201220180 李全昊 FLA_LAB 实验报告

一. 实验完成度

完成了三个阶段的所有任务

二. 分析与设计思路

总体步骤是按照 stage1-3 来写代码的;

1. stage1 要求我实现一个多带图灵机程序解析器；在这个阶段主要对应的是我的 TuringMachine 类的初始化函数;

我的数据结构构造如下图所示:

```
class TuringMachine
{
    private:
        //define
        vector<string> states;//Q

        vector<char> input_char;//S
        vector<char> tape_char;//G
        string start_state; //q0
        char blank; //B
        vector<string> final_states;//F
        int nTape; //N

        vector<list<Cell>> tapes;
        vector<list<Cell>::iterator> heads;
        map<input_, output_> transition;
        //argc
        bool verbose;
        string filename;
    public:
        TuringMachine (string file_name, bool verbose);
        string run(string input);
        void print_error(vector<string> message, int err_num);
        string clear_annotation(string s);
        void check_input(string input);
        void print_step(string cur_state, int step);
        string Get_result();
        void Write_Move(string old_symbols, string symbols, string directions);
};
```

在初始化函数中进行的主要是读入 tm 文件进行的字符串处理的操作；读入字符串并且初始化我的图灵机类的成员 Q, S, G, q0 等。在处理字符串的过程中如果碰到了相对应的错误会调用 prit_error 函数进行错误的输出（比如文件打不开等错误）;

这里的 verbose 成员用来指示当前是否为 verbose 状态，如果是的话会进行更加详细的输出。

2. stage2 要求我们的图灵机程序经过解析器解析后，将得到一个对应的多带图灵机模拟器。模拟器读入一个字符串(即命令行参数中的<input>) 作为图灵机的输入。在这个阶段主要对应的是我的 run(string input)函数。在这个函数里我们接收命令行传来的 input 参数，然后进行图灵机的运行；整体运行的过程其实就是由 tm 文件指定的 transition fuction 来决定的。

```
vector<list<Cell>> tapes;
vector<list<Cell>::iterator> heads;
map<input , output > transition;
```

在这里我把每个 tape 作为一个 list，因为他是一个双向的链表，符合我们对纸带的理解。每一个纸带上的单元是一个结构体 Cell;里面包含有对应的 index 和 char 类型的元素。

Tapes 是包含了多个 tape 的 vector;

同理每个 head 是每个 tape 对应的磁头;

而 transition 存有我们 tm 文件指定的所有转移函数，是一个 map 结构；他的 key value 分别是我自己定义的结构体 input_和 output_。

```
struct Cell
{
    int index;
    char symbol;
    Cell(int i, char s)
    {
        index = i;
        symbol = s;
    }
};
```

```
struct input_
{
    string state; //输入的状态
    string symbols; //在当前n个磁带上的symbol
    bool operator < (const input_ x) const {
            return (state < x.state || (state == x.state && symbols < x.symbols));
        }
    bool operator==(const input_ b) const
    {
        return this->state == b.state && this->symbols == b.symbols;
    }
};

struct output_
{
    string state; //输出的状态
    string symbols; //输出的n个symbol
    string directions; //输出的n个head的移动方向
    bool operator < (const output_ x) const {
            return (state < x.state || (state == x.state && symbols < x.symbols) || (state == x.state && symbols == x.symbols && directions <
        }
    bool operator==(const output_ b) const
    {
        return this->state == b.state && this->symbols == b.symbols && this->directions == b.directions;
    }
};
```

要特别注意的是这里作为 key 的 input_结构体必须要好好写他的小于号重载; 因为这在 map 中是非常重要的，用来排序 key 和比较 key！我当时 debug 了很久。

3. stage3 要求我们自己构思两个图灵机；并且编写对应的 tm 文件；这本身的思路我就不在此介绍了；值得一提的是，在测试的时候发现我没有对*号的匹配进行处理，因此在思考之后，我决定对于每一个规则，我都去试试看当前的 head 底下的 symbol 是否可以通过对于*的转换满足，如果不行那就彻底不行了（在此之前已经对普通的规则寻找过了）。

还需要指出的是*不能匹配 blank!所以我们应该在循环的过程中，如果发现不能匹配，而且有一个 head 底下是 blank，那就彻底不能匹配了。

三. 总结感想

这次试验难度不是很大，但是比较繁琐，在配环境的时候也遇到了一些问题，在询问了助教 gg 之后得到了解决！Windows 下 build 得到的是 sln 文件要用 VS 去编译；Linux 下直接得到 makefile ; make 就行了。