

CIS 540 Fall 2009 – Homework 2 Solutions

October 25, 2009

Problem 1

(a) We can choose a simple ordering for the variables: $x_1 < x_2 < x_3 < x_4$. The resulting OBDD is given in Fig. 1.

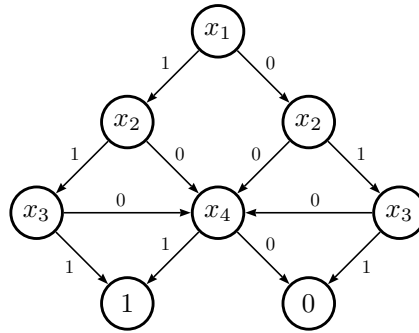


Figure 1: OBDD of Problem 1a.

The size of the OBDD can be reduced by reordering the variables: $x_2 < x_3 < x_4 < x_1$. The new OBDD is given in Fig. 2. This OBDD has optimal size because each variable corresponds to exactly one node in

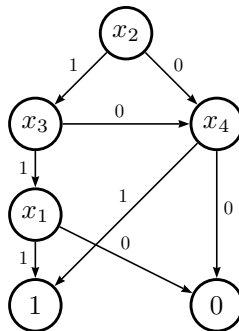


Figure 2: Size-reduced OBDD of Problem 1a

the diagram. The diagram has four nodes for four variables and two nodes for the constants 0 and 1.

(b) The expressions for z_0 , z_1 and c can be written as:

$$\begin{aligned}\varphi_0 &:= [z_0 = x_0 \oplus y_0] \\ \varphi_1 &:= [z_1 = (x_0 \wedge y_0 \wedge \neg(x_1 \oplus y_1)) \vee ((\neg x_0 \vee \neg y_0) \wedge (x_1 \oplus y_1))] \\ \varphi_c &:= [c = (x_1 \wedge y_1) \vee (x_0 \wedge y_0 \wedge (x_1 \vee y_1))]\end{aligned}$$

where \oplus is the *exclusive or* operator: $a \oplus b = (a \wedge \neg b) \vee (\neg a \wedge b)$. The complete expression is then $\varphi = \varphi_0 \wedge \varphi_1 \wedge \varphi_c$.

In order to obtain an optimal ordering of the variables, we consider the dependency between the variables. We note that:

- The value of z_0 depends on both x_0 and y_0 ;
- The value of z_1 depends on all x_0 , y_0 , x_1 and y_1 ;
- The value of c depends on x_0 , y_0 , x_1 and y_1 , but in some cases, the value is decided when x_0 , y_0 and either x_1 or y_1 are known;
- The order between x_0 and y_0 , x_1 and y_1 are not important because the roles of the variables in each pair are symmetric.

From the above observations, we can choose an optimal ordering as follows: $x_0 < y_0 < z_0 < x_1 < c < y_1 < z_1$. The resulting OBDD is given in Fig. 3.

Problem 2

(a) The process Split has one input queue (x) and two output queues (y_1 and y_2); it transfers messages from x to y_1 and y_2 in a nondeterministic manner.

- Input variables:

msg in

- Output variables:

msg out_1, out_2

queue(msg) $x, y_1, y_2 : x = \mathbf{null} \wedge y_1 = \mathbf{null} \wedge y_2 = \mathbf{null}$
$A_{in} : [\neg \text{Full}(x) \wedge x' = \text{Enqueue}(in, x) \wedge \text{same}(y_1, y_2)]$
$A_{out_1} : [\neg \text{Empty}(y_1) \wedge out_1 = \text{Front}(y_1) \wedge y_1' = \text{Dequeue}(y_1) \wedge \text{same}(x, y_2)]$
$A_{out_2} : [\neg \text{Empty}(y_2) \wedge out_2 = \text{Front}(y_2) \wedge y_2' = \text{Dequeue}(y_2) \wedge \text{same}(x, y_1)]$
$A_1 : [\neg \text{Empty}(x) \wedge \neg \text{Full}(y_1) \wedge y_1' = \text{Enqueue}(\text{Front}(x), y_1) \wedge x' = \text{Dequeue}(x) \wedge \text{same}(y_2)]$
$A_2 : [\neg \text{Empty}(x) \wedge \neg \text{Full}(y_2) \wedge y_2' = \text{Enqueue}(\text{Front}(x), y_2) \wedge x' = \text{Dequeue}(x) \wedge \text{same}(y_1)]$

Internal actions A_1 and A_2 transfer messages in the input queue x to output queues y_1 and y_2 , respectively, when x is not empty and the corresponding output queue is not full.

(b) The fairness specification is as follows:

- Weak fairness for output actions A_{out_1} and A_{out_2} because whenever the corresponding output queue is not empty, the action must eventually be executed to output the messages in the queue to the output channel.

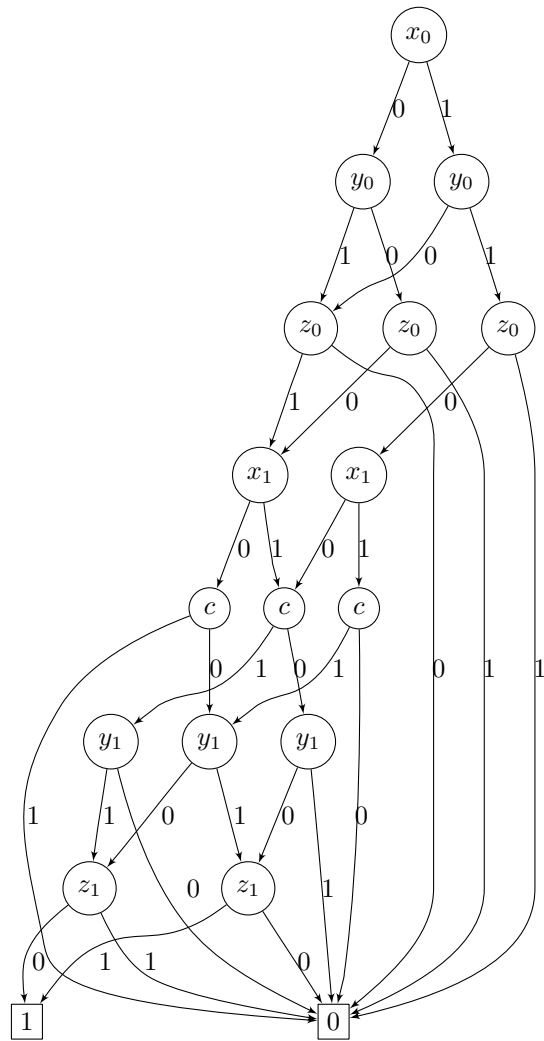


Figure 3: OBDD of Problem 1b

- Strong fairness for internal actions A_1 and A_2 because we want to prevent the situation where A_2 (A_1) is repeatedly executed, causing A_1 (A_2) repeatedly enabled but not executed. In other words, if A_1 (A_2) is repeatedly enabled then it must be repeatedly taken.

Problem 3

*Note: the problem becomes very simple if the value set is the Boolean set: for any number of processes, only a single **StickyBit** is needed. In this solutions, we consider the case where the value set can be any set, which is more complicated and more interesting.*

(a) The **StickyBit** shared object has a nice property: the first process that writes to the object will have the decision on the value of the object. We utilize this property to implement a protocol for consensus of processes.

Let's say we have two processes, P_0 and P_1 , with respective initial values v_0 and v_1 of the same type. We will use two atomic registers x_0 and x_1 (whose value sets are the same as the type of v_0 and v_1), and a single **StickyBit** s . Each process performs the following sequence of steps, where $i = 0, 1$:

1. Write its initial value v_i to register x_i .
2. Write i to s .
3. Read the value j from s .
4. Read the value d from x_j , then set its decision value d_i to d .

The protocol satisfies the three requirements of consensus:

Agreement: The first process that writes to s will decide the value of s , either 1 (if process P_1 writes first) or 0 (if process P_0 writes first). Both processes agree on the value j of s , hence agree on the decision value d read from x_j . Note that the value of x_j must be not null because the process P_j had written v_j to x_j before it wrote to s .

Validity: Clearly, the decision value d is the value of either x_0 or x_1 , which is the initial value of either P_0 or P_1 .

Wait-freedom: If only one process P_i is repeatedly executed, clearly it will decide on its initial value v_i , and the other process, when it is executed, will also decide on this value.

(b) For n processes, denoted P_1 to P_n , we can solve the consensus problem using multiple **StickyBit** and **AtomicReg** objects. As in (a), we use n atomic registers, x_1 to x_n , to store the initial values of the n processes.

The tricky part is to decide on which register the processes will agree. We cannot use **StickyBit** in a simple way as in part (a). Instead, we build a binary tree whose n leaf nodes correspond to the n processes and whose $n - 1$ other nodes correspond to **StickyBit** objects. For example, a binary tree for a consensus problem with 5 processes can be built as in Fig. 4, in which there are four **StickyBit** objects s_1, s_2, s_3, s_4 . The idea is to decide on a (unique) path from the root node to a leaf node, represented by the values of the **StickyBit** objects. For example, if the values of s_1, s_2, s_3, s_4 are 1, 0, 1, 0 respectively then the decision path is $s_1 \rightarrow s_2 \rightarrow s_4 \rightarrow P_3$ and the decision value will be the initial value of P_3 . The decision path will be built up from the leaf nodes to the root node in the following consensus protocol for each process P_i , $i = 1, \dots, n$:

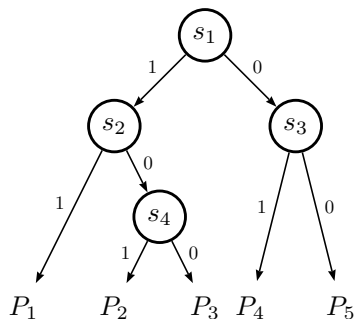


Figure 4: Binary tree for a consensus problem with 5 processes (Problem 3b).

1. The process writes its initial value v_i to register x_i .
2. Suppose that in the binary tree, the path from the root node to the leaf node P_i is $s_{j_1} \rightarrow s_{j_2} \rightarrow \dots \rightarrow s_{j_k} \rightarrow P_i$ for $k \geq 1$. The process then follows this path from s_{j_k} up to s_{j_1} and writes to the **StickyBit** object s_{j_m} , $m = k, \dots, 1$, the value 0 if the sub-path $s_{j_{m+1}} \rightarrow \dots \rightarrow P_i$ is on the right of s_{j_m} , and the value 1 otherwise.
3. The process reads the values of the **StickyBit** objects, starting from the root node, and follows the path until it reaches a leaf node P_j . The decision value d_i is then the value read from the register x_j .

The protocol satisfies the three requirements of consensus:

Agreement: When the **StickyBit** object corresponding to the root node is written, a unique path in the binary tree from the root node to a leaf node is defined. Because every **StickyBit** object retains its value once it is written, this decision path is fixed. Each process then reads this path from the tree and extracts the decision leaf node P_j . The initial value of P_j is then the decision value on which all processes agree. Note that v_j had been written to x_j before process P_j wrote to its parent node (which is a **StickyBit** object).

Validity: Clearly, the decision value d is the value of one of the initial values of the processes.

Wait-freedom: If only one process P_i is repeatedly executed, clearly it will decide on its initial value v_i , and the other processes, when executed, will also decide on this value.

Problem 4

- (a) The status of the processes after each phase are summarized in the following table.

Phase	Active nodes at end of phase ([new ID]([original ID]))
1	25 (3), 19 (8), 14 (4), 22 (21), 24 (1)
2	25 (8)

Therefore, the process whose original ID is 8 will be elected as the leader.

- (b) The best-case scenario happens when the identifiers of the processes are in a monotonically increasing or decreasing order. After the first round, only one node remains active, which is the one after the node

with the highest identifier. As an example, consider the following ring of 10 nodes where the identifiers of the processes in the order in which they are connected are: 1, 2, 4, 6, 7, 8, 10, 13, 14, 19. After the first round, the only node staying active is the one with original identifier 1.

The worst-case scenario is more complicated. Assume that the identifiers of the processes are $1, 2, \dots, n$ where n is a power of 2. We would like to place the nodes in the ring so that: after the first phase, the nodes with identifiers $1, \dots, \frac{n}{2}$ become inactive; after the second phase, the nodes with identifiers $\frac{n}{2} + 1, \dots, \frac{3n}{4}$ become inactive; and so on. We will place the nodes in the ring backward from the final phase to the first phase, as following:

- First, place the two final nodes in the increasing order: $n - 1, n$.
- Place the next two nodes ($n - 3$ and $n - 2$), in the increasing order, interleaving between the two final nodes: $n - 3, n - 1, n - 2, n$.
- Place the next four nodes ($n - 7$ to $n - 4$), in the increasing order, interleaving between the four already-placed nodes: $n - 7, n - 3, n - 6, n - 1, n - 5, n - 2, n - 4, n$.
- Repeat the step with the next eight nodes, then the next sixteen nodes, and so on until all nodes are placed in the ring.

As an example, consider the 16 nodes in part (a). We can re-order them to create a worst-case scenario as follows: 1, 15, 3, 22, 4, 18, 6, 24, 7, 19, 8, 23, 10, 21, 14, 25. It is easy to verify that after each phase, exactly half of the processes continue to stay active.