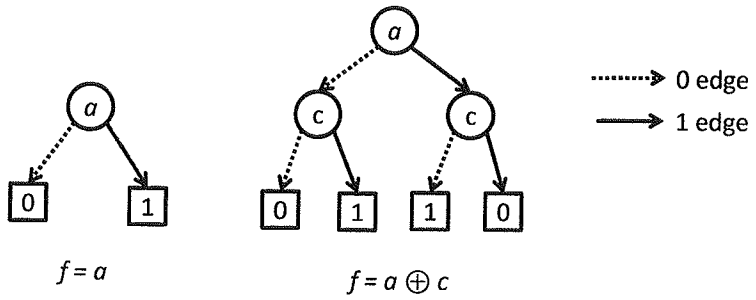
1 Boolean Functions (30 points)

Boolean functions can be represented in several ways, such as a truth table or a sum-of-products expression. This question explores two graph-based representations of Boolean functions, namely Binary Decision Trees and Binary Decision Diagrams.

1.1 Binary Decision Trees (10 points)

A *Binary Decision Tree* (BDT) is a representation of a Boolean logic function as a decision tree. A BDT has *internal nodes* and *leaves*. Each internal node is shown as a circle, and represents a decision on the input variable that it is labeled with. Each internal node has exactly two outgoing edges, which correspond to the input variable taking a value of 0 (left edge, drawn as a dotted line) or 1 (right edge, drawn as a solid line). Each leaf is shown as a rectangle, representing one of the two possible logic values: 0 or 1.

The following are two examples of BDTs (f is the output; a and c are the inputs).



Question: Draw the BDT for the function f whose truth table is given below.

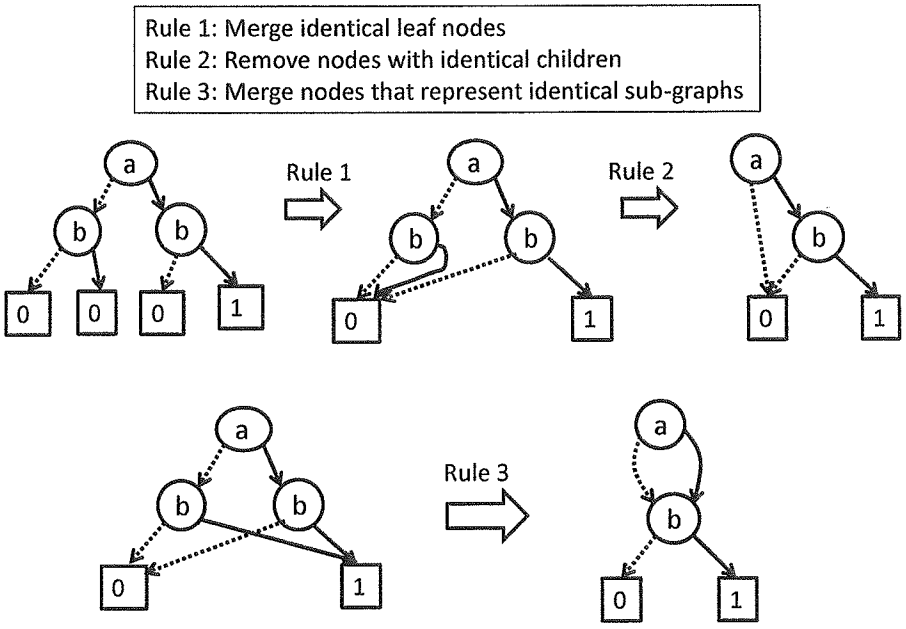
a	*b*	*c*	*f*
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

Write in Exam Book Only

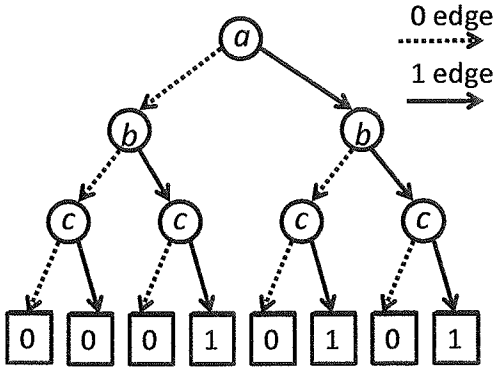
1.2 Binary Decision Diagrams (20 points)

Binary Decision Diagrams (BDDs) are graph-based representations that are, in general, more compact than BDTs. Like BDTs, BDDs also have *internal nodes* and *leaves*. Each internal node is shown as a circle, and represents a decision on the input variable that it is labeled with. Each internal node has exactly two outgoing edges, which correspond to the input variable taking a value of 0 (left edge, drawn as a dotted line) or 1 (right edge, drawn as a solid line). Each leaf is shown as a rectangle, representing one of the two possible logic values: 0 or 1.

It is possible to construct a BDD representation for a function from a BDT, by recursively applying a set of rules that are stated and illustrated in the following figure.



Question: Apply the rules presented above to derive a maximally reduced BDD from the BDT given below.



Write in Exam Book Only

2 Minimizing Combinational Logic (40 points)

This question deals with minimization of combinational logic for two-level implementation, and the design of complex CMOS gates that implement logic functions.

2.1 Two-level minimization (15 points)

Question: Design a minimal sum-of-products implementation for the multi-output function specified by the following equations. Your implementation should use a minimum number of product terms for both functions combined.

$f_1 = xy'z + xyz + x'yz$
$f_2 = x'y'z + xy'z + x'yz + x'yz' + xyz'$

2.2 Two-level minimization (10 points)

Question: What is the number of product terms in the minimum two-level (AND-OR) implementation of the function $f = x_1 \oplus x_2 \oplus x_3 \ldots \oplus x_n$? Justify your answer.
Note: $\oplus$ represents the XOR function.

2.3 Transistor-level implementation (15 points)

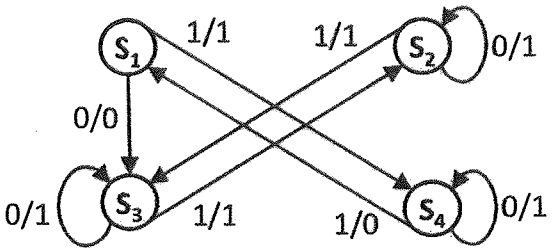
Question: Draw a transistor-level implementation of function f_2 from Question 2.1 using a complex CMOS gate and no inverters. Assume that all inputs are available in original and complemented forms.

3 Minimizing Sequential Circuits (30 points)

This question deals with minimizing sequential circuits that are specified as finite state machines, and in structural form (*i.e.*, as a network of gates and flip-flops).

3.1 Minimizing Finite State Machines (15 points)

In a *Mealy* finite state machine, the output depends on both the current state and the input. The input and output are annotated to each state transition, and represented as "input / output". The following is an example of a Mealy state machine with four states S_1, S_2, S_3, and S_4.



- If the current state is S_1:
  - If the input is 0, the machine goes to state S_3 and generates ouput 0.
  - If the input is 1, the machine goes to state S_4 and generates ouput 1.
- If the current state is S_2:
  - If the input is 0 the machine stays in state S_2 and generates ouput 1.
  - If the input is 1, the machine goes to state S_3 and generates ouput 1.

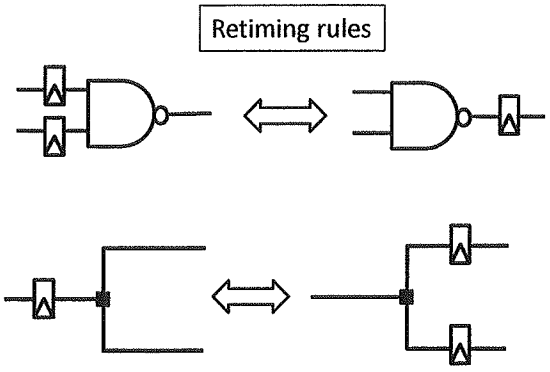
It is often desirable to design a state machine with the minimum number of states. This is usually achieved by *merging equivalent states.* Two states are equivalent if and only if, for any input, the two states have identical outputs and the corresponding next states are equivalent.

Question: Consider the state machine shown in the above figure. Minimize the state machine and draw the minimized machine.

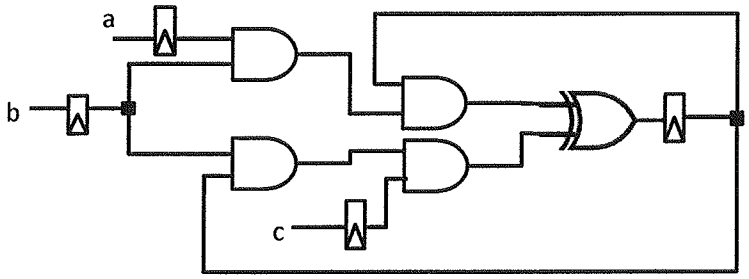
3.2 Retiming Sequential Circuits (15 points)

Retiming is a structural technique used to optimize sequential circuits without converting them into a finite state machine representation. Retiming consists of moving sequential elements (latches or flip-flops) across gates in order to position them at desirable locations so as to minimize a desired metric such as the number of flip-flops or the circuit delay. Any legal retiming can be expressed by applying a sequence of transformations to the circuit based on the rules shown in the following

figure. The first rule illustrates that a flip-flop appearing at a gate's output can be replaced by a flip-flop at each of the gate's inputs, and vice-versa. The second rule illustrates that a flip-flop on a fanout stem can be replaced by a flip-flop on each of the fanout branches, and vice-versa.



Question: Consider the sequential circuit shown in the figure below. Your goal is to apply retiming so as to minimize the number of flip-flops in the circuit. Draw the retimed circuit. How many flip-flops does the retimed circuit have?



Write in Exam Book Only