COMS12200 lab. worksheet: week #2

- We intend this worksheet to be attempted in the associated lab. session, which represents a central form of help and feedback for this unit.
- The worksheet is not *directly* assessed. Rather, it simply provides guided, practical exploration of the material covered in the lecture(s): the only requirement is that you archive your work (complete or otherwise) in a portfolio via the appropriate component at

  https://wwwa.fen.bris.ac.uk/COMS12200/

  This forms the basis for assessment during a viva at the end of each teaching block, by acting as evidence that you have engaged with and understand the material.
- The deadline for submission to your portfolio is the end of the associated teaching block (i.e., in time for the viva): there is no requirement to complete the worksheet in the lab. itself (some questions require too much work to do so), but there is an emphasis on *you* to catch up with anything left incomplete.
- To accommodate the number of students registered on the unit, the single 3 hour timetabled lab. session is split into two $1\frac{1}{2}$ hour halves. You should attend *one* half only, selecting as follows:
  1. if you have a timetable clash that means you *must* attend one half or the other then do so, otherwise
  2. execute the following BASH command pipeline

     ```
     id -n -u | sha1sum | cut -c-40 | tr 'a-f' 'A-F' | dc -e '16i ? 2 % p'
     ```

     e.g., log into a lab. workstation and copy-and-paste it into a terminal window, then check the output: 0 means attend the first half, 1 means attend the second half.

Q1. There is a set of questions available directly at

http://www.cs.bris.ac.uk/home/page/teaching/material/arch_old/sheet/exam-prelims_boolean.q.pdf

or via the unit web-page: using pencil-and-paper, each asks you to solve a short, practical problem relating to Boolean algebra. There are *way* too many for the lab. session alone, but in the longer term your challenge is simple: answer all the questions, applying theory covered in the lecture(s).

Q2. In the lecture(s), a very overt effort was made to differentiate COMS12200 from other units covering Boolean algebra: our main focus, it was claimed, is on practical application. *If* you needed further convincing, this and various following questions should provide it.

An important (theoretical) result we encountered was the universality of NAND: *all* the Boolean operators, and so *all* Boolean functions, can be realised by using NAND operators alone. To support COMS12200, we developed[1] a kit that allows translation of this theory into practice: at the beginning of the lab. you should pick up

- a NAND board (revision *C*),
- a USB cable, and
- a set of coloured jumper wires (which can be carefully stripped apart from each other)

all of which you can keep[2] afterwards: although we will not use the kit in *every* lab. session, it would be useful if you try to bring it along anyway (so it is available if need be). The board needs a power source, which is provided by connecting it to the USB port of a host workstation (via the connector on the left-hand edge of the PCB).

The board houses 4 groups of 4 identical NAND operators[3], meaning 16 in total. Conceptually at least, each individual operator can be described as follows:

[1] We are very interested in both positive and negative feedback about this kit. If, for instance, you have a problem or do something interesting "off piste" with the board, we want to hear about it: take a photograph and drop us an email, for example.

[2] As with any similar device, a given board will sometimes fail (or be faulty to begin with). If your board does not work, let us know: the deal is that if the failure is our fault we will try to replace it (up to the point that we run out of supplies), but if it is *your* fault then we cannot make the same guarantee.

[3] In reality, the operators are realised using **logic gates**; we encounter the internal transistor-based design of these components later in the unit, but you can ignore this for now.

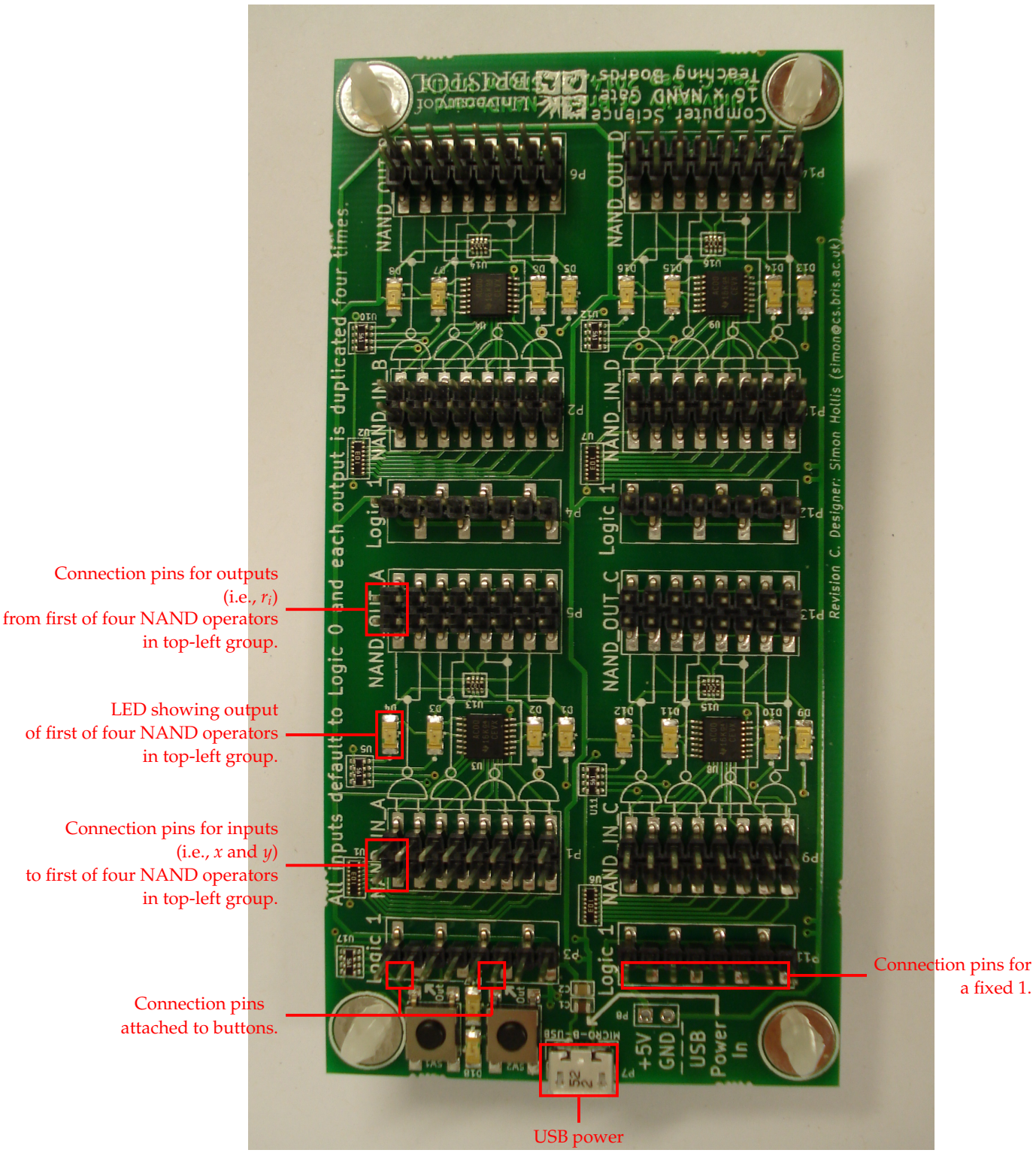
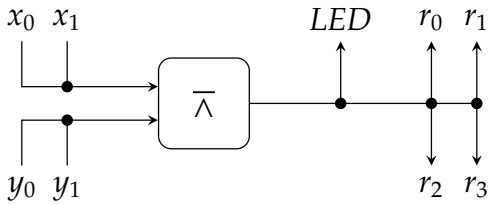


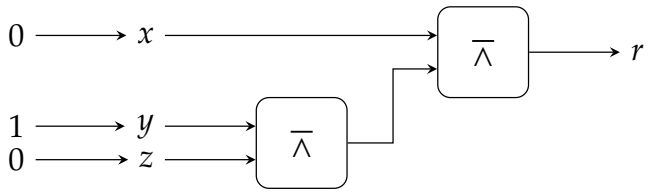
Figure 1: *An empty NAND board, with no connections (and no power).*



Note that:

- Each NAND operator produces *one* output, which is replicated on *four* different pins (i.e., *each* of r_0, r_1, r_2 *and* r_3).
- The two inputs to each NAND operator can each be provided via either one of *two* different pins (i.e., from x_0 *or* x_1, and y_0 *or* y_1). Since the input pins cannot contradict each other, you must ensure $x = x_0 = x_1$ and $y = y_0 = y_1$ st. $r_i = x \mathbin{\overline{\wedge}} y$ for $0 \leq i < 4$.
- The output of each NAND operator is visualised by an LED: an output of 0 means the LED is off, whereas an output of 1 means the LED is on.
- By default, both inputs are 0. However, we can use a jumper wire to form a connection between pins, and hence connect any input pin to either
  - a fixed 1, using pins to the left of the inputs,
  - an (unused) output pin associated with another NAND operator, *or*
  - an (unused) input[4] pin associated with another NAND operator.
- Notice that the top-left group of pins relating to 1 represent a special-case: two of them (the ones pointed to by tiny arrows) are connected to user-controlled buttons instead of being fixed to 1. These pins instead default to 0, but can be toggled to 1 by pressing the associated button.
- When your (empty, i.e., with no jumper wires) board is powered-on, each NAND operator computes $0 \mathbin{\overline{\wedge}} 0 = 1$ (meaning the associated LEDs will be on): this is a good way to check the board is working correctly before you carry on!

We already saw that a Boolean expression can be thought of Mathematically, *or* diagrammatically as a combination "blocks" that look similar to this. For example, we can evaluate the expression $r = x \mathbin{\overline{\wedge}} (y \mathbin{\overline{\wedge}} z)$ where $x = 0$, $y = 1$ and $z = 0$ by reading the following diagram from left-to-right:



As such, the NAND board allows us to *physically* implement a Boolean expression and then evaluate it by

- forming appropriate connections between the NAND operators to match the expression required (i.e., connecting the output pin of one operator to the input pin of another), then
- assigning values to each input (i.e., connect the input pin to 0 or 1 as appropriate).

Simply as a first step, convince yourself that theory and practice match: step through each row in the truth table for NAND, use jumper wires to connect the two input pins of an operator to 0 or 1 as appropriate, and verify the output is correct.

Q3. a Use the NAND board to implement

i NOT,

ii AND,

[4] At first glance, this might seem a little odd. However, the goal is to allow use of common inputs. Imagine you need to compute $x \mathbin{\overline{\wedge}} x$ for example: one way would be to connect the first input pin to x, then connect the second input pin to the first.

iii OR, and

iv XOR

operators. As above, verify that the physical computation matches what you expect in theory by stepping through each entry in the associated truth table.

b Stated generally, the **majority** function takes n inputs, and produces 1 as an output iff. $\lceil \frac{n}{2} \rceil$ or more of the inputs are equal to 1; otherwise the output is 0.

Consider the specific case of $n = 3$ inputs called a, b, and c, where the function can be written more descriptively as

$$\textsc{Maj}(a, b, c) = \begin{cases} 1 & \text{if 3 or 2 of } a, b \text{ and } c \text{ are equal to 1} \\ 0 & \text{if 1 or 0 of } a, b \text{ and } c \text{ are equal to 1} \end{cases}$$

Your goal in this question is to first design then implement this function using the NAND board. This is harder than the previous questions, so the recommended approach is to work step-by-step:

- write a truth table for the function to specify the behaviour required,
- think about how this behaviour can be realised using NOT, AND and OR operators, and write this design down on paper,
- translate your design into NAND-only form, then
- implement the result on your NAND board and verify it works correctly.

Q4[+]. This is an extended rather than core question: it caters for differing backgrounds and abilities by supporting study of more advanced topics, but is therefore significantly more difficult and open-ended. As such, you should *only* attempt the associated tasks having completed all core questions (which satisfy the unit ILOs) first; even then, there is no shame in ignoring the question, or deferring work on it until later. Note that a solution will not *necessarily* be provided.

Imagine you take your implementation of a majority gate, and connect the output to reuse it as an input. For example,

$$c = \textsc{Maj}(a, b, c)$$

describes a situation where the output is connected to c.

a Experiment with different values of the remaining inputs a and b; write a truth table to capture the behaviour you observe.

b Based on the truth table above, can you think of a use for this component?

Q5[+]. This is an optional question: since the topic is outside the unit ILOs, it is included simply for fun, to provide an interesting aside, or to highlight a resource for broader study. As such, you should *only* attempt the associated tasks iff. you have a strong interest *and* enough spare time to dedicate (e.g., having finished any assessed work).

bOOleO is a card game involving Boolean algebra. Yes, really. Using AND(r), OR(r) and XOR(r) to denote cards relating to Boolean AND, OR and XOR operators whose output is r, a full deck of bOOleO cards includes the following:

- 48 Boolean operator cards, namely
  - $8 \times$ AND(0) cards,
  - $8 \times$ AND(1) cards,
  - $8 \times$ OR(0) cards,
  - $8 \times$ OR(1) cards,
  - $8 \times$ XOR(0) cards,
  - $8 \times$ XOR(1) cards,
  - $8 \times$ NOT cards

and

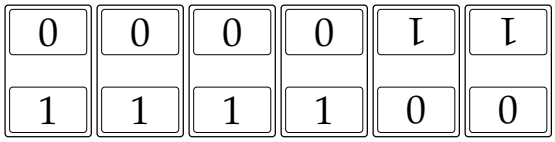
- 6 Boolean value cards (which have a 0 and 1 on).

There is a Wikipedia entry

http://en.wikipedia.org/wiki/b00le0

with a brief overview of the rules, but for completeness an alternative follows:

- Decide which player will act as the dealer: they should shuffle the operator cards, and (separately) the value cards. The dealer should then place the value cards between the players (st. the short edges face the players). An example might be as follows:



  The top player faces downward at the value cards (read from left-to right) 110000; the bottom player faces upward at the value cards 111100.

  Finally, the dealer should deal 4 operator cards to each player: this becomes their hand. All remaining operator cards form a draw deck from which players take cards (the draw deck is placed face down, meaning the card types cannot be seen).

- The goal of the game, for each player, is to form a valid tree (or pyramid) of operator cards. Some rules for forming the tree are as follows:
  - For a given player, the tree should start from (i.e., the base should be) the value cards and extend outward towards them.
  - The tree should end with (i.e., the tip should be) a single output matching the right-most value card (in the example, for the bottom player the output should be 0).
  - The inputs and outputs of operator cards in the tree should respect Boolean algebra: if one evaluates the operator (by using inputs from the layer of tree above it), the output should match the following:

x	y	valid operator card		
0	0	AND(0)	OR(0)	XOR(0)
0	1	AND(0)	OR(1)	XOR(1)
1	0	AND(0)	OR(1)	XOR(1)
1	1	AND(1)	OR(1)	XOR(0)

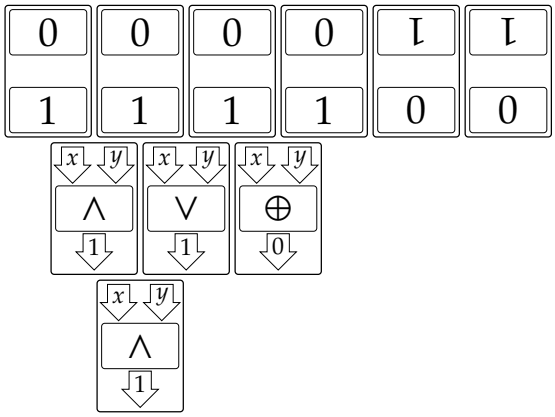
- The game is turn-based, st. one player has a turn then the other and so on until the game terminates: the non-dealer should start. Within their turn, a player
  - draws a new operator card from the draw deck, then
  - plays *or* discards an operator card from their hand

  meaning that after their turn, they *still* have a 4 card hand. If the player plays a card, this implies they will either
  - use a NOT operator card, which flips a value card (literally rotating it) selected by the player,
  - place any other operator card in the tree, thus extending it, or
  - replace any operator card in the tree.

- Some special-cases need further explanation:
  - When a card needs to be discarded, it is placed on a discard deck next to the draw deck. If/when the draw deck is empty, the dealer should shuffle the discard deck which then becomes the new draw deck.
  - When a turn is complete, one or *both* trees need to be reevaluated: any part of either tree which is invalid (e.g., the output of a given operator card no longer matches the inputs), must be discarded. The reevaluation process should start at the value cards, and progress layer-by-layer through the entire tree.

  - An operator card which is left "dangling" with no input(s), e.g., due to an operator card in the layer above having been discarded, is *not* viewed as invalid per se: rather, it is left inactive and then reevaluated when an input becomes available.
  - When a NOT operator card is played, it is removed from play permanently rather than being discarded.
  - When a NOT operator card is played, the opponent can optionally *cancel* the flipping effect by playing a NOT of their own during their turn. The need to wait and see what the opponent does suggests you should wait to see if the tree(s) need reevaluation, and only do so if the value card is flipped after all.

- Ignoring the top player, imagine the game state wrt. the bottom player is as follows:



  Assuming the relevant cards are available, some valid moves and their implication then include:

  - Extend the tree by placing an operator card next to the lower AND(1) card, e.g., XOR(1) or AND(0).
  - Update the tree by replacing the OR(1) operator card, for example: if we replace it with AND(1) then the card below it is still valid so is retained, if we replace it with XOR(0) (which is still valid wrt. the inputs) then the card below becomes invalid and is discarded.
  - Use a NOT operator, on the right-most value card whose value is 1 for example. In this case it will flip from 1 to 0, so the XOR(0) card below becomes invalid and is discarded.

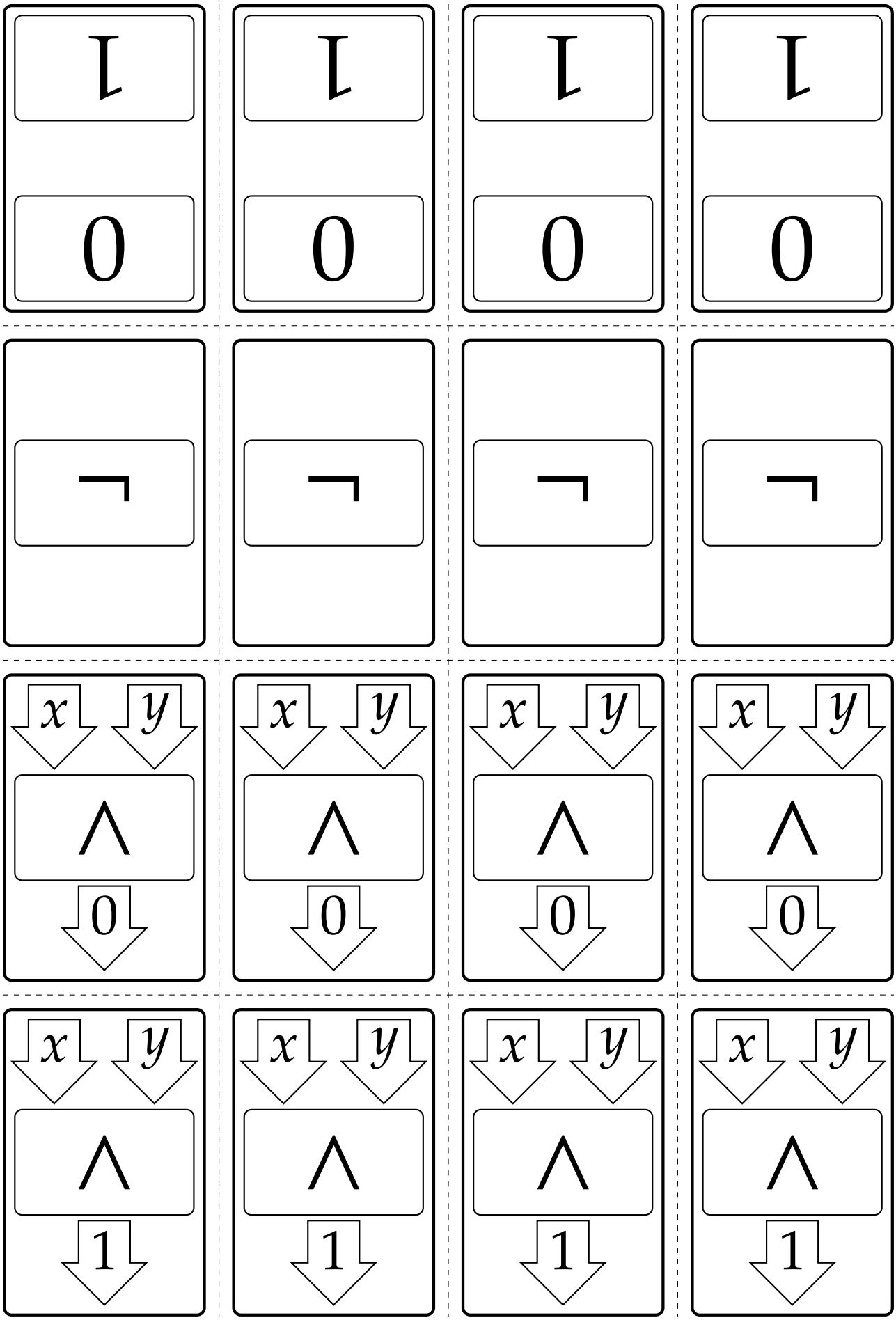
Based on this, and as a form of light relief vs. the rest of the worksheet, have a go at the following tasks:

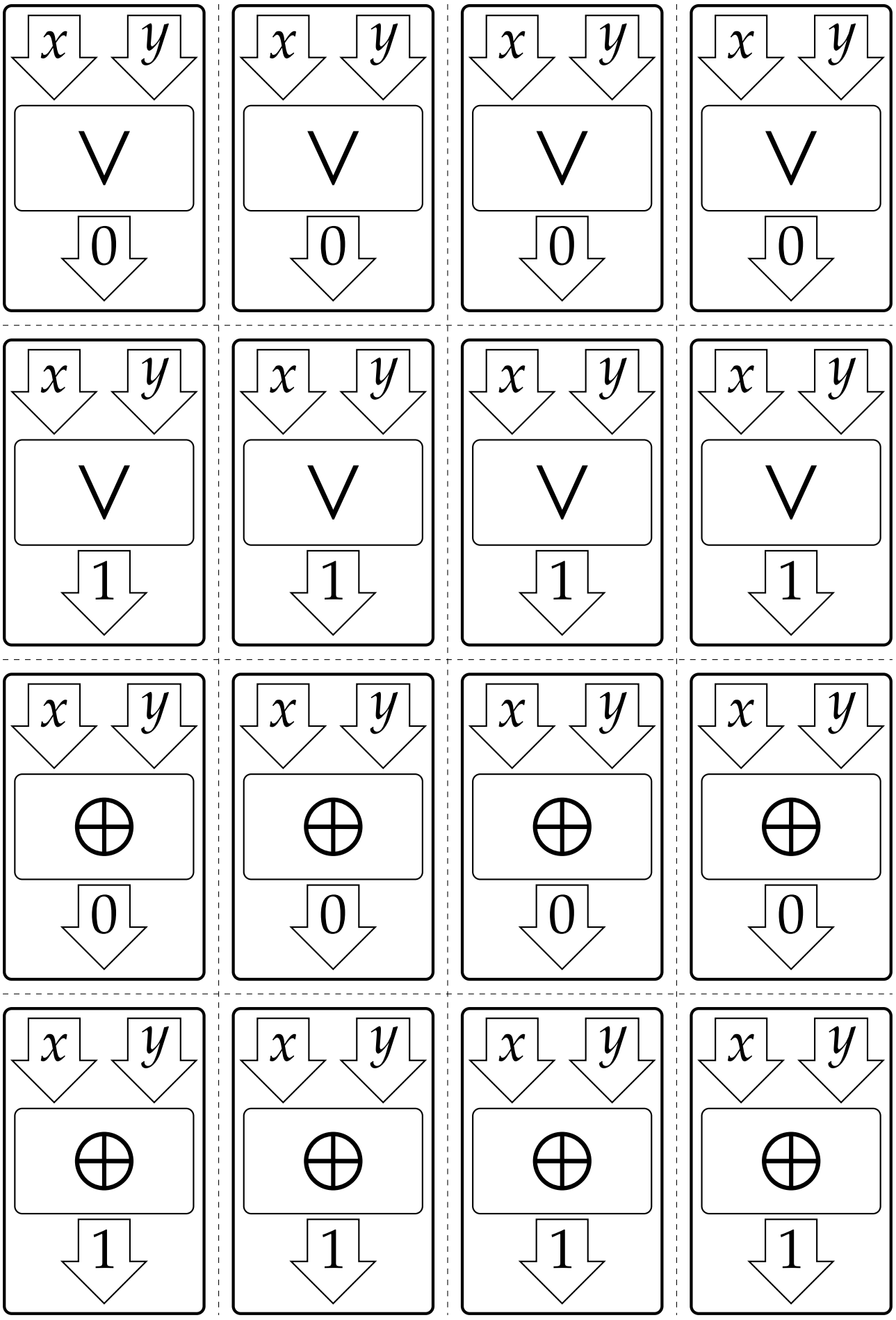
a The pages toward the end of this worksheet include enough cards for a *half* deck: this implies you first need to find an opponent to play against, and then combine your cards. Find another student, and see if you can complete a game of bOOleO.

b Based on your experience, think about the following:

  - There are various strategies relating to different aspects of the game. Think about the first row of operator cards placed below the value cards: is there a reason to favour one type over another, and why is this the case?
  - It is possible for one or other player to hit a "dead end" meaning the game cannot terminate (which is quite annoying): can you identify this case, and suggest a way to resolve the problem?
  - Various extensions or variations of bOOleO are possible; bOOleO-N, for example, introduces NAND and NOR operator cards. Can you think of any other interesting variations?

    For example, one *could* imagine a "wildcard" operator of some sort: is there a Boolean or physical justification for such an operator? Or, what happens if you add n-input, m-output operators (for $n > 2$ and/or $m > 1$): can you think of any interesting examples? Or, what about involving more than two players?


⊥
0
¬
x
y
∧
0
x
y
∧
1


x
y
∨
0
x
y
∨
0
x
y
∨
0
x
y
∨
0
x
y
∨
1
x
y
∨
1
x
y
∨
1
x
y
∨
1
x
y
⊕
0
x
y
⊕
0
x
y
⊕
0
x
y
⊕
0
x
y
⊕
1
x
y
⊕
1
x
y
⊕
1
x
y
⊕
1