ANSWERS

A8.9 Project: the robot's safety logic

Boolean algebra and logic circuits · A level · OCR H446 1.4.3, AQA 7517 4.6.4.1, Eduqas A500QS 1.2 · about 30 min

BugBotLab

What this lesson is about

From a rule table to a Karnaugh map, a simplified expression, a proof and a robot that stops for the right reasons.

- The brief
- Stage 1: the truth table
- Stage 2: the sum of products
- Stage 3: the Karnaugh map
- Stage 4: simplify further, and prove it
- Stage 5: wire it to the robot

Questions

5 marks in all

1. A rule has four inputs. How many rows does its truth table have? [1 mark]

> **Answer:** 16. 2 to the power 4 is 16.

2. With rows numbered 8C + 4K + 2L + D, which row number is C = 1, K = 0, L = 1, D = 1? [1 mark]

> **Answer:** 11. 8 + 0 + 2 + 1 = 11, the binary number 1011.

3. Put the stages of designing a logic rule in order. [1 mark]

Number the lines 1 to 6 to put them in the right order.

___ `Write the simplified expression`
___ `Draw the circuit or write the code`
___ `Write the truth table`
___ `Name each input with a letter`
___ `Check it against the truth table`
___ `Draw the Karnaugh map and group the 1s`

> **Answer:**
>
> ```
> Name each input with a letter
> Write the truth table
> Draw the Karnaugh map and group the 1s
> Write the simplified expression
> Check it against the truth table
> Draw the circuit or write the code
> ```
>
> From the problem to a table, from the table to a simple expression, then prove it before building it.

4. Two groups on a map give (¬K ∧ D) ∨ (¬K ∧ ¬C ∧ ¬L). Which law turns this into ¬K ∧ (D ∨ (¬C ∧ ¬L))? [1 mark]

- ☐ **A** Distribution
- ☐ **B** De Morgan's law
- ☐ **C** Double negation
- ☐ **D** Commutation

> **Answer:** A. Distribution in reverse takes the shared ¬K out of both terms.

5. Which expression is equal to ¬C ∧ ¬L? [1 mark]

- ☐ **A** ¬(C ∨ L)
- ☐ **B** ¬(C ∧ L)
- ☐ **C** C ∨ L
- ☐ **D** ¬C ∨ ¬L

> **Answer:** A. De Morgan: ¬(C ∨ L) = ¬C ∧ ¬L, which saves one NOT gate.

The task: the robot's safety logic

The starter gives `GO_ROWS`, the truth table rows (numbered 8C + 4K + 2L + D) where G = 1, and `draft(c, k, l, d)`, which takes four bits and returns 1 on those rows and 0 otherwise. 1. Write `go(c, k, l, d)`, taking four bits, as a single `return` line using at most six of `and`, `or` and `not` in total. It may not use `GO_ROWS` or call `draft`. Any true result means drive. 2. Check `go` against `draft` on all sixteen combinations, treating a true result as 1 and a false one as 0, and print `matches: True` if they agree on every one, or `matches: False` if not. 3. Drive the robot. Set `docking = 0`. In a loop, read `close` (1 if `distance()` is under 20, else 0), `hit` (1 if `bumped()`, else 0) and `low` (1 if `battery()` is under 20, else 0), and call `go(close, hit, low, docking)`. While it is true, drive forward. When it is false, stop, print one line in exactly the form `stopped: C=1 K=0 L=0 D=0` using the values that stopped it, and end the loop. The robot must finish in the band before the wall without touching it.

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

GO_ROWS = [0, 1, 3, 9, 11]

def draft(c, k, l, d):
    return 1 if 8 * c + 4 * k + 2 * l + d in GO_ROWS else 0

def go(c, k, l, d):
    return 0
```

The hint students can ask for: Put the five rows on a four-variable Karnaugh map in Gray code order and find the fewest, largest groups; then see whether factoring out a shared literal, or De Morgan, saves operators. Prove go against draft on all sixteen rows before you let it drive. In the loop, read the four inputs as 0 or 1 every time round.

A solution

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

GO_ROWS = [0, 1, 3, 9, 11]

def draft(c, k, l, d):
    return 1 if 8 * c + 4 * k + 2 * l + d in GO_ROWS else 0

def go(c, k, l, d):
    return (not k) and (d or not (c or l))

ok = True
for n in range(16):
    c, k, l, d = n >> 3 & 1, n >> 2 & 1, n >> 1 & 1, n & 1
    if (1 if go(c, k, l, d) else 0) != draft(c, k, l, d):
        ok = False
print(f"matches: {ok}")

docking = 0
while True:
    close = 1 if distance() < 20 else 0
    hit = 1 if bumped() else 0
    low = 1 if battery() < 20 else 0
    if go(close, hit, low, docking):
```

```
        forward(50)
        wait(0.1)
    else:
        stop()
        print(f"stopped: C={close} K={hit} L={low} D={docking}")
        break
```

Any program that meets the task's checks is marked correct in the simulator; this is one way, not the only way.