TM 部分

1. 解析器

解析器在 TM 中的 readFile 方法中实现，大致逻辑如下：

从指定文件中读取 TM 配置；

解析状态集、输入符号集、磁带符号集、初始状态、终结状态集、磁带数量和转移规则（分别对应 states, stateLookupTable, inputSymbols, tapeSymbols, currentState, finalStates, numberOfTapes, tapeCollection, transitionRules）；

使用字符串流和分割函数解析各部分内容（使用实例 splitString）；

检查每个部分的格式正确性，确保它们被正确解析和存储在 TM 类的相应属性中；

对不正确的语法格式，我们会把判断扔给 judge_函数，如果对应的语法条件不满足，我们会在 cerr 中抛出 syntax error:

```cpp
void judge_(bool condition) {
    if (!condition) {
        cerr << "syntax error" << endl;
        ::exit(1);
    }
}
```

2. 模拟器

我们除了图灵机所代表的类 TM，还创建了纸带类 Tape、transition 对应的类 Rule 作为子类，模拟时，通过调用 TM 的 run 类进行模拟，run 中对输入错误进行检查，随后调用 executeTuringMachine 类启动模拟，其流程如下：

初始化磁带（Tape），并设置初始状态；

在每个步骤中，根据当前状态和磁带符号选择合适的 transition，并应用；

更新当前状态、磁带内容和磁头位置；

检查是否达到终结状态，以判断输入字符串是否被接受;

3. Verbose 模式

我们完成了 verbose 模式，并且完成了对齐要求，效果如下：

```
================== RUN ==================
Step   : 0
State  : start
Index0 : 0 1 2 3
Tape0  : 1 1 1 1
Head0  : ^
Index1 : 0
Tape1  : _
Head1  : ^
-----------------------------------------
Step   : 1
State  : init
Index0 : 0 1 2 3
Tape0  : 1 1 1 1
Head0  : ^
Index1 : 0
Tape1  : 1
Head1  : ^
-----------------------------------------
```

```
-----------------------------------------
Step   : 16
State  : t3
Index0 : 4 5 6 7
Tape0  : t r u _
Head0  :       ^
Index1 : 0 1 2 3
Tape1  : 1 1 1 _
Head1  :       ^
-----------------------------------------
Step   : 17
State  : final
Index0 : 4 5 6 7
Tape0  : t r u e
Head0  :       ^
Index1 : 0 1 2 3
Tape1  : 1 1 1 _
Head1  :       ^
-----------------------------------------
Result: true
================== END ==================
```

```
(base) lier@LAPTOP-0IRRTPKQ:/mnt/d/形式语言与自动机/project-2024$ ./bin/fla -v ./tm/case2.tm 1111a
Input: 1111a
================== ERR ==================
error: 'a' was not declared in the set of input symbols
Input: 1111a
          ^
================== END ==================
```

三、样例编写

1. case.pda

这一 PDA 的目标是匹配任意可能的括号组合。我的思路是将所有左括号直接入栈，碰到右括号消除；如果右括号没有与之匹配的左括号，拒绝；如果正好栈空，我们接受。

#Q = {q,accept,reject}

#S = {(,)}

#G = {(,),Z}

#q0 = q

#z0 = Z

#F = {accept}

q (Z q (Z

q ((q ((

q) (q _

q) Z reject Z

q _ Z accept _

2. case1.tm

这个 TM 的作用是对给出的 a^ib^j,打印出 c^(i*j);我这里给出我的大致思路，具体过程见代码：

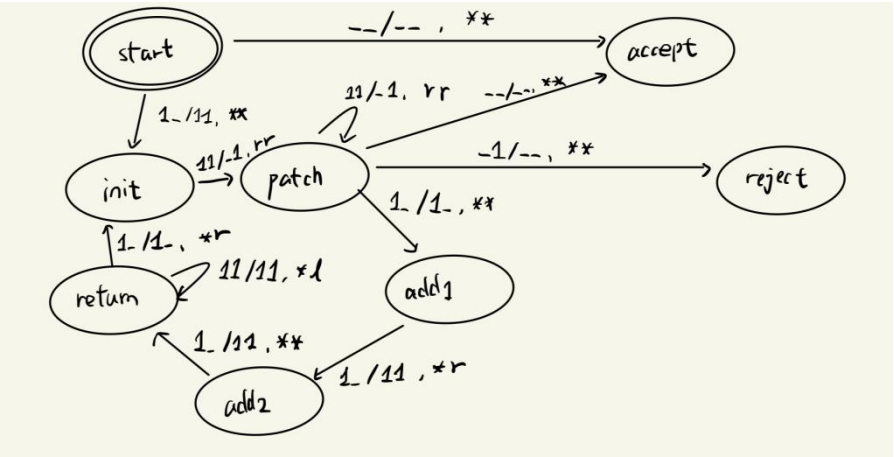
我一共用了三条纸带，第一条用来存结果，第二条用来放置 a,第三条放置 c。我们开始时读取到 a,我们将它搬到纸带 2 上；当读到 b 时，我们将与纸带 2 上 a 相同数量的 c 放在纸带 3 的后端；在读取 b 时如果有碰到 a,我们打印 illegal_input；在都读取完时，我们将纸带 3 上的 c 搬到纸带 1 上即可；

3. case2.tm

这个 TM 的作用是数平方数个数的 1，我的思路如下：

我的思路本质是在原来的 1 计数上依次减去 1,3,5,7……每次减的时候，如果正好够减那么是平方数；如果 1 还剩了，那么被减数加 2 继续减；如果 1 不够减，那么不是平方数。

这里放出一张设计简图：


start
accept
init
patch
reject
return
add1
add2
--/-- , **
1_/11, **
11/-1, rr
11/-1, rr
--/--, **
-1/--, **
1_/1_, **
1_/1_, *r
11/11, *1
1_/11, **
1_/11, *r