FLA课程项目实验报告

191220163 张木子苗 计算机科学与技术系

实验完成度

在本次实验中，我完成了所有3个任务

分析与设计思路

对错误情况的限制和处理

由于本次实验输入参数和文件内容格式的错误情况比较多，我对我的解析器加有以下限制：

1. 对于命令行工具的参数，只有以下3种情况可以正确解析：

   ```
   turing -h|--help
   turing <tm> <input>
   turing -v|--verbose <tm> <input>
   ```

   仅在输入 `turing -h|--help` 时可以查看 help 的内容，多提供参数或把 -h 放在其它参数之后都会报错；

   后2种情况在缺少 <tm> 或 <input>，或输入文件 <tm> 无法打开时，均会报 "Illegal_parameters" 的错误，并打印 usage 作为提示；

   -v|--verbose 必须紧跟在 ./turing 之后，才可正确进入verbose模式。

2. 对于输入的图灵机文件，代码会自动忽略空行与注释。在进行解析时，以#开头的行中的空格均会被忽略不计，但是代码不会检查花括号的匹配情况（不匹配时会解析错误）。Q、S、G、q0、B、F、N 的输入顺序可以打乱，但是缺一不可，且均必须在迁移函数delta的描述之前（否则报错）。

3. 本次实验简易实现了通配符，有以下两种使用：

   ```
   q_0 * * l/r/* q_1 #无论当前字符为什么，都保证其不变
   q_0 * a l/r/* q_1 #无论当前字符为什么，都将其变为a（代指任意磁带符号）
   ```

   如果需要使用通配符，在写迁移函数时务必注意其顺序，因为先写出来的规则会被先匹配。

4. 命令行参数错误，返回错误码-1；输入的.tm文件语法错误，返回错误码-2；输入串中出现不在输入符号表中的符号，返回错误码-3。

对于未列在上述情况中的错误情况，程序不能保证报错，也不能保证运行正确。

解析器实现思路

实现该步的核心在于以下数据结构：

```
class Turing_Machine
{
public:
    //构造函数，根据输入文件和verbose_mode标志建立图灵机
    Turing_Machine(FILE* fp, bool mode);
    //根据输入input，运行该图灵机
    bool Run(char* input);
    //展示当前图灵机内所有数据，用于观察该图灵机的建立是否正确
    void Show();
    //判断某个状态是否处于接受状态
    bool In_final_state(char* state);
    //查询迁移函数，根据cur_state和cur_tape，得到需要写的符号、磁头移动方向和下一状态
    void Look_up_transfer_function(char* cur_state, char* cur_tape, char* &write_tape, char* &transfer_directio
    //判断当前解析是否正确，即.tm文件的格式是否存在语法错误
    bool Parsed_successfully(){return parsed_successfully;}
private:
    std::vector<char*> Q; //State set
    std::vector<char> S;  //Input symbol set
    std::vector<char> G;  //Tape symbol set
    char* q0;             //Initial state
    char B;               //Space symbol
    std::vector<char*> F; //Final state set
    int N;                //Number of tapes
    std::vector<char**> delta;//Transfer function
    bool verbose_mode;
    bool parsed_successfully;
};
```

在通过解析参数，得到文件指针 fp 和标志是否处于 verbose 模式的标志位 mode 之后，我们根据文件内容和模式建立图灵机。从图中可见，我们将图灵机的状态机、迁移函数等都储存在了该 class 的 private 数据内。并且对外提供了5个函数，其作用已用注释给出。

解析器实现思路如下：

1. 一次读取文件的一行，存入缓冲区Buffer中
2. 越过每行最开始的空白符，如果当前行为注释或空行，则跳过。
3. 否则，判断该行第一个符号是否为'#'。如果是，则处理该行，去掉该行的空格和注释，放入 Processed_Buffer 中。然后根据其开头的字符，判断是图灵机哪一组成部分的描述，并分情况处理：

```
if(strncmp(Processed_Buffer,"Q={",3) == 0)
{…
}
else if(strncmp(Processed_Buffer,"S={",3) == 0)
{…
}
else if(strncmp(Processed_Buffer,"G={",3) == 0)
{…
}
else if(strncmp(Processed_Buffer,"q0=",3) == 0)
{…
}
else if(strncmp(Processed_Buffer,"B=",2) == 0)…
else if(strncmp(Processed_Buffer,"F={",3) == 0)
{…
}
else if(strncmp(Processed_Buffer,"N=",2) == 0)
{…
}
else continue;
count++;
```

4. 如果不是'#'，则为迁移函数的描述。由于要求了迁移函数必须放在最后，先判断是否前7个组成部分都已齐全，再解析当前行，存入迁移函数std::vector<char**> delta中即可；

   delta 中储存的为一个二维数组的地址，其内存通过动态分配。内容即是迁移函数描述的5个字符串，下标0-4的字符串与 当前状态、当前带头下符号、将要写入的新符号为、下一步移动方向和下一状态为对应。

```
if(count != 7)
{
    parsed_successfully = false;
    return;
}
```

当文件解析完成之后，我们的图灵机也就创建完毕，利用Show可以打印观察建立是否正确。

```
//Show();
parsed_successfully = true;
```

分析器实现思路

调用类Turing_Machine提供的方法Run，模拟运行：

```
//根据输入input，运行该图灵机
bool Run(char* input);
```

在这一步，核心在于该数据结构Tape，各个部分的作用已在注释中写明，具体实现请参照代码：

```
class Tape
{
public:
    Tape();
    //设置当前磁带的输入
    void set_input(char* input);
    //返回当前磁头所指向的符号
    char cur_symbol() {return tape[cur_index];}
    //根据需要写下的符号和移动方向，进行磁头移动
    void move(char write_symbol, char direction);
    //展示当前磁带上的内容
    void show_tape();
    //verbose模式下展示当前磁带上的内容
    void show_tape_v(int n);
private:
    int size;       //当前磁带大小
    int left_index; //最左的非空白符符号的下标，如果磁头越过了最左的非空白符，则是磁头下标
    int right_index;//最右的非空白符符号的下标，如果磁头越过了最右的非空白符，则是磁头下标
    int mid_index;  //0位置的下标
    int cur_index;  //当前磁头所指位置
    char* tape;     //磁带
    void extend_left(); //往左扩充磁带
    void extend_right();//往右扩充磁带
};
```

分析器实现思路如下：

先初始化，设置状态为初始状态，创建N条磁带，并且设置磁带0上内容为输入input

```
int step = 0;
char* cur_state = q0;
Tape tapes[N];
tapes[0].set_input(input);
```

之后只需要查询迁移函数，根据结果做移动即可。

当在当前状态和磁带符号下无移动，或者进入了终止状态时，模拟运行结束，打印结果。

```
char* write_tape = NULL;
char* transfer_direction = NULL;
char* next_state = NULL;
while(!In_final_state(cur_state))
{
    //int debug = 0;
    char* cur_tape = new char[N+1];
    for(int i = 0;i < N; i++)
        cur_tape[i] = tapes[i].cur_symbol();
    cur_tape[N] = '\0';
    //std::cout << cur_tape << std::endl;
    next_state = NULL;
    Look_up_transfer_function(cur_state,cur_tape,write_tape,transfer_direction,next_state);
    if(next_state == NULL)
    {…
    }
    else{…
    }
    //std::cin >> debug;
}
```

求最大公约数图灵机实现思路

思路：更相减损术

start状态下：先越过第一个'1'串，当遇到'0'时，代表已经完全越过。再往右移动一步，进入copy状态

```
start 1* 1* r* start ;Skip the first string of consecutive 1s
start 0* _* r* copy  ;Encountered 0, start copying the second number
```

copy状态下：

1. 将第1条磁带上的'1'复制到第2条磁带上，并且用'_'将第1条磁带的'1'覆盖；
2. 结束复制后，移动第1条磁带的磁头，越过刚刚放下的所有空白符，使磁头指向第1条磁带右边的第一个'1'，然后进入cmp状态。

```
copy 1_ _1 rr copy  ;Copy the number
copy __ __ ll copy  ;End copy
copy _1 _1 l* copy  ;Move the head of the first tape to the first 1 on the right
copy 11 11 ** cmp   ;Start compare
```

cmp状态下：两个磁头同时向左移动，如果：

1. 同时碰到空白符，两个数相等，此时即为求得的最大公约数。

```
cmp 11 11 ll cmp    ;Compare, move two heads at the same time
cmp __ __ rr accept ;Equal, accept
```

2. 第2条磁带先碰到空白符，代表第1条磁带上数更大， 需要用第1条磁带上的数减去第2条磁带上的，进入one_sub_two状态。

```
;The number on the first tape is greater than the number on the second tape,
;subtract the second one from the first one
cmp 1_ 1_ rr one_sub_two
```

3. 第1条磁带先碰到空白符，代表第2条磁带上数更大， 需要用第2条磁带上的数减去第1条磁带上的，进入two_sub_one状态。

```
;The number on the second tape is greater than the number on the first tape,
;subtract the first one from the second one
cmp _1 _1 rr two_sub_one
```

one_sub_two状态下：做减法。我们在cmp移动时，两条磁带磁头移动的步数是相同的，所以此时依旧可以同时向右移动。用'_'覆盖第一条磁带上的'1'，而第2条磁带上的'1'不变，移到最右端时，就完成了减法。

然后再将第1条磁带的磁头左移，越过我们刚刚放下的'_'直到其到达右边的第一个'1'处，返回cmp状态下继续比较

```
one_sub_two 11 _1 rr one_sub_two ;Do the subtraction and cover the 1 on the tape with '_'
one_sub_two __ __ l* one_sub_two ;Skip '_', move the head of the first tape to the first 1 on the right
one_sub_two 1_ 1_ *l cmp         ;Compare again
```

two_sub_one状态下类似：

```
two_sub_one 11 1_ rr two_sub_one ;Do the subtraction and cover the 1 on the tape with '_'
two_sub_one __ __ *l two_sub_one ;Skip '_', move the head of the second tape to the first 1 on the right
two_sub_one _1 _1 l* cmp         ;Compare again
```

注意，该图灵机在输入不合法时，无法正确处理！

实验中遇到的问题及解决方案

在本次实验过程中我遇到的bug比较少，主要方法是：每做完一步后都对代码进行一定的测试。

在构造完图灵机之后，我编写了类 Turing_Machine 的方法 Show，将当前构造图灵机的状态集合、迁移函数等全部打印出来，检查构造是否正确。

在编写好了单条磁带的输入设置 set_input 函数之后，用不同规模的输入进行测试，检查在纸带过短，字符溢出时，extend_right 函数是否能正确扩展纸带并且完成对各个参数的设置。

在每次遇到bug时，利用插桩法，在各个地方打印变量。一可根据最后打印的语句定位bug所在位置，二可检查各个变量的取值是否正常。

总结感想

1. 在面向对象编程时，注意把握好类的划分，做好封装和数据保护。
2. 在编写大项目时，务必注意边写边测，方便定位bug到刚写的代码中。
3. 在理不清楚代码逻辑时，用纸笔推算是很好的方法。

对课程和实验的意见与建议

尽量把课程项目和第3次作业提前布置，这样最后一个月备考压力会小一点。

最后感谢老师和助教学长在实验过程中的指导！