CS 5000: F21: Theory of Computability
Assignment 6

Vladimir Kulyukin
Department of Computer Science
Utah State University

October 09 2021

Learning Objectives

1. Cocke-Younger-Kasami Algorithm

Introduction

This programming assignment will focus on the CYK algorithm we covered in the last two lectures. You may want to review the PDFs and the reading materials in Canvas along with your class notes. No theoretical problems for this assignment.

Problem 1 (4 points)

Implement the CYK algorithm to determine if strings are in the language of a CNF grammar (CNFG). You'll write and save your solution in `cyk.py` that implements a Python class `CYK`. I've written some starter code for you in the `PyCYK` folder.

Your implementation of the `CYK` class must contain the boolean static method `is_in_cfl(x, cnfg)`, where `x` is a string and `cnfg` is a list of CNF rules that define a CNFG. When `x` is in the language of the CNFG specified by the second argument, the method returns `True` (i.e., the string `x` is in the language), else - `False` (i.e., the string `x` is not in the language).

Each rule can be implemented as a simple 2-tuple that represents the left hand and right hand sides of a rule. For example, `S -> AB` can be represented as `('S', 'AB')`. Here's an example of a simple CNFG implemented as a list of 2-tuples.

```
CFG = [('S', 'AD'),
       ('D', 'BC'),
       ('C', 'c'),
       ('B', 'b'),
       ('A', 'a')]
```

The class `CNFG` in `cnfg.py` implements a CNFG and allows you to add productions to the grammar and fetch the left hand sides given a right hand side, which is an important operation in the CYK algorithm. For example, suppose that we want to define the following CNFG.

1. $S \rightarrow AB$;
2. $S \rightarrow BC$;
3. $A \rightarrow BA$;
4. $A \rightarrow a$;
5. $B \rightarrow CC$;
6. $B \rightarrow b$;
7. $C \rightarrow AB$;
8. $C \rightarrow a$.

We can do it programmatically as follows with the `CNFG` class defined in `cnfg.py`.

```
grammar = CNFG()
## Add rule S -> AB
grammar.add_production("S", "A", "B")
## Add rule S -> BC
grammar.add_production("S", "B", "C")
## Add rule A -> BA
grammar.add_production("A", "B", "A")
## Add rule A -> a
grammar.add_production("A", "a")
## Add rule B -> CC
grammar.add_production("B", "C", "C")
## Add rule B -> b
grammar.add_production("B", "b")
## Add rule C -> AB
grammar.add_production("C", "A", "B")
## Add rule C -> a
grammar.add_production("C", "a")
## Fetch all X's from the grammar such that X -> AB
print("LHS for RHS AB is {}".format(grammar.fetch_lhs("A", "B")))
```

The call `grammar.fetch_lhs("A", "B")` is supposed to get the set of left-hand sides (LHSs) for which the right hand sides (RHSs) consists of two variables – A and B, in that order. The output of the last `print` statement is as follows.

```
LHS for RHS AB is {'C', 'S'}|
```

Why? Because the rules whose right hand side (RHS) is `AB` are $S \to AB$ and $C \to AB$. Hence, the `grammar.fetch_lhs("A", "B")` returns the set `{'C', 'S'}`.

I've written 19 unit tests in `cyktest.py` you can use to test your impelementation with different strings and CNFGs. Below is the output my implementation produces when I run the unit tests. You can comment them all out and then uncomment one unit test at a time and work on it.

```
>>> python cyktest.py
===== Test 1 =====
S -> AB
S -> BC
A -> BA
A -> a
B -> CC
B -> b
C -> AB
C -> a
Input string: baaba
Result = True
.===== Test 10 =====
Input string: ab
Result = False
.===== Test 1a =====
Input string: ab
Result = True
.===== Test 2 =====
Input string: baaa
Result = True
.===== Test 3 =====
Input string: baba
Result = False
.===== Test 4 =====
Input string: baaabab
Result = False
.===== Test 4a =====
Input string: baab
```

```
1: |B ||A C ||A C ||B |

1: |B ||A C ||A C ||B |
2: |A S ||B ||C S |

1: |B ||A C ||A C ||B |
2: |A S ||B ||C S |
3: |||B |

1: |B ||A C ||A C ||B |
2: |A S ||B ||C S |
3: |||B |
4: ||

Result = False
.===== Test 4b =====
Input string: aaab
1: |A C ||A C ||A C ||B |

1: |A C ||A C ||A C ||B |
2: |B ||B ||C S |

1: |A C ||A C ||A C ||B |
2: |B ||B ||C S |
3: |A C S ||B |

1: |A C ||A C ||A C ||B |
2: |B ||B ||C S |
3: |A C S ||B |
4: |C S |

Result = True
.===== Test 5 =====
Input string: aabb
Result = True
.===== Test 5a =====
Input string: bbb
Result = True
.===== Test 5b =====
Input string: bbb
Result = True
.===== Test 5c =====
Input string: cccccb
Result = False
.===== Test 5d =====
Input string: bababaaa
Result = True
.===== Test 6 =====
Input string: abc
Result = True
.===== Test 7 =====
Input string: abbbabb
Result = True
.===== Test 7b =====
Input string: abbc
Result = True
.===== Test 8 =====
Input string: bbc
Result = False
.===== Test 9 =====
Input string: aaabb
Result = False
.===== Test 00 =====
LHS for RHS AB is {'C', 'S'}
```

```
.
----------------------------------------------------------------------
Ran 19 tests in 0.008s

OK
```

What to Submit

1. `cyk.py` with your implementation of the `is_in_cfl(test_string, cnfg)` method.