Report : Row dominance, Column dominance 도전과제 구현

이전 과제에서 구해낸 PI 중에서 EPI를 제외한 PI들로 Row dominance, Column dominance를 이용하여 Minterm을 커버하는 Secondary EPI를 구하고 EPI와 Secondary EPI를 이용하여 function을 최적화하고, final solution을 구하는 것이 목적이다.

Step:

1. 모든 PI를 찾아서 Minterm들과 함께 테이블을 만든다.(구현)
2. EPI를 찾아서(구현) **EPI와 EPI가 커버하는 Minterm들을 테이블에서 지운다.**
3. **Row dominance를 적용하여 지배당하는 PI들을 테이블에서 지운다.**
4. **Column dominance를 적용하여 지배하는 Minterm들을 테이블에서 지운다.**
5. **Simplification된 테이블에서 Secondary EPI를 찾는다.**
6. **EPI와 Secondary EPI로 모든 Minterm을 커버할 수 있는지 확인한다.**

고려해야할 점:

1. Row dominance, Column dominance를 적용할 때 interchangeable 한 경우가 있는지 확인한다.
2. 위의 과정을 진행했을 때 모든 Minterm을 커버하지 못한다면 위 과정을 반복한다.

Algorithm 구현

def ComparePI(r, m)

파라미터로 PI가 들어있는 리스트 (r) 와 minterm이 들어있는 리스트 (m) 를 받아 PI와 minterm을 비교하여 각 PI로 만들 수 있는 모든 minterm의 경우의 수를 찾는 함수이다. minterm을 정해서 모든 PI와 비교하여 각 PI 내부에 있는 "-"를 minterm과 같은 값으로 바꿔가면서 비교하고, 바꾼 값이 같으면 그 minterm은 그 PI로 만들 수 있는 경우의 수

인 것으로 확인하는 방식으로 구현했다.

```
def comparePI(r, m):
    numList = []
    combineList = []
    for i in m:
        for j in range(len(r)):
            a = r[j]
            for k in range(len(r[j])):
                if r[j][k] == "-":
                    a = a.replace("-", i[k], 1)
            if (a == i):
                numList.append(a)
                combineList.append(r[j])
    return numList, combineList
```

```
numList:  ['0000', '0100', '1000', '1000', '1010', '1010', '1100', '1100', '1011', '1011', '1101', '1101', '1111', '1111']
combineList:  ['--00', '--00', '--00', '10-0', '101-', '10-0', '--00', '110-', '101-', '1-11', '11-1', '110-', '1-11', '11-1']
```

PI의 "-"가 존재하는 index를 찾고, minterm에서 그 index에 있는 값으로 바꾼 뒤 "-"를 바꾼 PI와 minterm의 값이 같으면 numList에 그 minterm을 append하고, combineList에 그 PI를 append하여 동일한 index값을 가지게 하였다.

def findEPI(resPi, mintermList)

```
def findEPI(resPi, mintermList):
    r = resPi[:]
    m = mintermList[:]
    epiList = []
    numList, combineList = comparePI(r, m)
    usedMinterm = []
    for i in numList:
        if numList.count(i) == 1:
            epiValue = combineList[numList.index(i)]
            if combineList[numList.index(i)] not in epiList:
                epiList.append(epiValue)
    for i in epiList:
        for j in range(len(combineList)):
            if i == combineList[j]:
                usedMinterm.append(numList[j])
    return epiList, usedMinterm
```

```
epiList: ['--00']
usedMinterm: ['0000', '0100', '1000', '1100']
```

EPI와 그 EPI가 커버하는 minterm들을 찾는 함수이다. comparePI 함수를 호출해 PI는 combinedList에, 그 PI가 커버하는 minterm은 numList에 동일한 index값을 가지게 해서 얻어낸다.

numList에서 minterm이 하나밖에 없다면 그 minterm을 커버하는 PI가 EPI이니 numList에서 하나밖에 존재하지 않는 minterm의 index를 찾고, combinedList에서 그 index 값에 존재하는 PI를 epiList에 넣었다.

이렇게 EPI를 구하고 그 EPI값이 combinedList에서 몇번째 index에 존재하는지 찾고, numList의 같은 index를 가지는 값을 찾아 EPI가 커버하는 minterm을 구한다.

def row_dominance(resPi, mintermList)

```
def row_dominance(resPi, mintermList):
    print("rd check...")
    n, c = comparePI(resPi, mintermList)
    rowList = [[] for j in range(len(resPi))]
    for i in range(len(resPi)):
        rowList[i].append(resPi[i])

    for i in resPi:
        for j in range(len(c)):
            if i == c[j]:
                for k in rowList:
                    if i in k:
                        k.append(n[j])
    row_eraseList = []
    for i in rowList:
        for j in range(len(rowList)):
            if i[0] != rowList[j][0]:
                row_intersection = list(set(i) & set(rowList[j]))
                if sorted(row_intersection) == sorted(i[1:]):
                    if rowList[j][0] in row_eraseList:          #interchangable 제외 위함
                        continue
                    print(i[0], "dominated by", rowList[j][0])
                    if i[0] in row_eraseList:                   #rd가 여러번 일어나는 것을 방지하기 위함
                        continue
                    row_eraseList.append(i[0])
    print("rd:", row_eraseList, "will be erased")
    return row_eraseList
```

comparePI 함수를 호출하여 PI와 PI가 커버하는 minterm들의 리스트를 받고 rowList라는 2차원배열을 만들어서 원소로 PI와 그 PI가 커버하는 minterm들을 함께 넣어 table을 만든다.

```
rowList: [['101-', '1010', '1011'], ['11-1', '1101', '1111'], ['110-', '1101'], ['1-11', '1011', '1111'], ['10-0', '1010']]
```

rowList에서 for문을 이용하여 2차원 배열 내부의 1차원 배열들끼리 비교한다. row dominance를 확인하기 위하여 파이썬 내장함수인 set 함수를 이용하여 집합으로 바꾼 뒤 &를 이용해 교집합인 row_intersection을 구하였다. 만약 비교한 1차원배열을 PI를 제외하기 위하여 첫번째 index부터 slicing하고, 그 값이 row_intersection의 값과 같다면 row dominance인 것이다. 예를 들어 ['10-0', '1010']이 i이고 ['101-', '1010', '1011']이 rowList[j][0]이면 row_intersection은 ['1010']이다. 그리고 i[1:]과 row_intersection값을 비교하면 ['1010']으로 같으니 ['10-0', '1010']과 ['101-', '1010', '1011']은 row dominance 관계이다.

구하는 과정에서 교집합의 배열 순서와 i[:1]의 배열 순서가 다르면 ==이 적용되지 않아 sorted 함수를 이용하였고, interchangable 관계 중 하나만 남기기 위해 지배하는 경우인 rowList[j][0]이 row_eraseList에 이미 존재할 때는 continue 처리를 해주었다.

def column_dominance(resPi, mintermList)

```
def column_dominance(resPi, mintermList):
    print("cd check...")
    n, c = comparePI(resPi, mintermList)
    columnList = [[] for i in range(len(mintermList))]
    for i in range(len(mintermList)):
        columnList[i].append(mintermList[i])
    for i in mintermList:
        for j in range(len(n)):
            if i == n[j]:
                for k in columnList:
                    if i in k:
                        k.append(c[j])
    column_eraseList = []
    for i in columnList:
        for j in range(len(columnList)):
            if i[0] != columnList[j][0]:
                column_intersection = list(set(i) & set(columnList[j]))
                if sortPi(column_intersection) == sortPi(i[1:]):
                    if i[0] in column_eraseList:
                        continue
                    print(i[0], "dominated by", columnList[j][0])
                    if columnList[j][0] in column_eraseList:
                        continue
                    column_eraseList.append(columnList[j][0])
    print("cd:", column_eraseList, "will be erased")
    return column_eraseList
```

```
columnList: [['1010', '101-'], ['1011', '101-', '1-11'], ['1101', '11-1'], ['1111', '11-1', '1-11']]
```

column_dominance 함수는 columnList를 구현할 때 minterm과 그 minterm을 커버하는

PI로 table을 구성한 것 이외에 로직은 row_dominance와 동일하다.

Main

```
cnt = 1
while True:
    print("case", cnt)
    epi, remove_min = findEPI(resPi, mintermList)
    answer += epi
    print("pi, minterm :", resPi, mintermList)
    for i in epi:
        if i in resPi:
            resPi.remove(i)
    for i in remove_min:
        if i in mintermList:
            mintermList.remove(i)
    print("epi, minterm :", epi, mintermList)
    dominanced_PI = row_dominance(resPi, mintermList)
    for i in dominanced_PI:
        if i in resPi:
            resPi.remove(i)
    dominanced_min = column_dominance(resPi, mintermList)
    for i in dominanced_min:
        if i in mintermList:
            mintermList.remove(i)
    if not dominanced_min and not dominanced_PI:
        break
    cnt += 1
```

```
if mintermList:
    print("Petrick's Method needed!!!")
    print("EPI and Secondary EPI:", answer)
else:
    print("All Minterm Covered!!!")
    print("EPI and Secondary EPI:", answer)
```

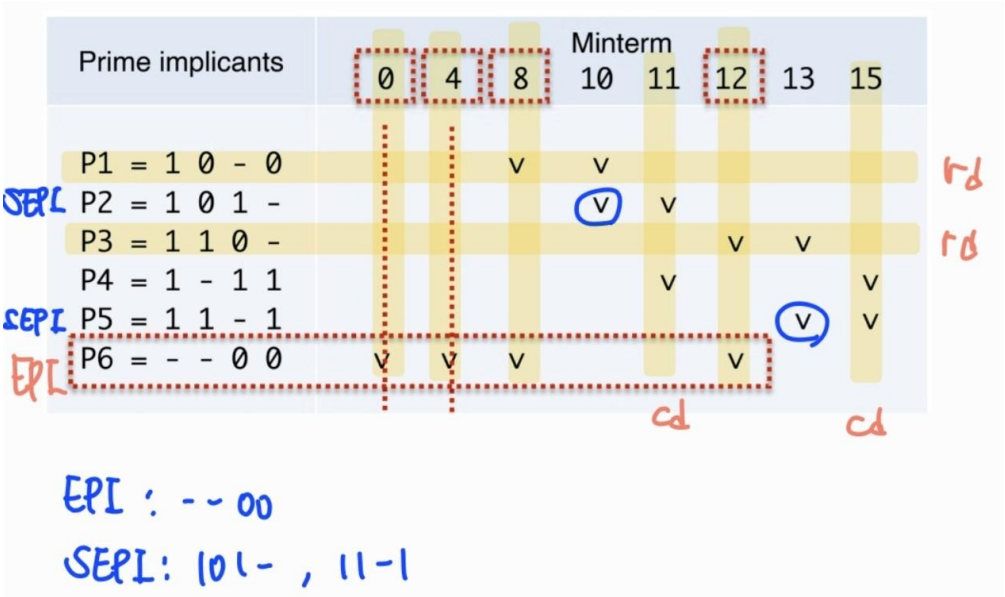
메인 함수에서는 이전에 구했던 PI의 리스트인 resPi와 minterm들이 담겨있는 mintermList를 이용하여 EPI와 EPI가 커버하는 minterm의 리스트를 구하고 epi를 answer에 더한다. resPI와 mintermList에서 이를 지우고 row_dominance 함수를 호출하여 그 리턴값인 row_eraseList를 dominanced_PI에 저장하고, 이를 다시 resPI에서 지운다. column dominance도 동일하게 호출하여 dominanced_min에 저장하고 리턴값을 mintermList에서 지운다.

이를 while문 안에 넣어서 한번 더 실행되었을 때 구해지는 epi가 secondary epi이고, 계속 진행하였을 때 더 이상 row dominance와 column dominance가 이뤄지지 않으면

break 한다. While 문이 끝났을 때의 answer값은 EPI와 Secondary EPI가 합쳐진 것이고 mintermList에 더 이상 minterm이 없다면 이는 EPI와 Secondary EPI가 모든 minterm을 커버한다는 것이다. 그래서 mintermList가 존재하면 Petrick's Method가 필요하다고 출력하였고, 존재하지않으면 최적화가 끝났다는 All Minterm Covered를 출력하였다.

실행 결과

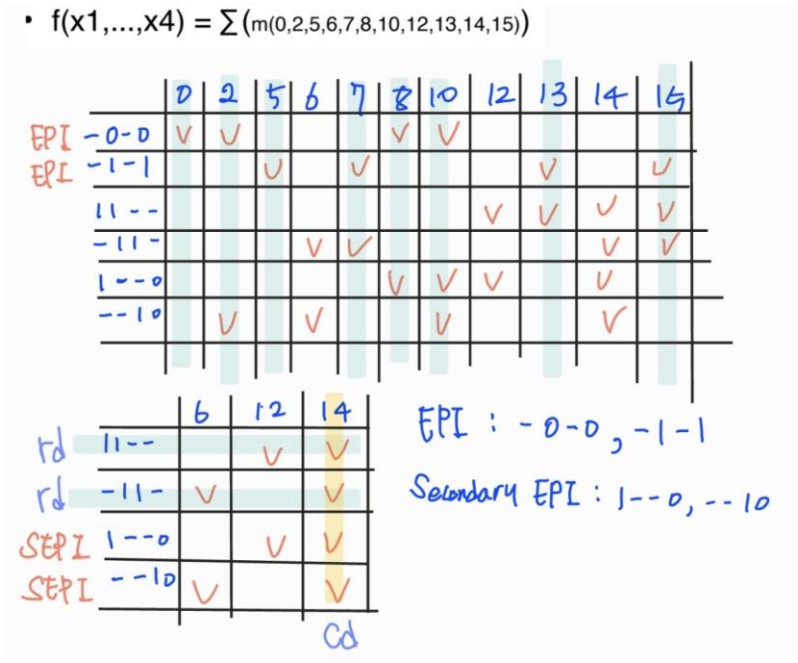
1. **입력이 [4, 8, 0, 4, 8, 10, 11, 13, 15]일 때** (결과가 모든 minterm을 커버하는 경우)




```
case 1
pi, minterm : ['11-1', '110-', '10-0', '101-', '--00', '1-11'] ['0000', '0100', '1000', '1010', '1100', '1011', '1101', '1111']
epi, minterm : ['--00'] ['1010', '1011', '1101', '1111']
rd check...
110- dominated by 11-1
10-0 dominated by 101-
rd: ['110-', '10-0'] will be erased
cd check...
1010 dominated by 1011
1101 dominated by 1111
cd: ['1011', '1111'] will be erased
case 2
pi, minterm : ['11-1', '101-', '1-11'] ['1010', '1101']
epi, minterm : ['101-', '11-1'] []
rd check...
rd: [] will be erased
cd check...
cd: [] will be erased
All Minterm Covered!!!
EPI and Secondary EPI: ['--00', '101-', '11-1']
finished
```

2. 입력이 [4, 11, 0, 2, 5, 6, 7, 8, 10, 12, 13, 14, 15]일 때

(결과가 모든 minterm을 커버하면서 interchangable이 존재하는 경우)

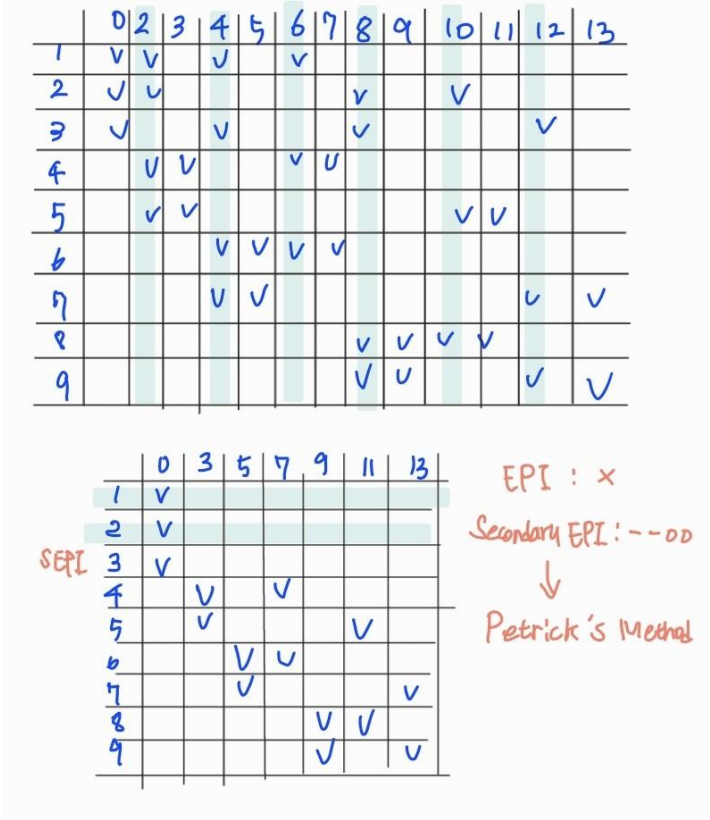



```
case 1
pi, minterm : ['--10', '1--0', '-11-', '-1-1', '-0-0', '11--'] ['0000', '0010', '1000', '0101', '0110', '1010', '1100', '0111', '1101', '1110', '1111']
epi, minterm : ['-0-0', '-1-1'] ['0110', '1100', '1110']
rd check...
--10 dominated by -11-
1--0 dominated by 11--
rd: ['--10', '1--0'] will be erased
cd check...
0110 dominated by 1110
1100 dominated by 1110
cd: ['1110'] will be erased
case 2
pi, minterm : ['-11-', '11--'] ['0110', '1100']
epi, minterm : ['-11-', '11--'] []
rd check...
rd: [] will be erased
cd check...
cd: [] will be erased
All Minterm Covered!!!
EPI and Secondary EPI: ['-0-0', '-1-1', '-11-', '11--']
finished
```

interchangable이 존재함에도 하나만 잘 골라내는 것을 볼 수 있다.

3. 입력이 [4, 13, 0, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13] 일 때

(결과가 모든 minterm을 커버하지 못하면서 interchangable도 존재하는 경우)




```
case 1
pi, minterm : ['-0-0', '0-1-', '-01-', '0--0', '-10-', '01--', '--00', '1-0-', '10--'] ['0000', '0010', '0100', '1000', '0011', '0101', '0110', '1001', '1010', '1100', '0111', '1011', '1101']
epi, minterm : [] ['0000', '0010', '0100', '1000', '0011', '0101', '0110', '1001', '1010', '1100', '0111', '1011', '1101']
rd check...
rd: [] will be erased
cd check...
0011 dominated by 0010
0101 dominated by 0100
1001 dominated by 1000
0111 dominated by 0110
1011 dominated by 1010
1101 dominated by 1100
cd: ['0010', '0100', '1000', '0110', '1010', '1100'] will be erased
```

```
case 2
pi, minterm : ['-0-0', '0-1-', '-01-', '0--0', '-10-', '01--', '--00', '1-0-', '10--'] ['0000', '0011', '0101', '1001', '0111', '1011', '1101']
epi, minterm : [] ['0000', '0011', '0101', '1001', '0111', '1011', '1101']
rd check...
-0-0 dominated by 0--0
-0-0 dominated by --00
0--0 dominated by --00
rd: ['-0-0', '0--0'] will be erased
cd check...
cd: [] will be erased
case 3
pi, minterm : ['0-1-', '-01-', '-10-', '01--', '--00', '1-0-', '10--'] ['0000', '0011', '0101', '1001', '0111', '1011', '1101']
epi, minterm : ['--00'] ['0011', '0101', '1001', '0111', '1011', '1101']
rd check...
rd: [] will be erased
cd check...
cd: [] will be erased
Petrick's Method needed!!!
EPI and Secondary EPI: ['--00']
finished
```

한계

Petrick's Method를 구현하지 않았기 때문에 3번 예제같이 row dominance와 column dominance가 적용되지 않는 경우에는 유의미한 최적화 결과를 도출하기 어렵다.