使用文档

我们实现了两个可以用于移入规约分析的简单的接口，并进行了相应的测试。

语法输入约定

输入文件格式

语法：

```
# N 表示符号数量（0-based） M 表示产生式数量（0-based）
# 要加入 start -> E 这个产生式，且应该放在第一条（E只是举例子），start 记作 -1

N M
prod 1
prod 2
...
prod M

# 样例
# 产生式的第一个数表示产生式的右边的符号数量，第二个数表示产生式左边的符号，之后的数表示右边的符号序列

4 3
1 -1 1
3 1 2 3 0
1 1 2
```

终结符序列输入：

```
# N 表示终结符数量

N
Array of terminal characters

# 样例

5
1 1 2 4 1
```

输入文件命名

语法文件：`grammar_{name}.txt`

示例：`grammar_add.txt`

终结符序列文件：`{name}_instance{index}_input.txt`

示例：`add_instance1_input.txt`

接口

直接与移入规约相关的有以下两个接口：

```
/* 接受一个语法，生成它的状态信息、状态转移表
 * 在接受的语法有歧义时，这个函数会输出错误信息到 stdout
 * state_info 用于存放生成的各个状态的信息
 * trans 用于存放生成的状态转移表
 * state_num 用于存放生成的状态的数量
 */
void GenerateStates(
    int number_of_symb, int number_of_prod,
    struct prod grammer[MAX_NUMBER_OF_PROD],
    struct state state_info[MAX_NUMBER_OF_STATE],
    struct trans_result trans[MAX_NUMBER_OF_STATE][MAX_NUMBER_OF_SYMB],
    int* state_num);

/* 接受语法、状态转移表、终结符序列，输出移入规约结果到 output 里
 * output 用于存储移入规约的结果
 */
void shift_reduce(
    int number_of_symb, struct prod grammer[MAX_NUMBER_OF_PROD],
    struct trans_result trans[MAX_NUMBER_OF_STATE][MAX_NUMBER_OF_SYMB],
    int input[MAX_NUMBER_OF_INPUT], int length_of_input, OutputSequence* output);
```

我们还提供了一些相关的辅助函数：

```
// 初始化移入规约结果序列
void init_output_sequence(OutputSequence* seq);

// 从文件中读入语法
void read_in_grammar(char *filename, int *number_of_symb, int
*number_of_prod,struct prod grammar[MAX_NUMBER_OF_PROD]);

// 获取移入规约结果中的某一步的结果
int get_from_output_sequence(OutputSequence* seq, int index)
```

使用方法

使用我们的移入规约接口需要安装 gcc。

请将您要编写的文件与本文件中的 `cfg.h util.c parser_gen.c` 放在同一目录下。在您要使用这些接口的文件中需要包含语句： `#include "cfg.h"`。

之后您就可以使用我们提供的接口了。我们在 `generator.c` 中示范了怎样使用的我们的接口，可供参考。

在编译文件时，可以采用以下方式：`gcc sourcefile.c parser_gen.c util.c -o targetfile`

测试方法

我们提供了三个语法以及每种语法十四组终结符序列的数据，并提供了测试的工具。

需要注意的是，在每一组数据中，都有两到三个是不合法的终结符序列。

如果您想要测试我们提供的数据，可以按照如下步骤进行：

- 使用命令 `gcc generator.c parser_gen.c util.c -o generator` 来编译我们提供的示范程序。

- 使用命令 `python3 ./driver.py` 来在所有数据上运行 `generator` 并输出移入规约结果以用于验证正确性。
- 使用命令 `python3 ./testall.py` 来验证所有移入规约过程的正确性。

自定义测试输入

如果您希望使用我们的测试工具并自定义需要测试的输入，那么首先，您需要在相应的位置放置您希望测试的输入并按照规范来命名。

例如，如果您希望新建一个语法：`new_grammar`，那么您应该在 shift_reduce 目录下新建一个文件夹 `test_shift_reduce`，然后在这个文件夹中再新建一个文件夹 `test_data`，并将所有的语法输入以及终结符序列输入放在这个目录下。

需要注意的是，所有的语法输入以及终结符序列输入的文件的命名应当遵循之前提到的文件命名规则，并且所有终结符序列输入的序号应当是从 1 开始的连续整数。

在确定待测试数据已经正确命名并放到正确位置之后，您需要对我们的测试脚本稍作修改来适应额外的数据。具体来说，您需要修改 `driver.py` 和 `testall.py` 中的两处待测试输入范围。

```
grammar_name = ["add_minus", "add_multiply", "simple_language"]
instance_indices = [range(1, 15), range(1, 15), range(1, 15)]
```

（这是默认的定义）

在这两个文件中，这一定义都在文件的开头处，您可以轻易找到并进行修改。

在修改完毕之后，只要像之前那样运行这两个脚本即可。