Started on	Monday, 8 May 2023, 10:28 AM
State	Finished
Completed on	Monday, 8 May 2023, 10:35 AM
Time taken	6 mins 3 secs
Grade	**2.33** out of 10.00 (**23**%)

Question **1**

Correct

Mark 1.00 out of 1.00

Select all nullable non-terminals for the following CFG?

S → TU | aU

T → VW

U → aTb | b

V → aVc | WW

W → bW | Λ

Select one or more:

- ☐ a. U
- ☑ b. V ✓
- ☐ c. S
- ☑ d. T ✓
- ☑ e. W ✓

Question 2

Partially correct

Mark 1.33 out of 2.00

Fill in the missing values of Chomsky normal form equivalent to the following CFG.

(a, and b are terminals)

S → AbA

A → Aa | Λ

S → TA | BA | AB | [b]

A → A [B] C | a

T → A [B]

B → b

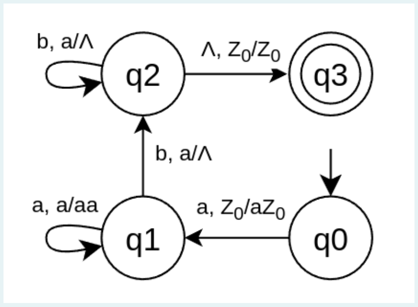
C → a

Question 3

Incorrect

Mark 0.00 out of 2.00

What are the strings accepted by the following PDA?



Select one or more:

- [✓] a. aaaaabbbbb ✓
- [✓] b. abba ✗
- [✓] c. ab ✓
- [✓] d. aabb ✓
- [✓] e. aaabb ✗

Question **4**

Incorrect

Mark 0.00 out of 2.00

What is the closest number of productions that the final CFG would contain after removing the Λ productions?

S → ACBDAC

A → BD

B → b | Λ

C → Bab,

D → d | Λ

Select one:

- a. 23
- b. 18
- c. 5
- [✓ d. 16] ✖

Question 5

Incorrect

Mark 0.00 out of 3.00

Complete the following function which **returns the new grammar after removing the lambda productions from a given CFG**. You are expected to write a Python code under the commented TODO, and do not alter any other lines.

For simplicity, you can assume that the **given grammar has no productions with more than one nullable non-terminal on the right-hand side**.
Consider the parameter grammar passed to the function as a dictionary with keys and values described as follows:

1. A key is a non-terminal symbol.
2. A value is a list of the right-hand side productions of the respective key.

For example, the production rules S -> TU | T | U | V and U -> cU | d | Λ can be represented as

G = {

'S': ['TU', 'T', 'U', 'V'],

'U': ['cU', 'd', ''],

}

Note that lambda production is represented with empty string('')

The parameter nullables contain nullable non-terminals as a list.

For example:

Test	Result

```
grammar = {
    'S': ['A', 'bC'],
    'A': ['aB'],
    'B': ['bB', ''],
    'C': ['cC', ''],
}
nullables = ['B', 'C']
print(remove_lambda_productions(grammar, nullables))
```
	`{'S': ['A', 'b', 'bC'], 'A': ['a', 'aB'], 'B': ['b', 'bB'], 'C': ['c', 'cC']}`

Answer: (penalty regime: 0 %)

Reset answer

Ace editor not ready. Perhaps reload page?
Falling back to raw text area.

```
def remove_lambda_productions(grammar, nullables):
    """
    Removes lambda productions from a given grammar.

    Args:
        grammar (dict): A dictionary representing the grammar to modify.
            The keys are non-terminal symbols, and the values are lists of
            production rules.
        nullables (list): A list of nullable non terminal symbols.

    Returns:
        dict: A new dictionary representing the modified grammar with lambda
            productions removed. The new dictionary has the same format as
            the input grammar.
    """
    new_grammar = {}
    for nt, productions in grammar.items():
        new_productions = set()
        for prod in productions:
            # Create new productions by removing nullable non-terminals
            # Note that the given grammar has no productions with more
            # than one nullable non-terminal in the right hand side.
            for i in range(len(prod)):
                # TODO: Complete the following code




            new_productions.add(prod) # add the initial production
        new_grammar[nt] = sorted(list(new_productions))

    # Remove lambda productions
    new_grammar = {
        nt: [prod for prod in prods if prod != ''] for nt, prods in new_grammar.items()
    }

    # Remove productions of the form A -> A
    new_grammar = {
        nt: [prod for prod in prods if prod != nt] for nt, prods in new_grammar.items()
    }
```

Syntax Error(s)

```
Sorry: IndentationError: expected an indented block (__tester__.python3, line 29)
```

Incorrect

Marks for this submission: 0.00/3.00.

Previous activity

◀ Lecture 7: Slides (PDF)

Jump to...