# State reduction

- Sometimes when we derive a state diagram or a state table we might end up with more states than are really required.
- Since the number of flip flops and the complexity of the circuit is largely dependent on the number of states, we should always ask ourselves if there is a solution to our problem that requires *fewer states*.
- The best way to reduce the number of states is through *state reduction* which helps us identify whether or not states are *equivalent* to each other — equivalent states can be combine together into a single state.

# State reduction

- A pair of states is said to be equivalent if:
  1. For every possible circuit input combination, the states give exactly the same circuit output; and
  2. For every possible circuit input combination, the states transition to the same *next state* or the same *equivalent state*.
- In effect, what these conditions mean is that, for a pair of states which are equivalent, they will produce the same outputs (all the time) and will transition to the same set of next states (all the time) for all possible input combinations — the states "look" and "behave" the same so they can be merged into a single state.

# State reduction

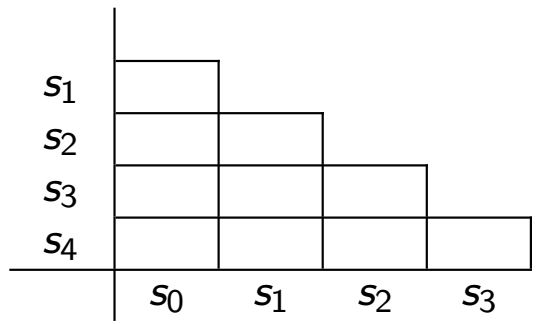
- State reduction can be accomplished using a *implication chart* and a *merger diagram*. The implication chart helps us identify whether or not pairs of states are equivalent. The merger diagram helps us decide and check whether or not a particular merging of states is correct.
- Example... consider the following state table...

| **Current State** | **Next State** | | **Output ($z$)** | |
|---|---|---|---|---|
| | $a = 0$ | $a = 1$ | $a = 0$ | $a = 1$ |
| $s_0$ | $s_3$ | $s_2$ | 1 | 1 |
| $s_1$ | $s_0$ | $s_4$ | 0 | 0 |
| $s_2$ | $s_3$ | $s_0$ | 1 | 1 |
| $s_3$ | $s_1$ | $s_3$ | 0 | 0 |
| $s_4$ | $s_2$ | $s_1$ | 0 | 0 |

This state table is for a circuit that has one input $a$, one output $z$ and 5 states. With 5 states, we would require a minimum of 3 flip flops.

# State reduction

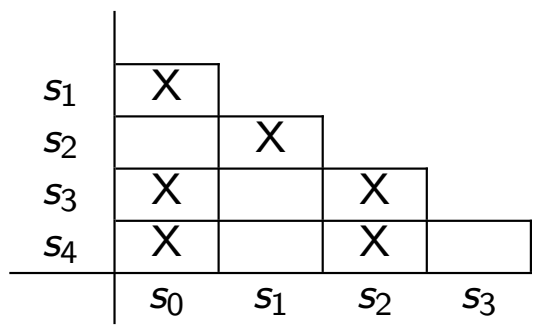
- We start by creating an implication chart. The implication chart will tell us: which states are *definitely not equivalent*, which states are *definitely equivalent* and which states are *equivalent under conditions*.
- The implication chart looks like the lower triangle of a matrix. Each entry $(i, j)$ tells us about the equivalency of state $i$ and state $j$.
- Example... a blank implication chart for our example...




- We need to fill in the implication chart appropriately.

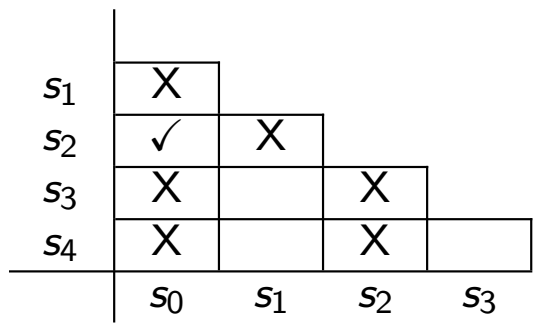
# State reduction

- Step 1: We should pick pairs of states and compare their outputs *for every possible input combination*. If, for *any* input, the outputs are different, the states *cannot* be equivalent — this makes sense because to be equivalent, the outputs must be the same for every input combination. If not equivalent, we mark the appropriate entry in the implication chart with an $X$.
- For example, if we compare the outputs for $s_0$ and $s_1$, we see that they must produce different output values when $a = 0$ (and also when $a = 1$). Therefore, this pair of states cannot be equivalent. If we compare states $s_0$ and $s_2$, we see that the output values are the same for $a = 0$ and $a = 1$ (in other words, for all input combinations). We cannot yet claim $s_0$ and $s_2$ are equivalent (we haven't checked the next state conditions), but they might be. Comparing outputs only allows us to determine when states cannot be equivalent.
- The updated implication chart showing states which cannot be equivalent due to the output values is:



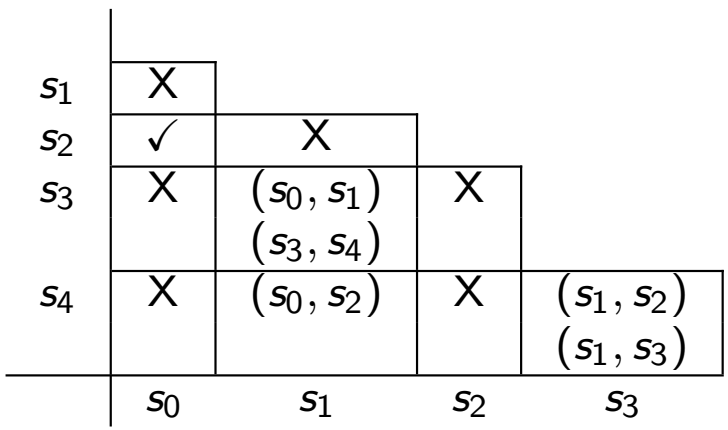
# State reduction

- Step 2: We should mark entries that are *definitely* equivalent because they go to same next state under every input combination. Note that if comparing two states $i$ and $j$... If $i$ goes to $j$ and $j$ goes to $i$ under some input condition, this is obviously okay since we are comparing $i$ and $j$. We don't need to consider states that have previously been marked as not equivalent.
- For example, if we compare the outputs for $s_0$ and $s_2$, we see that when $a = 0$, they both transition to $s_3$. When $a = 1$, $s_0$ goes to $s_2$ and $s_2$ goes to $s_0$. Therefore, states $s_0$ and $s_2$ are definitely equivalent.
- The updated implication chart showing states which are definitely equivalent:



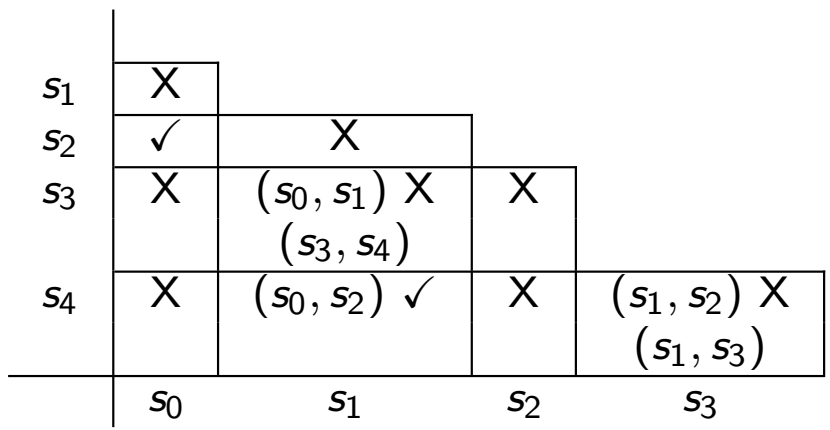
# State reduction

- Step 3: We should mark entries that have *conditions* for equivalency. A condition is along the lines of "$i$ and $j$ are equivalent if $k$ and $l$ are equivalent".
- For example, if we compare $s_1$ and $s_4$, we see that when $a = 0$, $s_1$ transitions to $s_0$ and $s_4$ transitions to $s_2$. When $a = 1$, $s_0$ transitions to $s_4$ and $s_4$ transitions to $s_1$. Therefore, $s_1$ and $s_4$ are equivalent **assuming** that $s_0$ and $s_2$ are equivalent (which we may or may not know yet).
- We enter the conditions for equivalency into the implication chart which gives:



# State reduction

- Step 4: Review the implication chart and attempt to decide whether or not conditions are true or false. When any condition fails, the the associated states cannot be equivalent. We might need to make more that one pass through the chart.
- For example, $s_0$ and $s_1$ are not equivalent. Therefore, the condition $(s_0, s_1)$ is false. This means that $s_1$ and $s_3$ cannot be equivalent — we don't even need to check the additional condition $(s_3, s_4)$. Conversely, $s_0$ and $s_2$ are equivalent so the condition $(s_0, s_2)$ is true. This means that $s_1$ and $s_4$ can be equivalent.
- The updated implication chart is given by:

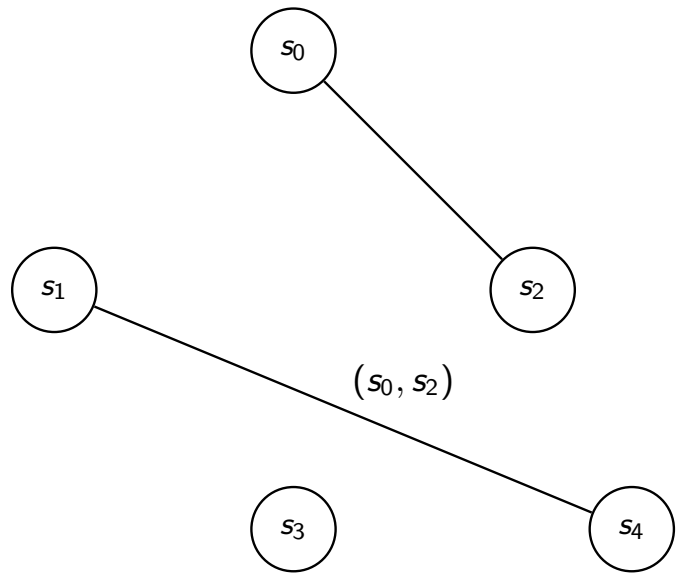


# State reduction

- Implication chart summary:
  1. Decide which states are definitely not equivalent due to differing output values;
  2. Decide which states are definitely equivalent due to having the same outputs and the same next states (always);
  3. Decide which states are equivalent subject to other states begin marked as equivalent. These become conditions for equivalence;
  4. Check the conditions to decide whether or not conditions can be met. If not, then mark the states subject to the conditions as not equivalent.

# State reduction

- Given an implication chart, we can proceed to draw a merger diagram. The purpose of the merger digram is to help us visualize which equivalent states should be merged into a single state. It also allows use to check any conditions (whether or not they are satisfied) if we merge states with conditions.
- The merger diagram for our implication chart:




- To decide what states to merge together, we look for *cliques* of states and then check any conditions to ensure they are valid.
- A clique of states is a set of states in which every state has an edge connecting it to every other state in the set. For example, a clique of 3 states $i$, $j$ and $k$ would have an edge between $i$ and $j$, $i$ and $k$ and $j$ and $k$ (a triangle).

# State reduction

- In our merger diagram there are two cliques: $s_0$ and $s_2$ as well as $s_1$ and $s_4$. However, the edge between $s_1$ and $s_4$ has a condition $(s_0, s_2)$ which means $s_1$ and $s_4$ can only be merged if out solution also merges $s_0$ and $s_2$ — in this case, the condition is satisfied.
- Therefore, we can merge $s_0$ and $s_2$ into a single state... just use $s_0$. We can also merge $s_1$ and $s_4$ can be merged into a single state... just use $s_1$.
- The original and reduced state tables:

| **Current State** | **Next State** | | **Output ($z$)** | |
|---|---|---|---|---|
| | $a=0$ | $a=1$ | $a=0$ | $a=1$ |
| $s_0$ | $s_3$ | $s_2$ | 1 | 1 |
| $s_1$ | $s_0$ | $s_4$ | 0 | 0 |
| $s_2$ | $s_3$ | $s_0$ | 1 | 1 |
| $s_3$ | $s_1$ | $s_3$ | 0 | 0 |
| $s_4$ | $s_2$ | $s_1$ | 0 | 0 |

$\rightarrow$

| **Current State** | **Next State** | | **Output ($z$)** | |
|---|---|---|---|---|
| | $a=0$ | $a=1$ | $a=0$ | $a=1$ |
| $s_0$ | $s_3$ | $s_0$ | 1 | 1 |
| $s_1$ | $s_0$ | $s_1$ | 0 | 0 |
| $s_3$ | $s_1$ | $s_3$ | 0 | 0 |

- Whatever problem the original state table was designed to solve can actually be done with 3 states and not 5 — this reduction would require less flip flops and less additional logic and is therefore likely leads to a more efficient implementation.