Assignment 2 v. 1.7

COMP9021, Session 2, 2015

Aims: The purpose of the assignment is to:

- design and implement an interface based on the desired behaviour of an application program;
- practice the use of builtin data structures: lists, tuples, sets, and dictionaries;
- develop problem solving skills.

Submission

Your program will be stored in a file named `picture_grammar.py`. Upload your file using WebCMS. Assignments can be submitted more than once: the last version is marked. Your assignment is due by September 27 11:59pm.

Assessment

The assignment consists of three parts, from easiest to more difficult. The easier the part, the more marks it gets, so everyone should easily score at least 50%.

For each test, the automarking script will let your program run for 30 seconds.

Up to one mark will reward good comments, good choice of names for identifiers and functions, readability of code, simplicity of statements, compactness of functions. This will be determined manually.

One extra mark will be awarded for making available to the class two interesting and insightful examples of grammars, which can then be used by everyone for testing purposes.

Late assignments will be penalised: the mark for a late submission will be the minimum of the awarded mark and 10 minus the number of full and partial days that have elapsed from the due date.

1 Some terminology

In this assignment, we work with a variation on the model of grammatical picture generation described in the paper *Two-dimensional Picture Grammar models*; have a look at the first three sections of this paper (of course you can read the whole paper, but only the first three sections are relevant to the assignment).

When we talk about "blank line", we mean a line with nothing but space characters, and possibly none (an empty line).

Call *token* any string which is a valid Python identifier, that is, a sequence of alphanumeric characters or underscores not starting with a digit. Call *symbol* any nonspace character. So ε is a symbol; it is a special symbol, meant to denote the empty character.

For instance:

- `a_token` is a token but not a symbol
- `A_token` is a token but not a symbol
- `_token_1` is a token but not a symbol
- `token10` is a token but not a symbol
- `x` is both a token and a symbol
- `X` is both a token and a symbol
- `*` is a symbol but not a token
- `@` is a symbol but not a token
- ε is a symbol but not a token

Tokens which are not symbols can only play the role of nonterminals of a grammar. Symbols which are not tokens can only play the role of terminals of a grammar. Tokens which are also symbols can play the role of either nonterminals or terminals of a grammar, and which role they actually play will be determined by how they occur in the rules of the grammar.

Call *rule* any sequence of characters consisting of a token (the left hand side of the rule) possibly preceded or followed by spaces, followed by `->`, followed by one or more tokens or symbols (the right hand side of the rule), with at least one space (possibly more) between successive tokens or symbols and with possibly spaces before the first or after the last token or symbol, and such that if ε occurs in the rule (necessarily on the right hand side), then it occurs only once and nothing else occurs on the right hand side. For instance,

- `a_token->a_token`
- `A_token -> *  a_token x`
- `    token10  ->  X X   X  X`
- `X    ->@   y @`
- `    x   ->    ` ε
- `x->  x`

are valid rules, but

- `* -> x`
- `x ->` ε ε
- `x ->` ε `a`

are not.

Call *table* any sequence of N rules, $N \geq 0$, written down on consecutive lines, with no blank lines in-between, such that different rules have different left hand sides, all right hand sides have the same number of tokens or symbols, and if one rule has ε as right hand side then all rules have ε as right hand side. Note that N can be equal to 0, in which case the table is "empty". For instance,

```
abc -> X Y Z
X -> X Y Z
Y -> a b A
```

is a valid table, but

```
X -> Y
abc -> U
X -> a
```

is not a valid table (because of X occurring twice on the left hand side),

```
X -> Y
abc -> U b
Y -> a
```

is not a valid table (because the right hand sides have size 1 for two rules but 2 for another rule), and

```
X -> ε
abc -> b
Y -> ε
```

is not a valid table either (because ε is the right hand side of some rules, but not all of them).

Call *axiom array* any sequence of N consecutive lines, $N \geq 1$, with no blank lines in-between, each line consisting of M tokens or symbols, $M \geq 1$, with the exception of ε, with at least one space (possibly more) between successive tokens or symbols on a given line, and with possibly spaces at the beginning or end of each line. Note that N and M cannot be equal to 0 and that ε cannot occur in an axiom array. The following is an example of an axiom array for $N = M = 3$:

```
A_token   X      token10
    X x X
@ A_token  x
```

Call *grammar* any sequence of lines consisting of three blocks:

- A line with the text "`# Axiom array`" (exactly as shown, with no space before or after), followed by an axiom array, possibly preceded and followed by any number of blank lines.
- A line with the text "`# Row tables`" (exactly as shown, with no space before or after), followed by N tables, $N \geq 0$, two consecutive tables being separated by at least one blank line (possibly more), and with possibly any number of blank lines before the first table and after the last one. Note that N can be equal to 0 (in which case there is no row table, but the header "`# Row tables`" should still be present).
- A line with the text "`# Column tables`" (exactly as shown, with no space before or after), followed by N tables, $N \geq 0$, two consecutive tables being separated by at least one blank line (possibly more), and with possibly any number of blank lines before the first table and after the last one. Note that N can be equal to 0 (in which case there is no column table, but the header "`# Column tables`" should still be present).

Moreover:

- These three blocks can be present in any order, and any number of blank lines can precede the first block.
- If ε occurs in one of the two blocks "row tables" and "column tables", then the other block is empty (so in case ε occurs in one of the rules, we limit ourselves to the simplest case of having nothing but row tables or nothing but column tables in our grammar).

Call *nonterminals* of a grammar all tokens that occur on the left hand side of at least one rule of at least one table of the grammar. Call *terminals* of a grammar all symbols that occur on the right hand side of at least one rule of at least one table of the grammar, but not on the left hand side of any rule of any table of the grammar.

Call *picture* any string of the form `'s_1\ns_2\ns_N'` for some $N \geq 0$, where `s_1`, ..., `s_N` are nonempty sequences of terminals, ε excluded, all of the same length. For instance, if `a` and `b` are symbols, then `'aba\nbab\naba'` is a picture. Note that N can be equal to 0, in which case the string is empty, denoting the "empty picture".

2 First task (4 marks)

Your program will first read a grammar stored in a file located in the directory where the program is located and run (you can assume that that file exists). It will have a function to check that the grammar is correct. It will have functions to, in case the grammar is correct, output a representation of the axiom array, row tables, column tables, nonterminals, and terminals of the grammar.

Your outputs have to be **exactly** as shown, up to single spaces. Note that:

- No space is output at the beginning of any line
- No space is output at the end of any line.
- The output is neither preceded nor followed by a blank line.
- Consecutive tables are separated by a single blank line.
- A table is output with left hand sides listed in lexicographic order.
- The order of the tables as stored in the file is preserved in the output.
- `->` are nicely aligned.
- The tokens and symbols that make up an axiom array and the tokens and symbols on the right hand side of the rules of a given table are nicely aligned.
- Nonterminals and terminals are listed in lexicographic order.
- ε (on the right hand side of a rule) is not output.

```
>>> from picture_grammar import *
>>> grammar = get_grammar('bad_grammar_1.txt')
Incorrect grammar
>>> grammar = get_grammar('bad_grammar_2.txt')
Incorrect grammar
>>> grammar = get_grammar('bad_grammar_3.txt')
Incorrect grammar
>>> grammar = get_grammar('bad_grammar_4.txt')
Incorrect grammar
>>> grammar = get_grammar('bad_grammar_5.txt')
Incorrect grammar
```

```
$ cat grammar_1.txt
# Column tables

S -> a S a
Y -> B Z B
X -> b X b

S -> a
Y -> A
X -> a

# Row tables

B -> b
A -> a

B -> B b
Z -> Y X

# Axiom array

a S a
B Y B
$ python3
...
>>> from picture_grammar import *
>>> grammar = get_grammar('grammar_1.txt')
>>> axiom_array, row_tables, column_tables = grammar
>>> print_pattern(axiom_array)
a S a
B Y B
>>> print_tables(row_tables)
A -> a
B -> b

B -> B b
Z -> Y X
>>> print_tables(column_tables)
S -> a S a
X -> b X b
Y -> B Z B

S -> a
X -> a
Y -> A
>>> non_terminals, terminals = symbols(grammar)
>>> print(non_terminals)
['A', 'B', 'S', 'X', 'Y', 'Z']
>>> print(terminals)
['a', 'b']
```

```
$ cat grammar_2_a.txt
# Axiom array
symbol_A1
# Row tables
symbol_A1 -> other_symbol_B1 to_become_a
other_symbol_B1 -> symbol_A1 to_become_b
# Column tables
symbol_A1 -> symbol_A1
to_become_a -> symbol_A1
to_become_b -> other_symbol_B1

symbol_A1 -> other_symbol_B1 a
other_symbol_B1 -> symbol_A1 b

symbol_A1 -> a
other_symbol_B1 -> b
$ python3
...
>>> from picture_grammar import *
>>> grammar = get_grammar('grammar_2_a.txt')
>>> axiom_array, row_tables, column_tables = grammar
>>> print_pattern(axiom_array)
symbol_A1
>>> print_tables(row_tables)
other_symbol_B1 -> symbol_A1       to_become_b
symbol_A1       -> other_symbol_B1 to_become_a
>>> print_tables(column_tables)
symbol_A1   -> symbol_A1
to_become_a -> symbol_A1
to_become_b -> other_symbol_B1

other_symbol_B1 -> symbol_A1       b
symbol_A1       -> other_symbol_B1 a

other_symbol_B1 -> b
symbol_A1       -> a
>>> non_terminals, terminals = symbols(grammar)
>>> print(non_terminals)
['other_symbol_B1', 'symbol_A1', 'to_become_a', 'to_become_b']
>>> print(terminals)
['a', 'b']
```

```
$ cat grammar_2_b.txt
# Row tables
Origin -> B1 A

A1 -> B1 A
B1 -> A1 B



# Axiom array
Origin



# Column tables
A1 -> A1
A -> A1
B -> B1


A1 -> B1 a
B1 -> A1 b


A1 -> a
B1 -> b
$ python3
...
>>> from picture_grammar import *
>>> grammar = get_grammar('grammar_2_b.txt')
>>> axiom_array, row_tables, column_tables = grammar
>>> print_pattern(axiom_array)
Origin
>>> print_tables(row_tables)
Origin -> B1 A

A1 -> B1 A
B1 -> A1 B
>>> print_tables(column_tables)
A  -> A1
A1 -> A1
B  -> B1

A1 -> B1 a
B1 -> A1 b

A1 -> a
B1 -> b
>>> non_terminals, terminals = symbols(grammar)
>>> print(non_terminals)
['A', 'A1', 'B', 'B1', 'Origin']
>>> print(terminals)
['a', 'b']
```

```
$ cat grammar_3.txt
# Axiom array
   S   *   S
   S   *   S
   S   *   S
# Row tables
S    ->    *
# Column tables
S    ->    *
$ python3
...
>>> from picture_grammar import *
>>> grammar = get_grammar('grammar_3.txt')
>>> axiom_array, row_tables, column_tables = grammar
>>> print_pattern(axiom_array)
S * S
S * S
S * S
>>> print_tables(row_tables)
S -> *
>>> print_tables(column_tables)
S -> *
>>> non_terminals, terminals = symbols(grammar)
>>> print(non_terminals)
['S']
>>> print(terminals)
['*']
```

```
$ cat grammar_4.txt
# Axiom array
A I X M
B J Y N
C K Z M

# Row tables

# Column tables
A -> A1
B -> B1
C -> A1

A1 -> ε
B1 -> ε
C1 -> ε

A2 -> ε
B2 -> ε
C2 -> ε

I -> W
J -> W
K -> Z
L -> Z

M -> A2 Y
N -> B2 Z

W -> a b
X -> a b
Y -> a b
Z -> a b

W -> A1
X -> A1
Y -> B1
Z -> B1
```

```
$ python3
...
>>> from picture_grammar import *
>>> grammar = get_grammar('grammar_4.txt')
>>> axiom_array, row_tables, column_tables = grammar
>>> print_pattern(axiom_array)
A I X M
B J Y N
C K Z M
>>> print_tables(row_tables)
>>> print_tables(column_tables)
A -> A1
B -> B1
C -> A1

A1 ->
B1 ->
C1 ->

A2 ->
B2 ->
C2 ->

I -> W
J -> W
K -> Z
L -> Z

M -> A2 Y
N -> B2 Z

W -> a b
X -> a b
Y -> a b
Z -> a b

W -> A1
X -> A1
Y -> B1
Z -> B1
>>> non_terminals, terminals = symbols(grammar)
>>> print(non_terminals)
['A', 'A1', 'A2', 'B', 'B1', 'B2', 'C', 'C1', 'C2', 'I', 'J', 'K', 'L', 'M', 'N', 'W', 'X', 'Y', 'Z']
>>> print(terminals)
['a', 'b', ε, ]
```

3 Second task (3 marks)

For this task, we consider only programs with no occurrence of ε. The program has to implement a function which tests whether a given picture is valid and in case it is, whether it can be generated and in case it can, displays the **longest deterministic final part** of the generation, that is, the longest final part of the generation which is the same for all possible generations of the picture. Again, the output has to be **exactly** as shown. Note that the width of each column is determined by the length of the largest token.

```
$ python3
...
>>> from picture_grammar import *
>>> grammar = get_grammar('grammar_1.txt')
>>> generate(grammar, 'aSa')
Picture is invalid.
>>> generate(grammar, 'aaaaaaa\nbbbabbb\nbbbabbb')
Picture cannot be generated.
>>> generate(grammar, 'aaaaaaa\nbbbabbb\nbbbabbb\nbbbabbb')
Picture can be generated in only one way.
Here it is:
a S a
B Y B

a a S a a
B B Z B B

a a S a a
B B Y B B
b b X b b

a a a S a a a
B B B Z B B B
b b b X b b b

a a a S a a a
B B B Y B B B
b b b X b b b
b b b X b b b

a a a a a a a
B B B A B B B
b b b a b b b
b b b a b b b

a a a a a a a
b b b a b b b
b b b a b b b
b b b a b b b
```

```
$ python3
...
>>> from picture_grammar import *
>>> grammar = get_grammar('grammar_2_a.txt')
>>> generate(grammar, 'abc')
Picture is invalid.
>>> generate(grammar, 'ab')
Picture cannot be generated.
>>> generate(grammar, 'ababa\nbabab\nababa\nbabab\nababa')
Picture cannot be generated in only one way.
Here is the final deterministic part:
symbol_A1

other_symbol_B1
to_become_a

symbol_A1
to_become_b
to_become_a

other_symbol_B1
to_become_a
to_become_b
to_become_a

symbol_A1
to_become_b
to_become_a
to_become_b
to_become_a

symbol_A1
other_symbol_B1
symbol_A1
other_symbol_B1
symbol_A1

other_symbol_B1 a
symbol_A1       b
other_symbol_B1 a
symbol_A1       b
other_symbol_B1 a

symbol_A1       b               a
other_symbol_B1 a               b
symbol_A1       b               a
other_symbol_B1 a               b
symbol_A1       b               a

other_symbol_B1 a               b               a
symbol_A1       b               a               b
other_symbol_B1 a               b               a
symbol_A1       b               a               b
other_symbol_B1 a               b               a

symbol_A1       b               a               b               a
other_symbol_B1 a               b               a               b
symbol_A1       b               a               b               a
other_symbol_B1 a               b               a               b
symbol_A1       b               a               b               a

a b a b a
b a b a b
a b a b a
b a b a b
a b a b a
```

```
$ python3
...
>>> from picture_grammar import *
>>> grammar = get_grammar('grammar_2_b.txt')
>>> generate(grammar, 'ababa\nbabab\nababa\nbabab\nababa')
Picture can be generated in only one way.
Here it is:
Origin

B1
A

A1
B
A

B1
A
B
A

A1
B
A
B
A

A1
B1
A1
B1
A1

B1 a
A1 b
B1 a
A1 b
B1 a

A1 b  a
B1 a  b
A1 b  a
B1 a  b
A1 b  a

B1 a  b  a
A1 b  a  b
B1 a  b  a
A1 b  a  b
B1 a  b  a

A1 b  a  b  a
B1 a  b  a  b
A1 b  a  b  a
B1 a  b  a  b
A1 b  a  b  a

a b a b a
b a b a b
a b a b a
b a b a b
a b a b a
```

```
$ python3
...
>>> from picture_grammar import *
>>> grammar = get_grammar('grammar_3.txt')
>>> generate(grammar, '***\n***\n***')
Picture cannot be generated in only one way.
Here is the final deterministic part:
* * *
* * *
* * *
```

4 Task 3 (2 marks)

For this task, we consider only grammars with occurrences of ε. The function implemented for the second part has to be extended to deal with these grammars and in case the picture can be generated, only output the message that it can be (with no consideration for whether it can be generated in many ways, and with no consideration for the final deterministic part). Whereas the empty picture can never be generated by the grammars considered in the second part, this is no longer true for this part.

```
$ python3
...
>>> from picture_grammar import *
>>> grammar = get_grammar('grammar_4.txt')
>>> generate(grammar, 'abc')
Picture is invalid.
>>> generate(grammar, '')
Picture can be generated (as ε occurs in the grammar, we will not say more).
>>> generate(grammar, 'ab\nab\nab')
Picture can be generated (as ε occurs in the grammar, we will not say more).
>>> generate(grammar, 'abab\nabab\nabab')
Picture can be generated (as ε occurs in the grammar, we will not say more).
>>> generate(grammar, 'ababab\nababab\nababab')
Picture can be generated (as ε occurs in the grammar, we will not say more).
>>> generate(grammar, 'abababab\nabababab\nabababab')
Picture cannot be generated.
```