EX (1): Let U = {a,b,c,d,e,f,g,h,i,j }, A = {a,b,c}, B = {a,f,g}, C = {h,i,f}, Find:

1. **A U B**
2. **A ∩ C**
3. **A – B**
4. **A' U B'**
5. **(A U B) '**

EX (2): Find cardinality for:

1. **Φ**
2. **{a, b}**
3. **{1,2,3,4,5,6}**
4. **{Φ}**
5. **{{}}**

EX (3): Find power set for:

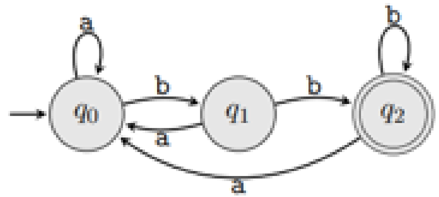
1. **{1,2}**
2. **{a,b,c}**
3. **Φ**
4. **{a, {a,b}}**

Find cardinality for each power set:

EX (4): Let A = {1,2}, B = {a,b,c}, Find:

1. **AxB**
2. **BxA**
3. **Cardinality**
4. **Let A = {1,2}, B = {2,3}, prove that: AxB ≠ BxA**

EX (5): DFA example



States:

Alphabet:

Transitions:

Start state:

Accepting states:

Test : (aabba) and (ababb) and (abbbabbb):

EX (6): Build a DFA for the following language:

L = {w | w is a binary string that contains 01 as a substring}

EX (7): Clamping Logic: A clamping circuit waits for a "1" input and turns on forever. However, to avoid clamping on spurious noise, we'll design a DFA that waits for *two consecutive 1s* in a row before clamping on.

Build a DFA for the following language:
L = { w | w is a bit string which contains the substring 11}

EX (8): Build a DFA for the following language:

L = { w | w is a binary string that has even number of 1s}

EX (9): Build a DFA for the following language:

L = { w | w is a binary string that has even number of 1s and even number of 0's}

EX (10): Build an NFA for the following language:

L = { w | w ends in 01}

EX (11): L = {w | w ends in 01} , NFA to DFA construction

EX (12): L = {w | w is a binary string s.t., the k^{th} symbol from its end is a 1}

NFA has k+1 states.

But an equivalent DFA needs to have at least 2^k states.

EX (13): L = {w | w is empty, <u>or</u> if non-empty will end in 01} An NFA

EX (14): Let E = {Q_E,∑,δ_E,q_0,F_E} be an ε-NFA

<u>Goal:</u> To build DFA D={Q_D,∑,δ_D,{q_D},F_D} s.t. L(D)=L(E)

EX (15): L = {w | w is empty, or if non-empty will end in 01} , ε-NFA ➔ DFA: