WORKSHEET

A6.5 Backus-Naur Form and syntax diagrams

Theory of computation · A level · AQA 7517 4.4.3.1, Eduqas A500QS 1.8 · about 25 min

Name _____ Class _____ Date _____

What this lesson is about

Production rules, syntax diagrams, a grammar for robot programs, recursive descent, and why BNF can describe what a regex cannot.

- Production rules
- A robot command language
- Syntax diagrams
- Why BNF can do what a regex cannot
- Checking syntax with recursion

Questions

6 marks in all

1. In BNF, what does ::= mean? [1 mark]

- ☐ **A** Is equal to
- ☐ **B** Is defined as
- ☐ **C** Or
- ☐ **D** Is followed by

2. Using <integer> ::= <digit> | <digit><integer> and <move> ::= <direction><integer> with <direction> ::= F | B | L | R, which are valid moves? [1 mark]

Tick every answer that is true.

- ☐ **A** F20
- ☐ **B** R7
- ☐ **C** F
- ☐ **D** 20F
- ☐ **E** L005

3. Why can BNF describe some languages that regular expressions cannot? [1 mark]

- ☐ **A** BNF allows alternatives with |
- ☐ **B** BNF rules can be recursive in the middle of a rule, so they can describe nesting such as balanced brackets
- ☐ **C** BNF uses angle brackets
- ☐ **D** BNF can only describe finite languages

4. In a syntax diagram, what does a path that loops back round a box show? [1 mark]

- ☐ **A** The item may be skipped
- ☐ **B** The item may be repeated
- ☐ **C** The item is a terminal
- ☐ **D** The syntax is invalid

5. Given <s> ::= ab | a<s>b, how many times must the rule be used to produce aaabbb? [1 mark]

6. What does this program print? [1 mark]

```
def s(text, i):
    if text[i:i + 2] == "ab":
        return i + 2
    if text[i:i + 1] == "a":
        j = s(text, i + 1)
        if j != -1 and text[j:j + 1] == "b":
            return j + 1
    return -1

for t in ["aabb", "aab", "abb"]:
    print(t, s(t, 0))
```

The task: check robot programs

Write a recursive descent checker for the robot command language above and use it on the strings in `tests`. - Write one function per rule you need, each taking `(text, i)` and returning the position after the match, or -1 if it does not match. At least `program`, `command` and `integer`. - A string is valid only if `program` matches all of it. - Check the nesting with your functions. Do not use the `re` module: no regex can check matched brackets. - For each string in `tests`, in order, print the string, a space, then `valid` or `invalid`. For example `F20R90 valid`. Eight lines in all.

```
# the two lines every program starts with: the commands, then the robot
from bugbot import *
connect()

tests = ["F20R90", "[4F20R90]", "[2F10[3L5]]", "F20R", "[4F20", "4F20", "[3]", "F1]"]

def integer(text, i):
    return -1
```

Plan your program here, then type it in and press Run.

Do it on the robot

www.bugbotlab.com/learn/a6-5-bnf-and-syntax-diagrams/

The simulator checks it and tells you when it passes. Nothing to install, no account.

Challenges

1. Add a rule so a program may contain `S`, meaning stop, as a command on its own. Change both the BNF and the syntax diagram.
2. Write BNF for a signed integer that may start with `+` or `-`.
3. Is the language of your `<integer>` rule regular? Write a regex for it. Is `<program>` regular?