Tabular Method

Quine-McCluskey Algorithm
Petrick's Method

Language

Java

Quine-McCluskey Algorithm
Petrick's Method

논리식 최소화

Step 1. 입력 & 클래스 분류

Step 1. 입력 & 클래스 분류

# of 1s	Minterm	Binary
0	0	0 0 0 0
1	4 8	0 1 0 0 1 0 0 0
2	10 12	1 0 1 0 1 1 0 0
3	11 13	1 0 1 1 1 1 0 1
4	15	1 1 1 1

“Minterm” Class

Step 1. 입력 & 클래스 분류

```
for(int i = 0; i < mintermSize; i++)
{
    System.out.printf("Minterm [%d] > ", i + 1);
    int t = scan.nextInt();
    minterm[i] = new Minterm(t, binarySize);
    dontCareNumbers.add(i);
}

for(int i = 0; i < dcsize; i++)
{
    System.out.printf("Don't Care [%d] > ", i + 1);
    int t = scan.nextInt();
    minterm[mintermSize + i] = new Minterm(t, binarySize);
    minterm[mintermSize + i].setDontCare(true);
}
```

Step 2. Prime Implicants 찾기

Step 2. Prime Implicants 찾기

Group Minterms

1의 개수

구성한 숫자의 개수

```
@Override
public int compareTo(Minterm m) {
    int myOne = this.getNumberOfOnes();
    int otherOne = m.getNumberOfOnes();
    if(myOne < otherOne) return -1;
    else if(myOne == otherOne)
    {
        int myTermSum = this.sumOfTermsNumber();
        int otherTermSum = m.sumOfTermsNumber();
        if(myTermSum < otherTermSum) return -1;
        else if(myTermSum == otherTermSum) return 0;
        else return 1;
    }
    else return 1;

}
```

Step 2. Prime Implicants 찾기

Group Minterms

Before

# of 1s	Minterm	Binary	Combined
0	0	0000	
1	2	0010	
2	3	0011	
1	1	0001	
1	4	0100	
3	7	0111	
2	5	0101	
2	6	0110	
2	9	1001	
2	10	1010	

After

# of 1s	Minterm	Binary	Combined
0	0	0000	
1	1	0001	
1	2	0010	
1	4	0100	
2	3	0011	
2	5	0101	
2	6	0110	
2	9	1001	
2	10	1010	
3	7	0111	

Combine Minterms

```
ArrayList<Minterm> CombinedMinterm = new ArrayList<Minterm>();
HashSet<String> duplicate = new HashSet<String>();
for(int i = 0; i < max_ones; i++)
{
    Minterm[] minterms_a = Minterm.getMinterms(minterm, i);
    Minterm[] minterms_b = Minterm.getMinterms(minterm, i + 1);
    for(int ai = 0; ai < minterms_a.length; ai++)
    {
        for(int bi = 0; bi < minterms_b.length; bi++)
        {
            Minterm minterm_ai = minterms_a[ai];
            Minterm minterm_bi = minterms_b[bi];
            if(Minterm.getHammingDistance(minterm_ai, minterm_bi) == 1)
            {
                escape = true;
                Minterm newMinterm = new Minterm(minterm_ai, minterm_bi);
                Combined[Minterm.getIndexOfMintermGroups(minterm, i, ai)] = 1;
                Combined[Minterm.getIndexOfMintermGroups(minterm, i + 1, bi)] = 1;

                if(!duplicate.contains(newMinterm.toString()))
                {
                    duplicate.add(newMinterm.toString());
                    CombinedMinterm.add(newMinterm);
                }
            }
        }
    }
}
```

Hamming Distance == 1?

Combined에 표시

중복 검사

```
duplicate = new HashSet<String>();
for(int i = 0; i < minterm.length; i++)
    if(Combined[i] == 0 && !duplicate.contains(minterm[i].toString()))
    {
        duplicate.add(minterm[i].toString());

        ArrayList<Minterm> mL;
        if(notCombinedMinterm != null && notCombinedMinterm.length != 0)
            mL = new ArrayList<Minterm>(Arrays.asList(notCombinedMinterm));
        else mL = new ArrayList<Minterm>();
        mL.add(minterm[i]);
        notCombinedMinterm = mL.toArray(new Minterm[mL.size()]);
    }

Minterm.sort(notCombinedMinterm);
```

Group minterms

합치지 못한 Minterms는 별도로 분리

Step 2. Prime Implicants 찾기

Combine Minterms

Before

# of 1s	Minterm	Binary	Combined
0	0	0000	
1	1	0001	
1	2	0010	
1	4	0100	
2	3	0011	
2	5	0101	
2	6	0110	
2	9	1001	
2	10	1010	
3	7	0111	

After

# of 1s	Minterm	Binary	Combined
0	0, 1	000-	V
0	0, 2	00-0	V
0	0, 4	0-00	V
1	1, 3	00-1	V
1	1, 5	0-01	V
1	1, 9	-001	
1	2, 3	001-	V
1	2, 6	0-10	V
1	2, 10	-010	
1	4, 5	010-	V
1	4, 6	01-0	V
2	3, 7	0-11	V
2	5, 7	01-1	V
2	6, 7	011-	V

# of 1s	Minterm	Binary	Combined
0	0, 1, 2, 3, 4, 5, 6, 7	0---	

Step 3. Essential Prime Implicants 찾기

Step 3. Essential Prime Implicants 찾기

합치지 못한 Minterms

처음 입력된 Minterm

```
PITable pit = new PITable(notCombinedMinterm, origin_minterm, binarySize);
```

세로

가로

Step 3. Essential Prime Implicants 찾기

	origin_minterm							
notCombinedMinterm	00	04	08	10	11	12	13	15
P0 = --00	V	V	V			V		
P1 = 10-0			V	V				
P2 = 101-				V	V			
P3 = 110-						V	V	
P4 = 1-11					V			V
P5 = 11-1							V	V

Step 3. Essential Prime Implicants 찾기

```
public void findEPI()
{
    for(int i = 0; i < minterm.length; i++)
    {
        if(minterm[i].isDontCare()) continue; // don't care이면 무시!

        int check = 0; int last_location = -1;
        for(int j = 0; j < PI.length; j++)
            if(table[j][i] == 1)
            {
                check += 1;
                last_location = j;
            }
        if(check == 1) setEPI(last_location);
    }

    // EPI print
    System.out.print("EPI: ");
    for(int i = 0; i < EPI.length; i++)
        if(EPI[i] == 1)
            System.out.printf("P%d ", i);
    System.out.println();
}
```

Step 3. Essential Prime Implicants 찾기

Before

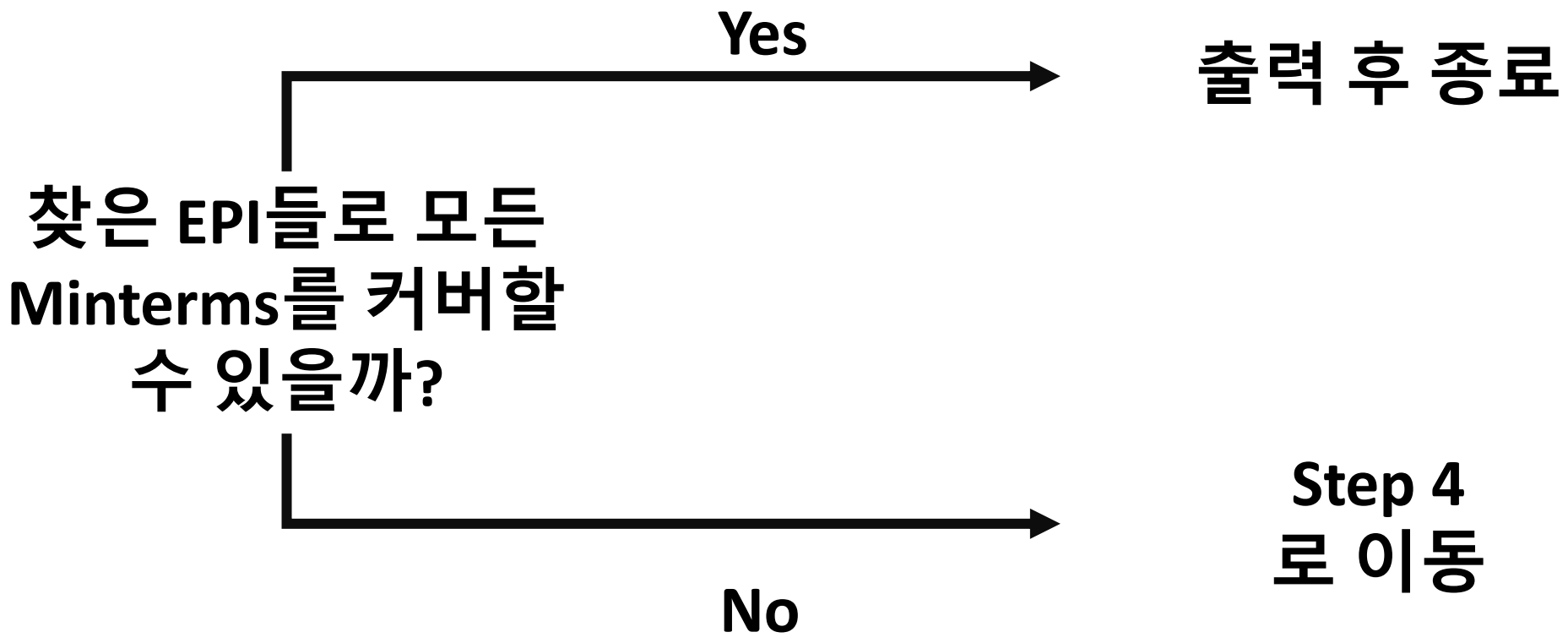
	00	04	08	10	11	12	13	15
P0 = --00	V	V	V			V		
P1 = 10-0			V	V				
P2 = 101-				V	V			
P3 = 110-						V	V	
P4 = 1-11					V			V
P5 = 11-1							V	V

After

EPI: P0

	10	11
P1 = 10-0	V	
P2 = 101-	V	V
P3 = 110-		
P4 = 1-11		V
P5 = 11-1		

Step 3. Essential Prime Implicants 찾기



Step 4. Row / Column Dominance

Step 4. Row / Column Dominance

```
                |10|11
P1 = 10-0|  V|
P2 = 101-|  V|  V
```

```java
public void columnDominace()
{
    for(int i = 0; i < minterm.length; i++)
    {
        for(int j = 0; j < minterm.length; j++)
        {
            if(i != j) // 달라야 함.
            {
                if(ignore[i] == 1 || ignore[j] == 1) continue;

                if(minterm[i].isDontCare() || minterm[j].isDontCare()) continue;

                boolean dominanced = true; // i가 j를 dominance 했는가
                for(int k = 0; k < PI.length; k++)
                {
                    if(table[k][i] == 0 && table[k][j] == 1) // 유일하게 dominance가 안되는 경우!
                    {
                        dominanced = false; break;
                    }
                }

                if(dominanced) ignore[i] = 1;
```

Column Dominance

```
for(tempPI tPIv : tPI)
{
    int i = tPIv.idx;
    for(int j = 0; j < PI.length; j++)
    {
        if(i != j)
        {
            boolean iempty = true;
            boolean jempty = true;
            boolean dominanced = true;
            for(int k = 0; k < minterm.length; k++)
            {
                if(!minterm[k].isDontCare() && ignore[k] == 0)
                {
                    if(table[i][k] == 1) iempty = false;
                    if(table[j][k] == 1) jempty = false;
                    if(table[i][k] == 0 && table[j][k] == 1)
                    {
                        dominanced = false; break;
                    }
                }
            }

            if(iempty || jempty) continue;

            if(dominanced)
            {
                System.out.printf("NEPI: P%d는 P%d를 Dominance한다.\n", i, j);
                EPI[i] = 1; EPI[j] = 0; // 쉽게 표기하기 위해 EPI라 표기 , setEPI를 쓰지 않는다.
            }
```

Row Dominance

Greedy Solution
각 Row를 기준으로 V가 적은 것부터 탐색

그렇다면, V가 많은 것부터 탐색하면 무슨 문제가 생길까?

Step 4. Row / Column Dominance

Before

	00	02	03	04	05	06	07	08	09	10	11	12	13
P0 = 0--0	V	V		V		V							
P1 = -0-0	V	V						V		V			
P2 = --00	V			V				V				V	
P3 = 0-1-		V	V			V	V						
P4 = 01--				V	V	V	V						
P5 = -01-		V	V							V	V		
P6 = -10-				V	V							V	V
P7 = 10--								V	V	V	V		
P8 = 1-0-								V	V			V	V

Column Dominance

	00	03	05	07	09	11	13
P0 = 0--0	V						
P1 = -0-0	V						
P2 = --00	V						
P3 = 0-1-		V		V			
P4 = 01--			V	V			
P5 = -01-		V				V	
P6 = -10-			V				V
P7 = 10--					V	V	
P8 = 1-0-					V		V

Step 4. Row / Column Dominance

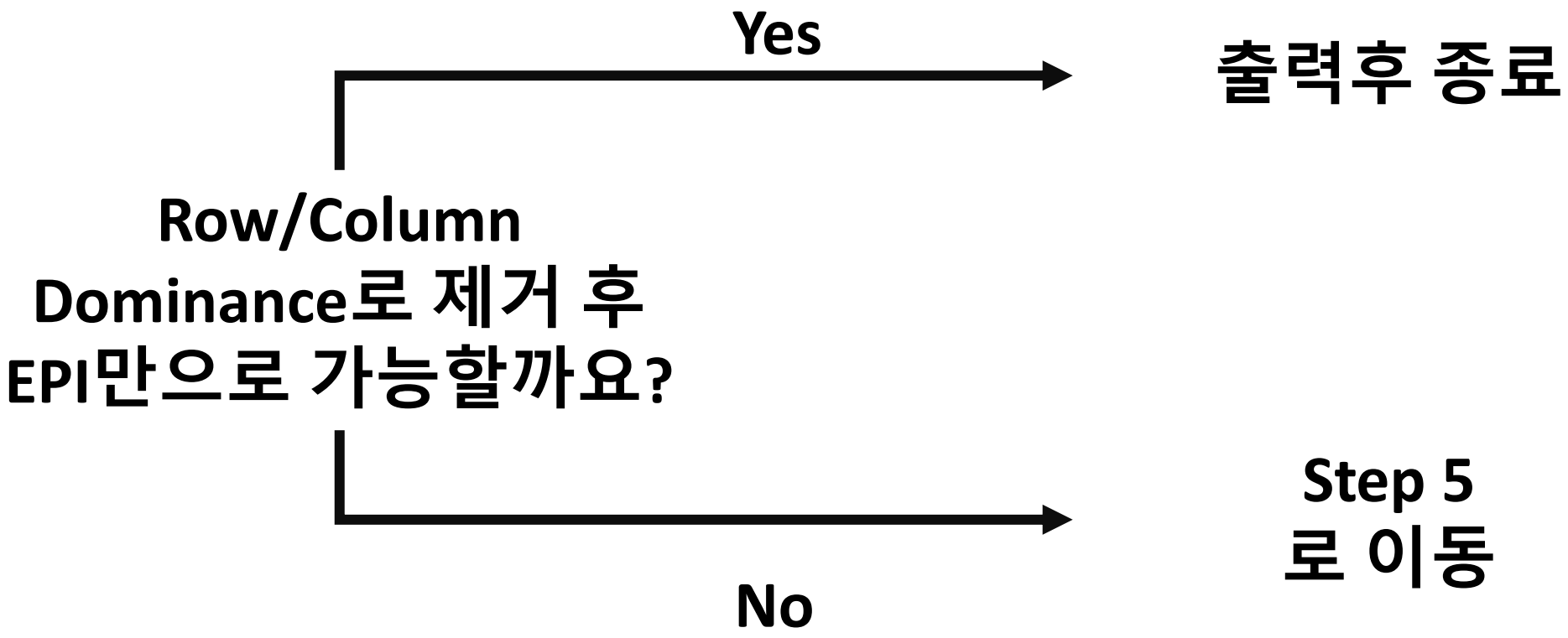
Column Dominance

	00	03	05	07	09	11	13
P0 = 0--0	V						
P1 = -0-0	V						
P2 = --00	V						
P3 = 0-1-		V		V			
P4 = 01--			V	V			
P5 = -01-		V				V	
P6 = -10-			V				V
P7 = 10--					V	V	
P8 = 1-0-					V		V

Row Dominance

	03	05	07	09	11	13
P0 = 0--0						
P1 = -0-0						
P3 = 0-1-	V		V			
P4 = 01--		V	V			
P5 = -01-	V				V	
P6 = -10-		V				V
P7 = 10--				V	V	
P8 = 1-0-				V		V

Step 4. Row / Column Dominance



Step 5. Petrick’s Method

Step 5. Petrick's Method

```
public class LogicExpression {

    ArrayList<ArrayList<Integer>> exp = new ArrayList<ArrayList<Integer>>();
```

"LogicExpression" = 하나의 SOP Class

	03	05	07	09	11	13
P0 = 0--0						
P1 = -0-0						
P3 = 0-1-	V		V			
P4 = 01--		V	V			
P5 = -01-	V				V	
P6 = -10-		V				V
P7 = 10--				V	V	
P8 = 1-0-				V		V

P3P6P7

F(with NEPIs) = (P3 + P5)(P4 + P6)(P3 + P4)(P7 + P8)(P5 + P7)(P6 + P8)

Step 5. Petrick's Method

```
Deque<LogicExpression> dq = new LinkedList<LogicExpression>();
for(int i = 0; i < minterm.length; i++)
{
    if(ignore[i] == 1) continue;

    ArrayList<Integer> numbers = new ArrayList<Integer>();
    for(int j = 0; j < PI.length; j++)
    {
        // EPI인 PI를 탐색해야 하는것인가? => 있었으면 ignore[i]는 1이었을 것이므로 탐색할 필요가 없다.
        if(EPI[j] == 1) continue;
        if(table[j][i] == 1)
            numbers.add(j);
    }

    dq.addLast(new LogicExpression(numbers));
}
```

Step 5. Petrick's Method

```
while(dq.size() > 1)
{
    LogicExpression exp1 = dq.getFirst(); dq.pop();
    LogicExpression exp2 = dq.getFirst(); dq.pop();
    dq.addFirst(LogicExpression.multiply(exp1, exp2));
    LogicExpression.show(dq);
}
```

```
F(with NEPIs) = (P3 + P5)(P4 + P6)(P3 + P4)(P7 + P8)(P5 + P7)(P6 + P8)
F(with NEPIs) = (P3P4 + P3P6 + P4P5 + P5P6)(P3 + P4)(P7 + P8)(P5 + P7)(P6 + P8)
F(with NEPIs) = (P3P6 + P4P5)(P7 + P8)(P5 + P7)(P6 + P8)
F(with NEPIs) = (P3P6P7 + P3P6P8 + P4P5P7 + P4P5P8)(P5 + P7)(P6 + P8)
F(with NEPIs) = (P3P6P7 + P4P5P8 + P3P5P6P8)(P6 + P8)
F(with NEPIs) = (P3P6P7 + P4P5P8)
```

Step 5. Petrick's Method

```
private static ArrayList<Integer> multiply(ArrayList<Integer> a, ArrayList<Integer> b)
{
    HashSet<Integer> set = new HashSet<Integer>();

    for(Integer v : a) set.add(v);
    for(Integer v : b) set.add(v);

    return new ArrayList<Integer>(set);
}
```

```
private void minimize()
{
    /* X + XY = X를 이용
     * X + X 도 내포되어 있고, X * X 는 multiply() 과정에서 제거
     */

    ArrayList<ArrayList<Integer>> removeArray = new ArrayList<ArrayList<Integer>>();

    ArrayList<Integer> expArray[] = exp.toArray(new ArrayList[exp.size()]);
    for(int i = 0; i < expArray.length; i++)
        for(int j = 0; j < expArray.length; j++)
            if(i != j && expArray[j].containsAll(expArray[i]))
                removeArray.add(expArray[j]);

    for(ArrayList<Integer> remove : removeArray)
        exp.remove(remove);

}
```

Step 5. Petrick's Method

F(with NEPIs) = (P3 + P5)(P4 + P6)(P3 + P4)(P7 + P8)(P5 + P7)(P6 + P8)
F(with NEPIs) = (P3P4 + P3P6 + P4P5 + P5P6)(P3 + P4)(P7 + P8)(P5 + P7)(P6 + P8)
F(with NEPIs) = (P3P6 + P4P5)(P7 + P8)(P5 + P7)(P6 + P8)
F(with NEPIs) = (P3P6P7 + P3P6P8 + P4P5P7 + P4P5P8)(P5 + P7)(P6 + P8)
F(with NEPIs) = (P3P6P7 + P4P5P8 + P3P5P6P8)(P6 + P8)
F(with NEPIs) = (P3P6P7 + P4P5P8)
F = P2 + P3P6P7

Thank you