《计算理论初步》课程项目报告

项目名称：标准图灵机模拟器

、

姓名：李颖

学号：161220072

2016 级 计算机科学与技术 系 3 班

任课老师：卜磊

邮箱：161220072@smail.nju.edu.cn

时间：2018/12/28

一、任务描述

在正确解析读入的图灵机程序的基础上，对输入串作合法性判定，对合法的输入串（纸带）进行模拟运行，并给出图灵机运行的瞬时描述和判定是否被接收的结果。

二、分析与设计思路

1、数据结构设计

在对图灵机程序的解析中，需要存储输入符号集（供合法性判定使用），初始状态、终结状态集和转移函数（供图灵机运行使用）。

输入符号集合和终结状态集合可用 List 存储，而对于转移函数，其使用是通过搜索旧状态和旧符号来得到新状态、新符号和方向，为了提高搜索效率，可以使用 HashMap，直觉上是将旧状态和旧符号组成的对象作为 key。在 java 中，用自定义类作为 key，必须重写 equals()和 hashCode()方法，用起来并不那么简单。同时需要注意的是，图灵机语法中存在通配符，旧状态可能为任意匹配，所以为了实现的方便，可以只将旧状态作为 key，相同旧状态的转移函数组成的 List 作为 value，这样查找就是先得到该状态下的所有转移函数，优先找能精确匹配的，若找不到，再查找有通配符匹配的。

在图灵机运行过程中，还有一个重要的部分，那就是纸带。纸带的假设是无限长的，也就是说纸带的两边是可无限拓展的，两个栈是可以模拟无限纸带的，更为直接的做法是用字符串表示纸带，左右的扩展通过字符串的连接来实现。

2、模拟器单步运行

对图灵机模拟器进行模块划分，将每一次的状态改变作为一个单独的函数，也即单步模拟器，对每个输入串不断调用 SingleExecutor 直到其返回值为 False。

```
boolean SingleExecutor(int step, OutputStreamWriter writer)
```

SingleExecutor 功能及实现：查表进行状态迁移和纸带读写，在没有下一步的迁移的情况下返回 False。在实现中，根据当前读写头所指向的纸带内容和当前状态寻找相应的转移，将读写头所指向的地方改写为新符号，状态也改为新状态，如果方向为 left，读写头往左移动一格；方向为 right，则向右移动一格；方向为 *，则不移动。

当读写头指向了当前字符串表示的纸带之外的地方，进行扩展。如果是纸带的左边，便将一个空格符与原纸带连接起来作为新纸带，如果是右边，便将原纸带与一个空格符连接起来作为新纸带。

为了方便打印纸带内容，需要一个变量记住纸带 index 为 0 的字符在纸带字

符串中的位置，这两个 index 在开始状态是一样的，只有在纸带向左扩展时才会出现变化：纸带最左边扩展一个位置，原来纸带 index 为 0 的字符在字符串中也向右移动一位置，那么该变量需要加一。

3、接收/拒绝判别

图灵机程序运行结束后需要给出判定：输入串是否在图灵机描述的语言中。对于 Project 中的标准图灵机设定，final state 后不再有转移函数，所以判别是否在该语言中很简单，只要看程序终止时是否在 final state。

4、图灵机程序语法错误检测

对于输入的图灵机程序，有可能会出现语法上的错误，在本实验中考虑的语法错误主要有：

（1）图灵机定义不全：TM 的七个组成部分（状态集、输入符号集、纸带符号集、开始状态、接受状态集合、转移函数）有缺失。

例如，当 tm 文件中缺少#q0 = XX 行时报错

```
==================== ERR ====================
        Cannot find the start state in program
==================== END ====================
```

（2）转移函数中出现未定义的状态，报错格式如下：

```
==================== ERR ====================
Unexpected new symbol ‘s’ in transition functions
==================== END ====================
```

（3）转移函数中出现未定义的符号，报错格式如下：

```
==================== ERR ====================
  Unexpected state ‘st’ in transition functions
==================== END ====================
```

5、两个图灵机判定程序的设计

（1） $L_1 = \{1^m x 1^n = 1^{mn} \mid m,n \in N^+\}$

L_1 表示的是形如 1..1x1..1=1..1 的输入串，其中‘x’和‘=’将输入串分割为三个 1-串，且最后一个连续 1--串的长度是前两个 1-串长度的乘积。要判定输入串是否属于该语言就是要判断这三个连续 1-串的长度关系。

m*n 可以表示为 m 个 n 的和，顺着这样的想法，每当在第一个 1-串有一个 1，就去第三个 1-串中消去 n 个 1。某个时刻对第三个 1-串组消去之后第一个 1-串和第三个 1-串均为空，说明该输入串符合该语言，图灵机应该接受，不然的话，图灵机就拒绝。

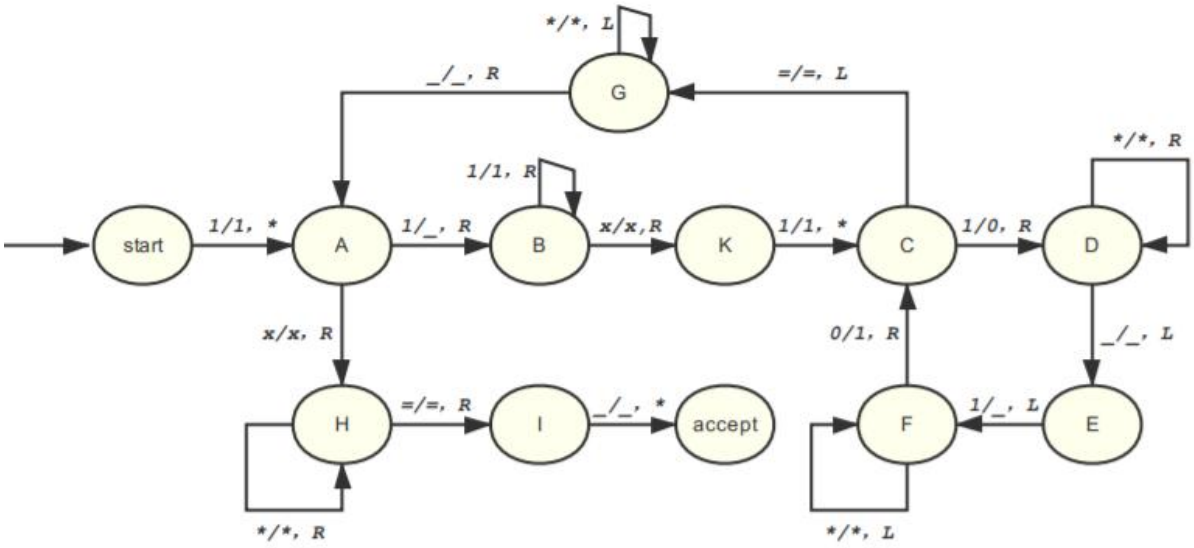
进一步思考：如何消去 n 个 1？

图灵机虽然不能记住 n，但是存在一个长度为 n 个 1-串。简单来想，对于第二

个和第三个 1-串，每在第二个串中消去一个 1，就在第三个串中消去一个 1。这样，第二个串为空时第三个串减少了 n 个 1。但是第二个串是要反复使用的，不应当被消去，常见的做法是使用不同的符号标记被“消去”的串，于是引入一个新的纸带符号‘0’,每当消去第二个串中的 1 时就将 1 改为 0，从串的最左边开始。不仅如此，0 还可以作为 mark 来标记下一次消去的开始位置。每次向左定位到 0 时，0 的 mark 功能结束，可以将该位置的 0 恢复为 1 以便于下一次的使用，接着就可以将下一个 1 改为 0 作为 mark。

实现总述：磁头从最左边开始，遇到一个‘1’便消去，磁头往右移动到 x 后，出现一个‘1’便将之标记为‘0’，磁头移动到整个串的最后消去一个‘1’，接着回到上一次标记‘0’的地方，恢复为‘1’并向右移动一格，判断是否为‘1’，若是重复上述操作，若不是‘1’而是‘=’，则磁头移动到最左边，重复一开始的操作。

画出在不考虑最终在纸带上打印 True 或 False 的情况下的转移图：



描述各状态含义：

start:开始状态,检测第一个是否为‘1’(m > 0)

A: 消去第一个 1-串中的一个‘1’

B: 往右移动直到遇到‘x’

C: 将第二个 1-串中的一个‘1’标记为‘0’

D: 往右移动直到遇到空符号

E: 消去第三个 1-串中的最后一个‘1’

F: 向左移动找到标记‘0’停下

G: 对第三个 1-串消去 n 个‘1’结束，返回到非空纸带最左边

H: 第一个 1-串消去完毕，往右移动直到遇到‘=’

I: 检测第三个 1-串是否为空

K: 检测‘x’后是否为‘1’(n > 0)

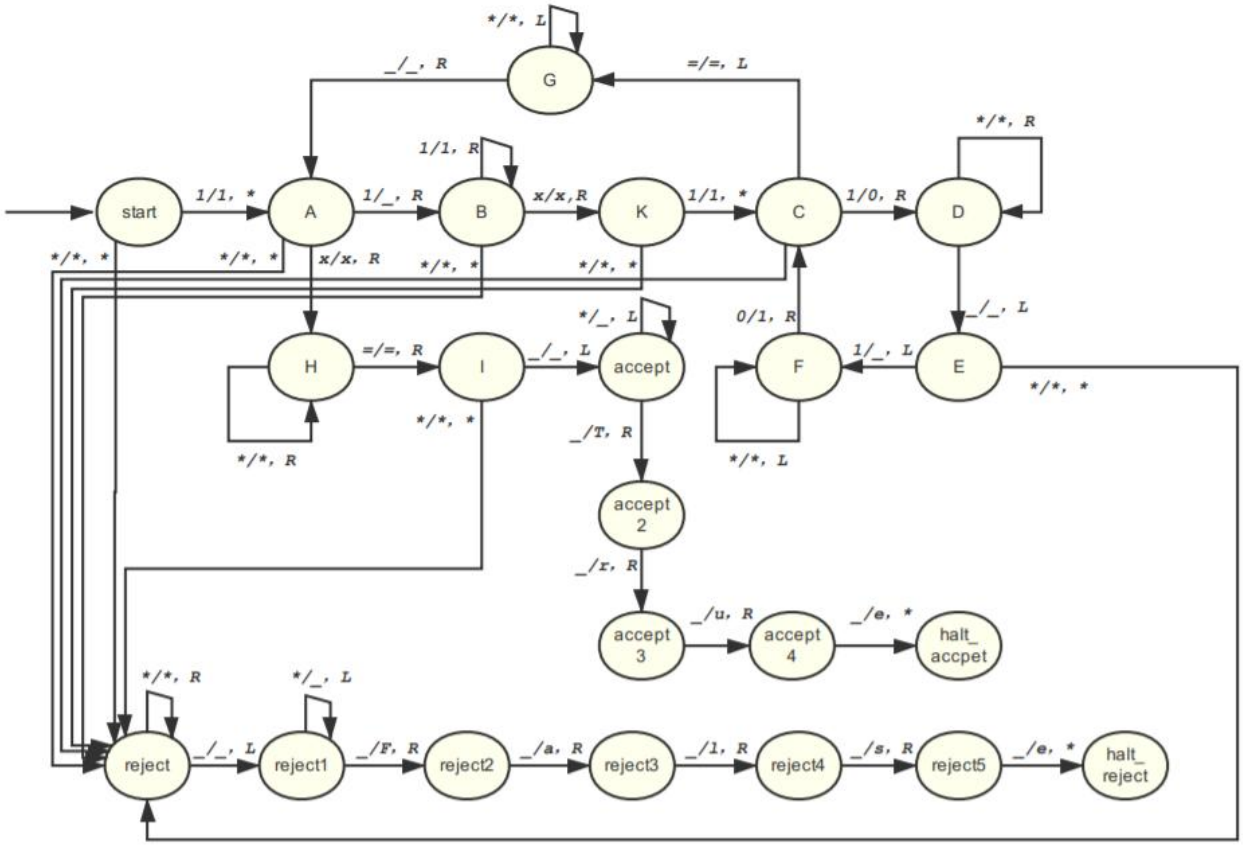
判定：只有落在 accept 状态的输入串才被接受，停在任何其他状态都不被接受。

为了满足“对于符合图灵机语言要求的字符串，在纸带上打印 True；对于不符合图灵机语言要求的字符串，在纸带上打印 False。要求在图灵机停止运行时，纸带上不出现 True 和 False 外的其他内容”的要求，可以通过增加一些状态来处理来实行打印的操作。

于是对终止状态进行处理，增加状态：

accept: 向左扫描消去纸带上所有非空的内容，之后写上‘T’
accept2: 在纸带上写下‘r’
accept3: 在纸带上写下‘u’
accept4: 在纸带上写下‘e’
halt_accept: 终止且接收
reject: 磁头移动到非空内容的最右边
reject1: 向左扫描消去纸带上所有非空的内容，之后写下‘F’
reject2: 在纸带上写下‘a’
reject3: 在纸带上写下‘l’
reject4: 在纸带上写下‘s’
reject5: 在纸带上写下‘e’
halt_reject: 终止但不接受

画出最终版本的图灵机状态转移图：



该图灵机的状态集为

#Q = {start,A,B,C,D,E,F,G,H,I,K,accept,accept2,accept3,accept4,
halt_accept,reject,reject1,reject2,reject3,reject4,reject5,halt_reject}

输入字符集为 #S = {1,x,=}

纸带符号集为 #T = {0,1,x,=,_}

开始状态为 #q0 = start

接收状态为 #F = {halt_accept}

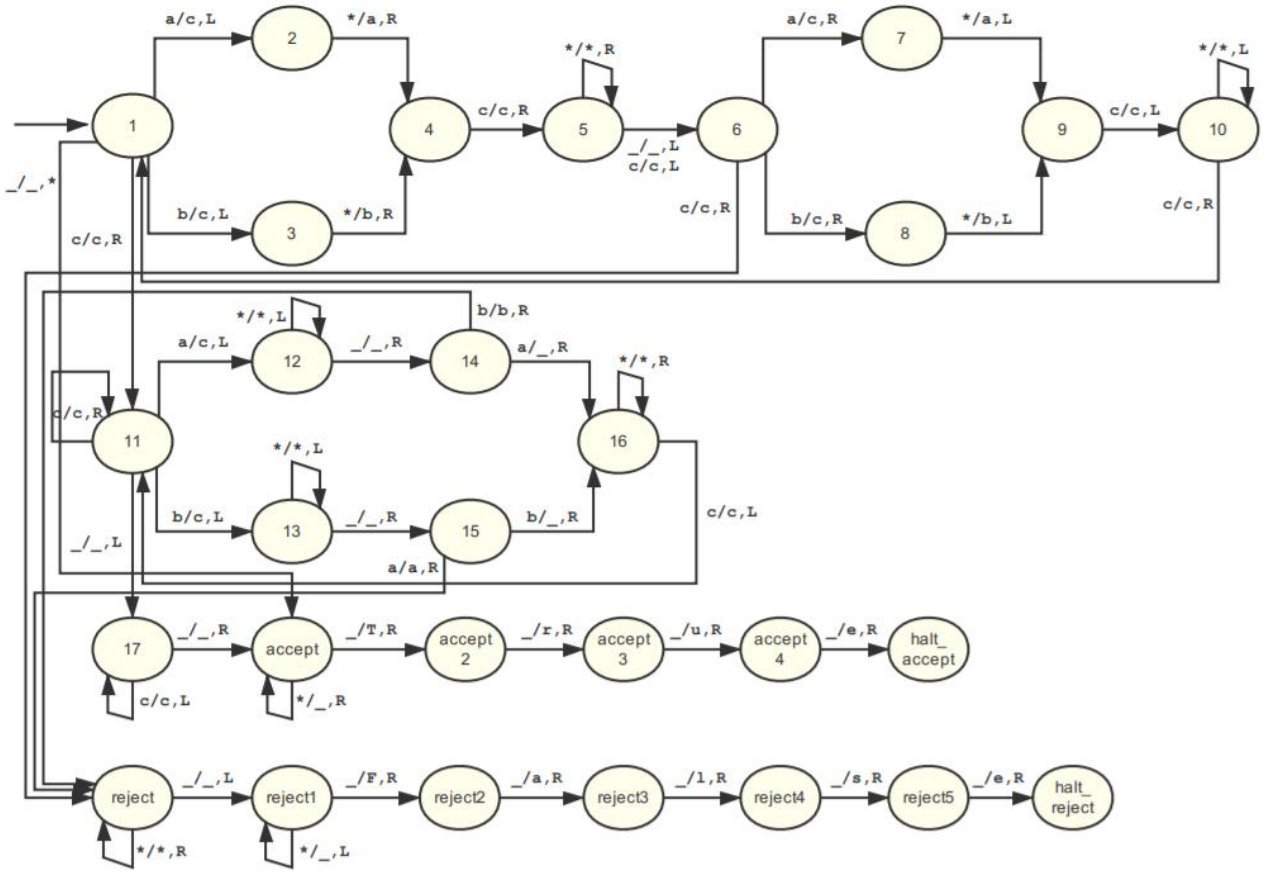
（2） $L_2 = \{ \ ww \mid w \in \{a,b\}^* \}$

要判断一个仅含 a 和 b 的串是否属于 L2，就是要判断是否能把该输入串分为相同的两部分。图灵机分为两步：第一步，试着将串分为长度相等的两部分，如果可以，继续下一步，判断这两部分是否相同。为了将串分隔，引入一个新的符号 c，用 c 将这两部分分隔开。

第一步：在纸带中非空内容的最左边和最右边写下‘c’，每将左边的‘c’向右移动一个位置（也即与右边的字符交换），就将右边的‘c’向左移动一个位置（也即与左边的字符交换），直到两个子串由两个连续的‘c’隔开。如果左边移动之后右边的‘c’无法再移动，说明这个串的长度是奇数，显然不被接受。

第二步：检验两个长度相等的子串相同，从右边串的起始处开始，每扫描一个字符，记住其内容，并将其改为‘c’用作标记下次扫描的起点。回到整个串的最左边，左边串的起点，如果第一个字符与记住的字符相同，那么消去该字符，移动到右边串的起始处，重复上述过程，否则的话，说明串不同，不被接受。

画出图灵机的状态转移图：



给出每个状态含义的描述：

1：开始状态

2：待交换的字符是‘a’

3：待交换的字符是‘b’

4：将左边‘c’与刚才遇到的字符交换

5：往右移动寻找右边的‘c’或者空白符

6：找到了右边的‘c’

7：待交换的字符是‘a’

8：待交换的字符是‘b’

9：将右边的‘c’与刚才遇到的字符交换

10：往左移动寻找左边的‘c’

11：输入字符已被分割为两个长度相等的子串，以“cc”隔开

12：右边的串开始处为‘a’，并往左移动

13：右边的串开始处为‘b’，并往左移动

14：移动到最左边

15：移动到最左边

16：左边的串开始处与右边串的开始处相同，往右继续查找右边串的起始处

17：两个子串相等

accept：向右扫描消去纸带上所有非空的字符，之后写上‘T’

accept2：在纸带上写下‘r’

accept3：在纸带上写下‘u’

accept4: 在纸带上写下‘e’
halt_accept: 终止且接收
reject: 磁头移动到非空内容的最右边
reject1: 向左扫描消去纸带上所有非空的字符，之后写下‘F’
reject2: 在纸带上写下‘a’
reject3: 在纸带上写下‘l’
reject4: 在纸带上写下‘s’
reject5: 在纸带上写下‘e’
halt_reject: 终止但不接受

该图灵机的状态集为

#Q = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,accept,accept2,accept3,
accept4,halt_accept,reject,reject1,reject2,reject3,reject4,reject5,halt_reject}

输入字符集为 #S = {a,b}
纸带符号集为 #T = {a,b,c,_}
开始状态为 #q0 = 1
接收状态为 #F = {halt_accept}

三、遇到的问题

在实验中遇到的问题有：

（1）实验中由于转移函数设计不慎，导致了图灵机没法停下来，程序陷入了死循环，经过仔细检查，最终定位到错误的地方。

（2）编程时遇到了 I/O 上的问题，也是因为对 java 语言不够熟悉，同时出现两个 OutputStreamWriter 的实例，导致第二个实例的写失效。于是做出调整，等 console.txt 写操作全部结束后再对 result.txt 进行写操作，就解决了这个问题。

四、实验结果演示

（1）对于 tm1，当输入串为“1x1=1”，运行过程如下：

```
Input: 1x1=1
================== RUN ==================
Step  : 0
Index : 0 1 2 3 4
Tape  : 1 x 1 = 1
Head  : ^
State : start
-----------------------------------------
Step  : 1
Index : 0 1 2 3 4
Tape  : 1 x 1 = 1
Head  : ^
State : A
-----------------------------------------
Step  : 2
Index : 1 2 3 4
Tape  : x 1 = 1
Head  : ^
State : B
```

```
-------------------------------------------
Step  : 3
Index : 1 2 3 4
Tape  : x 1 = 1
Head  :   ^
State : K
-------------------------------------------
Step  : 4
Index : 1 2 3 4
Tape  : x 1 = 1
Head  :     ^
State : C
-------------------------------------------
Step  : 5
Index : 1 2 3 4
Tape  : x 0 = 1
Head  :       ^
State : D
-------------------------------------------
Step  : 6
Index : 1 2 3 4
Tape  : x 0 = 1
Head  :         ^
State : D
-------------------------------------------
Step  : 7
Index : 1 2 3 4 5
Tape  : x 0 = 1 _
Head  :           ^
State : D
-------------------------------------------
Step  : 8
Index : 1 2 3 4
Tape  : x 0 = 1
Head  :         ^
State : E
-------------------------------------------
Step  : 9
Index : 1 2 3
Tape  : x 0 =
Head  :       ^
State : F
-------------------------------------------
Step  : 10
Index : 1 2 3
Tape  : x 0 =
Head  :     ^
State : F
-------------------------------------------
Step  : 11
Index : 1 2 3
Tape  : x 1 =
Head  :       ^
State : C
-------------------------------------------
Step  : 12
Index : 1 2 3
Tape  : x 1 =
Head  :     ^
State : G
-------------------------------------------
Step  : 13
Index : 1 2 3
```

```
Tape  : x 1 =
Head  : ^
State : G
-------------------------------------------
Step  : 14
Index : 0 1 2 3
Tape  : _ x 1 =
Head  : ^
State : G
-------------------------------------------
Step  : 15
Index : 1 2 3
Tape  : x 1 =
Head  : ^
State : A
-------------------------------------------
Step  : 16
Index : 1 2 3
Tape  : x 1 =
Head  :   ^
State : H
-------------------------------------------
Step  : 17
Index : 1 2 3
Tape  : x 1 =
Head  :     ^
State : H
-------------------------------------------
Step  : 18
Index : 1 2 3 4
Tape  : x 1 = _
Head  :       ^
State : I
-------------------------------------------
Step  : 19
Index : 1 2 3
Tape  : x 1 =
Head  :     ^
State : accept
-------------------------------------------
Step  : 20
Index : 1 2
Tape  : x 1
Head  :   ^
State : accept
-------------------------------------------
Step  : 21
Index : 1
Tape  : x
Head  : ^
State : accept
-------------------------------------------
Step  : 22
Index : 0
Tape  : _
Head  : ^
State : accept
-------------------------------------------
Step  : 23
Index : 0 1
Tape  : T _
Head  :   ^
State : accept2
```

```
------------------------------------------
Step  : 24
Index : 0 1 2
Tape  : T r _
Head  :     ^
State : accept3
------------------------------------------
Step  : 25
Index : 0 1 2 3
Tape  : T r u _
Head  :       ^
State : accept4
------------------------------------------
Step  : 26
Index : 0 1 2 3
Tape  : T r u e
Head  :       ^
State : halt_accept
------------------------------------------
Result: True
=================== END ==================
```

运行结果为：True

（2）对于 tm2，当输入串为“a”时，运行过程如下：

```
Input: a
=================== RUN ==================
Step  : 0
Index : 0
Tape  : a
Head  : ^
State : 1
------------------------------------------
Step  : 1
Index : 1 0
Tape  : _ c
Head  :   ^
State : 2
------------------------------------------
Step  : 2
Index : 1 0
Tape  : a c
Head  :     ^
State : 4
------------------------------------------
Step  : 3
Index : 1 0 1
Tape  : a c _
Head  :     ^
State : 5
------------------------------------------
Step  : 4
Index : 1 0
Tape  : a c
Head  :     ^
State : 6
------------------------------------------
Step  : 5
Index : 1 0 1
Tape  : a c _
Head  :     ^
State : reject
```

```
-----------------------------------------
Step  : 6
Index : 1 0
Tape  : a c
Head  :   ^
State : reject1
-----------------------------------------
Step  : 7
Index : 1
Tape  : a
Head  :  ^
State : reject1
-----------------------------------------
Step  : 8
Index : 2
Tape  : _
Head  :  ^
State : reject1
-----------------------------------------
Step  : 9
Index : 2 1
Tape  : F _
Head  :   ^
State : reject2
-----------------------------------------
Step  : 10
Index : 2 1 0
Tape  : F a _
Head  :     ^
State : reject3
-----------------------------------------
Step  : 11
Index : 2 1 0 1
Tape  : F a l _
Head  :       ^
State : reject4
-----------------------------------------
Step  : 12
Index : 2 1 0 1 2
Tape  : F a l s _
Head  :         ^
State : reject5
-----------------------------------------
Step  : 13
Index : 2 1 0 1 2
Tape  : F a l s e
Head  :         ^
State : halt_reject
-----------------------------------------
Result: False
=================== END ==================
```

运行结果为：False

五、实验总结

通过对图灵机解释器和模拟器的实现，加深了对图灵机功能的认识，在两个图灵机判定程序的设计中学会了一些图灵机设计的技巧。图灵机的运行过程就是不断查表进行状态迁移和纸带读写的过程，图灵机模拟器实现并不困难，较为困难的是图灵机的设计。