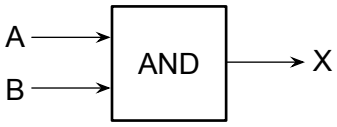
DCC888 - Static Analysis of Programs

Name: _____ **ID**: _____

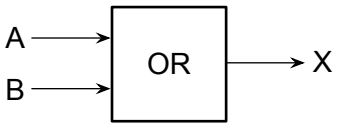
Name: _____ **ID**: _____

By turning this exam in, I give my word that I have done it with my team only, on the understanding that we are allowed to consult any material publicly available, except material disclosed by other colleagues outside our team who are taking this course, or who have taken it in the past.

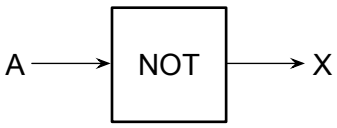
Goal: The goal of this exercise is to design static analyses for a hardware description language. This language is formed by blocks, whose graphical representation is given below.



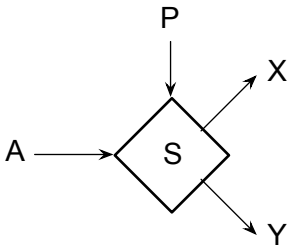
Enconding of the function X = A and B, e.g.: A = 0 => X = 0; B = 0 => X = 0; A = 1 and B = 1 => X = 1; A = undef (and B != 0) or B = undef (and A != 0) => X = undef



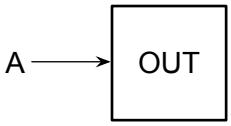
Enconding of the function X = A or B, e.g.: A = 1 => X = 1; B = 1 => X = 1; A = 0 and B = 0 => X = 0; A = undef (and B != 1) or B = undef (and A != 1) => X = undef



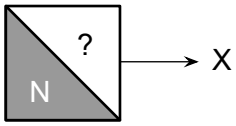
Enconding of the function X = not A, e.g.: A = 0 => X = 1; A = 1 => X = 0; A = undef => X = undef



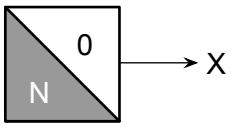
Enconding of a selector, e.g.:
P = 0 => X = A and Y = undef
P = 1 => Y = A and X = undef
P = undef => X = undef and Y = undef



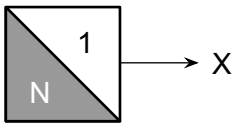
Output function, e.g.: A = 0 => print(0); A = 1 => print(1); A = undef => print(undef)



Random input, e.g., N > 0 => X = 0 or X = 1 at random; N <= 0, X = undef

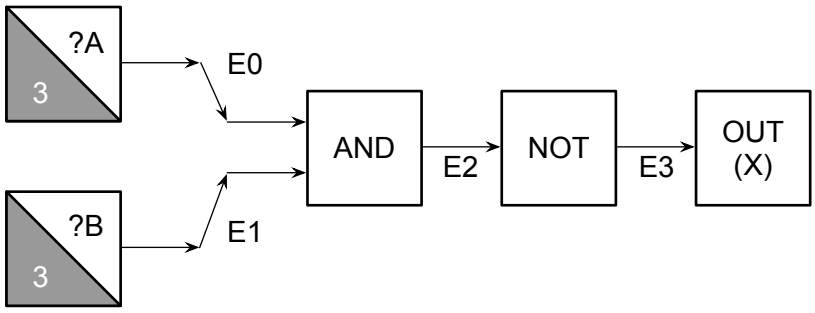


Zero input, e.g., N > 0 => X = 0; N <= 0, X = undef



Zero input, e.g., N > 0 => X = 1; N <= 0, X = undef

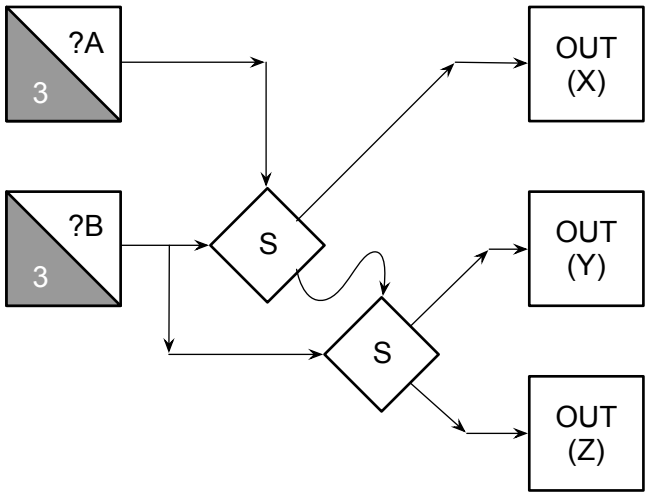
These blocks are combined into programs. Programs are controlled by a global clock. Each time the clock ticks, every block evaluates its inputs, and provides an output. The semantics of the blocks is always the same, except for the input blocks. At each click, the counter associated with the input block decreases by one. Below you can see an example of a function that prints A NAND B in four ticks.



For instance, if we have A = 0 and B = 1, then we have the following states for the edges at different moments:

tick	CA	CB	E0	E1	E2	E3	X
1	3	3	0	1	u	u	u
2	2	2	0	1	0	u	u
3	1	1	0	1	0	1	u
4	0	0	u	u	0	1	1
5	0	0	u	u	u	1	1
6	0	0	u	u	u	u	1
7	0	0	u	u	u	u	u

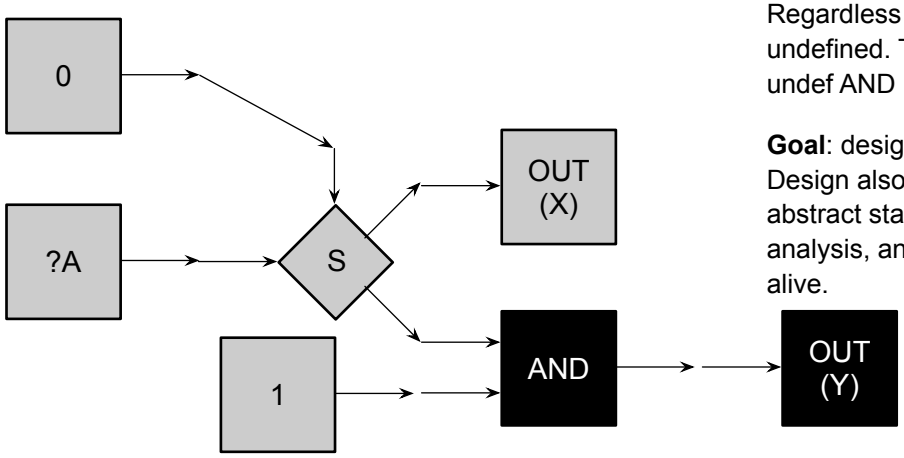
As another example, consider the program below, which implements a two-input function:



This program produces some output in either X, Y or Z, depending on the initial values of A and B, after either two iterations (if X is well-defined) or after three iterations (if either Y or Z is well defined):

A	B	X	Y	Z
0	0	0	u	u
0	1	1	u	u
1	0	u	0	u
1	1	u	u	1

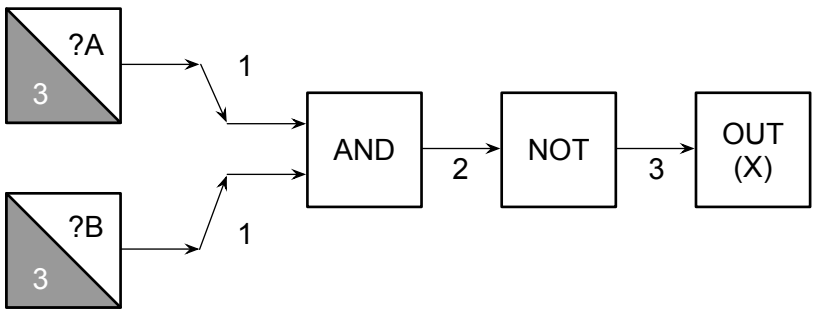
Question 1 [5 Points] Programs might contain dead code. To see why, consider the program below:



Regardless of the value of A, the output Y will be always undefined. The AND function is also undefined, because undef AND 1 is always undef.

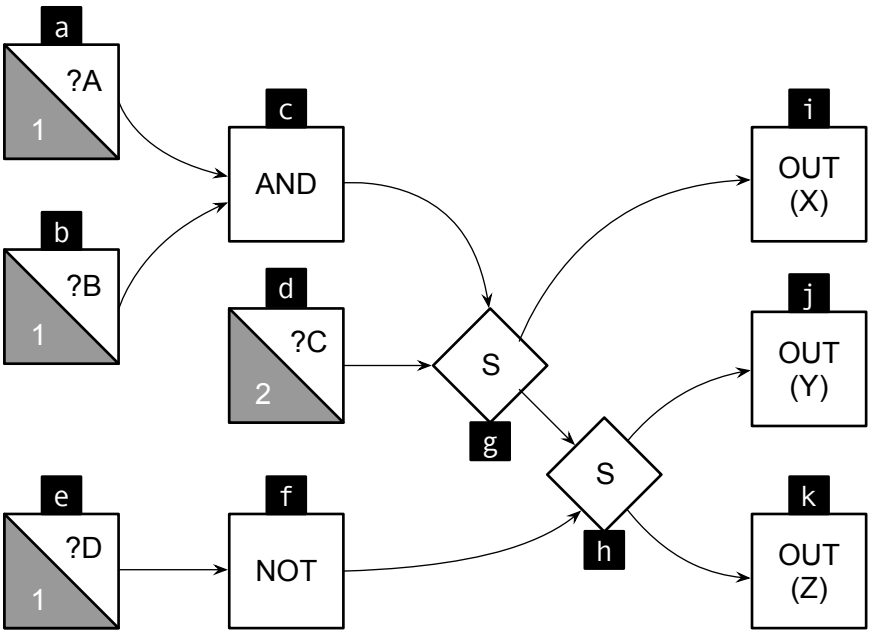
Goal: design a static analysis that marks gates as DEAD. Design also a query IS_DEAD(A), which receives the abstract state of a variable A, as computed by your static analysis, and reports if A MUST be dead or if it MIGHT be alive.

Question 2 [5 Points] Edges are *undefined* if they receive an undef value. Edges that hold either zero or one are said to be *well-defined*. Circuits might start with undefined edges, which eventually become well-defined. For instance, in the program below, we show, associated with each edge, the first tick when that edge becomes well-defined.



Goal: design a static analysis to predict the moment when each edge becomes well-defined. If your analysis says that the edge will never become well defined, you must bind that edge to the abstract state INFTY. Otherwise, you must bind that edge to the abstract state [N], whereas N is the first tick when the edge becomes well defined. Design a query WAKE_UP(A), that receives the abstract state of a variable A, and informs the moment when the variable A first became well-defined.

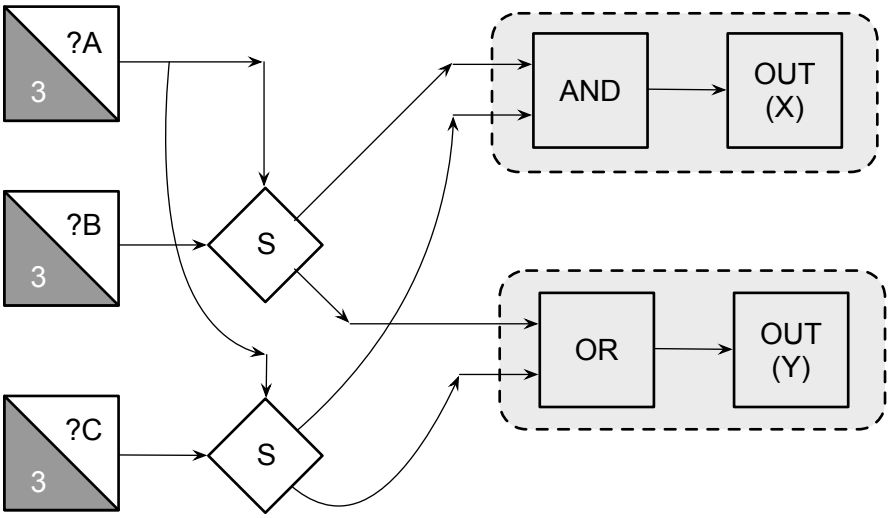
Question 3 [5 Points] Some gates can be interpreted in parallel. In other words, whenever there are no dependencies between two gates, they can run simultaneously, because the result of a gate will not influence the result of another one. For an example, see the program below:



In this program, Gate 'a' can influence gates 'c', 'g', 'i', 'h', 'j' and 'k', but gate 'a' cannot influence gate 'f', for instance.

Goal: In this exercise, you must design a static analysis that determines if a gate A can influence the outcome of another gate B. Design your analysis so that, once it runs, the result of the query M_INF(A, B) is true if A can influence the result of B. M_INF must consult the abstract states of A and B, and based on this query, return an answer.

Question 4 [5 Points] When porting our hardware description language to an actual hardware, it is necessary to implement the gates using resources, like flip-flops and memory. Some gates can be implemented onto the same resources, because they can never execute together. For instance, consider the program below:



This program computes X = B AND C if A is zero, and Y = B OR C, if A is one. What is interesting about this program is that the gates in the gray regions will never be simultaneously defined. Therefore, these gates can be implemented using the same resources. Or, in other words, there is no problem in allocating these gates onto the same flip-flops, or simulating them in the same memory.

Goal: design an analysis that tells if two gates will never receive well-defined inputs simultaneously. Notice that this is the hardest exercise. You do not need to find the most precise answer to this analysis. An approximation is fine. Try to think about what would be the abstract state associated with each gate, and try to give at least one example where your analysis produces a non-trivial result, e.g., given two gates, A and B, it tells that they might, indeed, overlap. The product of this exericse is a function MIGHT_OVERLAP(A, B) which is true if A and B might be allocated to the same resources, e.g., they are never well-defined at the same clock tick.

Final remarks: For every static analysis in this exam, you must explain what is the lattice that it uses, what are the variables that it considers (the entities that you associate abstract info with) and which abstract information is bound to each variable. You must also explain what is the time and space complexity to run the analysis, and to answer queries. For instance, you can run MIGHT_INFLUENCE in O(E), where E is the number of edges in the program, and you can answer queries M_INF in O(1). Also, notice that the interpretation of each question is part of the question itself. You must explain what are the assumptions that you are using to solve that exercise. Try to provide examples of how your analysis works, so that these assumptions can be made clear in your answer.