形式语言与自动机 课程项目 实验报告

2020 年 秋季

舒意恒 MF20330067 计算机科学与技术系

yhshu@smail.nju.edu.cn

1. 分析与设计

1.1 项目分析

本项目主要包含三个部分：

1. 多磁带图灵机的解析器：给定特定语法的图灵机程序代码，解析出图灵机的表示；其中，在调试模式下，如果解析出错需要打印错误信息
2. 多磁带图灵机的模拟器：给定对图灵机的输入字符串，模拟图灵机对该字符串的处理过程，并最终输出图灵机第一条磁带上的字符串；其中，在调试模式下，打印图灵机的每一步操作
3. 对于给定问题，编写图灵机代码，并在所编写的解析器和模拟器上运行

项目需要使用 C++ 进行编写，C++ 支持面向对象编程范式，可利用这一范式对图灵机进行描述。

1.2 解析器设计

解析器通过 `TuringMachine` 类的构造函数 `TuringMachine::TuringMachine(const vector<string> &tm_str` 实现，该函数的参数是根据换行符分割的图灵机程序代码。

解析器读入每一行，根据其声明：

- 设置图灵机的状态集 `void TuringMachine::BuildStateSet(const string &line)`
- 设置图灵机的开始状态，直接在构造函数中处理
- 设置图灵机的终止状态集 `void TuringMachine::BuildFinalStateSet(const string &line)`
- 设置图灵机的输入符号集 `void TuringMachine::BuildInputSymbolsSet(const string &line)`
- 设置图灵机的磁带符号集 `void TuringMachine::BuildTapeSymbolsSet(const string &line)`
- 设置图灵机的空白符号，直接在构造函数中处理
- 设置图灵机的迁移函数，直接在构造函数中处理
- 设置图灵机的磁带数，直接在构造函数中处理，并检查磁带数是否为正整数

解析器支持 `verbose` 模式，在解析器中，`void TuringMachine::CheckDefinition()` 方法支持检查以下类型的错误：

- 判断图灵机代码所声明的初始状态、终止状态集合是否都定义在同一程序的状态集中；
- 判断图灵机代码所声明的空白符号是否属于其磁带符号集；
- 判断图灵机代码所声明的磁带数是否是一个非零自然数；
- 判断图灵机代码所声明的转移函数是否满足确定性图灵机的要求

`bool TuringMachine::CheckTransitionFuncDefinition(const vector<string> &transition_func_vec)` 方法用于检查迁移函数的定义，支持检查以下类型的错误：

- 判断迁移函数所声明的输入状态是否属于图灵机的状态集；
- 判断迁移函数所声明的输入符号是否属于图灵机的磁带符号集；
- 判断迁移函数所声明的输出符号是否属于图灵机的磁带符号集；
- 判断迁移函数所声明的输出状态是否属于图灵机的状态集；
- 判断迁移函数所声明的输入输出符号是否对应于图灵机的磁带数；
- 判断迁移函数所声明的读写头操作是否合法

`void TuringMachine::CheckInputSymbols(const string &input)` 方法用于检查给定图灵机输入字符串，这些字符是否都属于图灵机定义中的输入符号集符号集。

1.3 模拟器设计

模拟器通过 `TuringMachine` 类的 `Run` 函数实现。模拟器支持 `verbose` 模式，语法检查将判断输入串中的所有字符是否均属于输入符号集。

模拟的主要过程在 `void TuringMachine::Run(const string &input)` 方法中：

```
void TuringMachine::Run(const string &input) {
  CheckInputSymbols(input); // 检查输入字符串是否合法
  InitTapes(input);         // 根据输入字符串初始化磁带
```

```
  bool accepted = true;      // 记录对于该输入字符串，图灵机是否接受
  while (final_state_set_.find(cur_state_) == final_state_set_.end()) {
    // 当前状态不是可接受状态时，循环
    if (GetVerbose()) {      // 调试模式
      cout << GetVerboseStr(); // 输出图灵机的这步操作
    } else {
      for (Tape &tape : tapes) {
        tape.CleanBothEnds(); // 清除读写头指向之外的多余的空白符号记录
      }
    }
    if (not Step()) {          // 图灵机执行一步操作，如果该步没有可用的迁移函数，返回
false
      accepted = false;
      break;
    }
  }
  if (accepted and GetVerbose()) // 如果图灵机接受，输出最后一步
    cout << GetVerboseStr();

  PrintFirstTapeResult();      // 输出第一条磁带的结果，清理其他磁带
}
```

1.4 多磁带图灵机程序 1

设计思路：正常的输入串包含两组 a 和两组 b. 将输入字符串的第二组 a 之前的子串，从第一条磁带剪切到第二条磁带，然后比对第一条磁带的剩余部分是否与第二条磁带相同，若相同则图灵机接受输入字符串。

图灵机程序代码详见 `programs.case1.tm`.

1.5 多磁带图灵机程序 2

设计思路：输入字符串中包含 3 组 1，将第一组 1 剪切到第二条磁带，删除乘号；对于第二组 1 中的每个 1，在第三条磁带上新增一次第二条磁带上的内容；之后，删除等号，比对第一条磁带上的剩余子串和第三条磁带，若相同则图灵机接受输入字符串。

图灵机程序代码详见 `programs.case2.tm`.

2. 实验完成度

按文档要求，实现了一个图灵机解析器和模拟器，支持对特定语法的多磁带图灵机进行模拟和解析。解析器与模拟器均包含调试功能。

- `turing-project` 目录包含 C++ 源码
  - `main.cpp` 主程序
  - `message.h` `message.cpp` 包含各种打印控制台信息的函数
  - `Tape.h` `Tape.cpp` 描述多磁带图灵机中的一条磁带
  - `TransitionFunction.h` `TransitionFunction.cpp` 描述多磁带图灵机中的迁移函数
  - `TuringMachine.h` `TuringMachine.cpp` 描述多磁带图灵机整体

    - `utils.h` `utils.cpp` 包含字符串处理等工具函数
- `programs` 目录包含图灵机程序代码
    - `case1.tm` 问题 1 图灵机代码
    - `case2.tm` 问题 2 图灵机代码

2.1 程序 1 测试

用例编号	输入字符串	期望结果	用例编号	输入字符串	期望结果
1	abbabb	true	8	abbab	false
2	aabaab	true	9	a	false
3	ab	false	10	b	false
4	abbabbb	false	11	baabaa	false
5	aabbabb	false	12	aabb	false
6	babb	false	13	aacbb	illegal input
7	abba	false			

2.2 程序 2 测试

用例编号	输入字符串	期望结果	用例编号	输入字符串	期望结果
1	11x11=1111	true	12	11x11=111	false
2	11x111=111111	true	13	11x11=11111	false
3	111x11=111111	true	14	1=1	false
4	1x=1	false	15	1=11	false
5	x1=1	false	16	11=1	false
6	1x1=1	true	17	x	false
7	x=	false	18	=	false
8	x=1	false	19	1=1x1	false
9	1x=	false	20	11=111x111111	false
10	x1=	false	21	11	false
11	11x111=1111	false	22	211x11=111	illegal input

3. 问题与解决方案

本节描述完成任务过程中遭遇的关键问题与解决方案。

3.1 图灵机定义的检查

图灵机可以由七元组描述，程序代码作为一个字符串，对七个元素的声明是需要按顺序的。但这七个元素之间，并不是简单的顺序关系，是彼此关联的，意味着解析器在解析的过程中就立刻完成对图灵机定义的检查可能是比较困难的。

例如，图灵机的定义应当满足终止状态集中的所有终止状态都应该属于状态集，但实际上终止状态集声明与状态集声明的先后顺序，是并不明确的。类似的关系还存在于初始状态与状态集、空白符号与磁带符合集之间，以及迁移函数的声明之中。

检查图灵机定义的方案

一种简单的解决方案是，对于不能在解析过程中完成的检查，在读取整个图灵机代码之后进行检查。相关设计已体现在 `1.2 解析器设计` 一节的叙述中。

3.2 磁带两端空白符号的处理

磁带两端的空白符号并不总是多余的，关于空白符号有以下要求：

- 图灵机的瞬时描述 注 4：一般情况下，两端非空格符号外侧的空格符号和对应索引都不需要打印，只需打印纸带上最左非空格符号到最右非空格符号之间的符号及对应索引；
- 图灵机的瞬时描述 注 5：当读写头指向的位置为非空格符号外侧的空格符号时，需要打印必要的空格符号及对应索引，空格符号以 `'_'` 表示；
- 模拟运行结束后，将第一条纸带上的内容 (首尾分别为纸带上最左和最右非空格符号) 作为最后的输出

处理磁带两端空白符号的方案

磁带外空白符号发挥作用的时刻仅在读写头指向它时。因此，在模拟图灵机的每步操作中，清理在读写头更外侧的空白符号即可。并在模拟结束后输出磁带内容时，注意考虑首尾分别为纸带上最左和最右非空格符号。

4. 感想

现代计算机的计算模型，来源于图灵机。而使用现代计算机回过头来模拟图灵机，对于了解计算机在计算方面最基本的原理，十分有意义。

本实验的难点是任务三关于图灵机的设计，对于学习本课程来说，设计图灵机比编程实现解析器和模拟器更加重要。图灵机可能被输入任何不符合我们期望的字符串，而对于这些情况，图灵机要一一识别并拒绝接受这些字符串。

而使用 C++ 实现解析器和模拟器，其核心思路是比较简单的。图灵机作为一种基本的计算模型，在编程上并不复杂。但类似地，图灵机的解析和模拟过程中，也可能存在大量我们不期望的异常情况，包括但不限于非法输出、磁带旁空白符号的处理等，实现一个健壮的解析器和模拟器的难度较高。

5. 实验与课程建议

感谢卜磊老师的授课与耐心答疑和助教学长的作业批改与习题课。

C++ 相比于 Java 或 Python 等编程语言，环境配置与程序调试稍复杂。而查重平台应当不局限于 C++ 语言，或许课程实验可重新考虑对编程语言的限制。

习题课内容极多，不仅包含作业本身，还包含相关例题，对于复习很有作用，只是在课堂上的 3 课时难以完全呈现。