논리회로설계 도전과제 결과보고서

●구현한 도전과제: Row Dominance, Column Dominance

●사용한 언어: Python

●전체 코드 기본 구조

1. PI를 탐색한다.

2. EPI를 모두 탐색한다. 다음 3단계의 과정을 while문을 통해 계속해서 반복시킨다.

2-1. EPI를 탐색한다. 탐색한 EPI와 해당 EPI를 이루는 minterm은 테이블에서 제거하여 테이블을 축소시킨다.

2-2. Column Dominance와 Row Dominance를 확인한다. Dominance가 존재하면 해당하는 특정 열이나 행을 삭제하여 테이블을 축소시킨다.

2-3. Column Dominance와 Row Dominance에 해당하는 것이 없다면, while문을 종료하고 EPI를 반환한다. 그렇지 않다면 2-1로 돌아가서 계속한다.

```
129  #find epi
130  def findEpi(piConsist, mintermConsist):
131      epi = list()
132      count = 1
133      while(True):
134          print(count,"회차----------------------------------------------------------------------------------")
135          #find EPI
136          deletedEPI = [] #삭제된 EPI
137          deletedMin = [] #삭제된 minterm
138          for k in sorted(piConsist.keys()):
139              for v in piConsist[k]:
140                  if(len(mintermConsist[v]) == 1 and k.replace('2','-') not in epi):
141                      print("EPI: " + k)
142                      epi.append(k.replace('2','-'))# 어떤 pi를 이루는 minterm이 한번만 사용되었다면, 그 pi는 epi임
143                      deletedEPI.append(k)
144                      for m in piConsist[k]: deletedMin.append(m)
145                      del piConsist[k] #epi는 제거
146          #deletedEPI와 deletedMin을 사용하여 테이블 축소
147          for dm in deletedMin:
148              for k in piConsist.keys():
149                  if dm in piConsist[k]: piConsist[k].remove(dm)
150          tempDic = copy.deepcopy(piConsist)
151          for m in tempDic.keys():
152              if len(tempDic[m]) == 0: del piConsist[m]
153          tempDic = copy.deepcopy(mintermConsist)
154          for m in tempDic.keys():
155              for e in deletedEPI:
156                  if e in tempDic[m] and m in mintermConsist.keys(): del mintermConsist[m]
157
158          print("Row after finding EPI:",piConsist)
159          print("Column after finding EPI",mintermConsist)
```

위의 코드는 2-1과정을 구현한 부분이다. EPI를 찾고 이를 사용하여 테이블을 축소시킨다

```
161         #Find Column Dominance
162         deletedColumn = columnDominance(mintermConsist, [])
163         print("deleted column:", deletedColumn)
164         for dc in deletedColumn: # 삭제된 column을 사용하여 테이블 축소
165             for k in piConsist.keys():
166                 if dc in piConsist[k]: piConsist[k].remove(dc)
167         tempDic = copy.deepcopy(piConsist)
168         for m in tempDic.keys():
169             if len(tempDic[m]) == 0: del piConsist[m]
170
171         #Find Row Dominance
172         deletedRow = rowDominance(piConsist, [])
173         print("deleted row:", deletedRow)
174         for dr in deletedRow: # 삭제된 row를 사용하여 테이블 축소
175             for k in mintermConsist.keys():
176                 if dr in mintermConsist[k]: mintermConsist[k].remove(dr)
177         tempDic = copy.deepcopy(mintermConsist)
178         for m in tempDic.keys():
179             if len(tempDic[m]) == 0: del mintermConsist[m]
180
181         print("Row after dominance check",piConsist)
182         print("Column after dominance check",mintermConsist)
183         count+=1
```

2-2과정을 구현한 부분이다. CD와 RD를 찾고 테이블을 축소시킨다. 주요 변수와 원리에 관한 설명은 밑에서 할 것이다.

```
184         if(len(deletedColumn)+len(deletedRow) == 0): break
185
186     return epi
```

2-3 과정을 구현한 부분이다. CD와 RD로 삭제된 행과 열이 없다면 while문을 종료한다. 찾은 epi를 반환한다. 이는 찾아낸 epi들이 모인 list형식이다.

●Row Dominance & Column Dominance 구현 원리

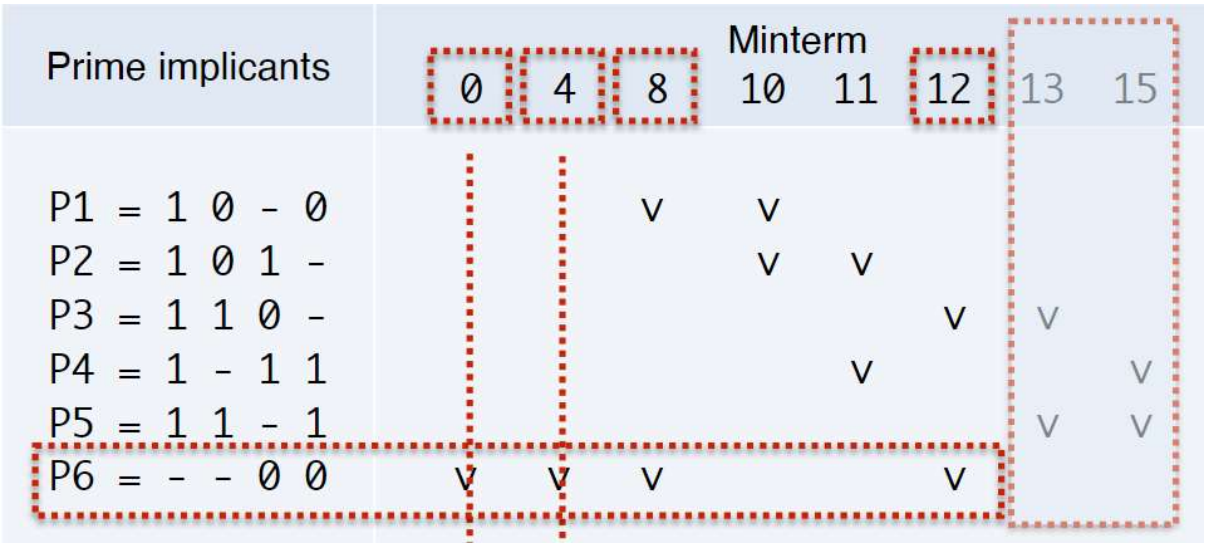
EPI를 찾고 RD와 CD를 구현하기 위해 필요한 주요 변수는 다음과 같다. 이는 dictionary형식이다.

piConsist: pi를 이루는 minterm을 기록한다. PI를 모두 찾고 난 후 만들어지는 테이블에서 '행'에 해당한다. 기본 형식은 {PI1: [minterm1, minterm2, ...], PI2: [minterm1, minterm2, ...], ...} 이다. 예를 들어 piConsist = {'002': [0,1]}이라면 PI 002는 minterm 0과 1을 합친 결과라는 것을 의미한다.

mintermConsist: 각각의 minterm이 사용되는 pi를 기록한다. PI를 모두 찾고 난 후 만들어지는 테이블에서 '열'에 해당한다. 기본 형식은 {minterm1: [PI1, PI2, ...], minterm2: [PI1, PI2, ...], ...} 이다. 예를 들어 mintermConsist = {8: ['1020', '2200']}이라면 minterm 8은 pi 1020과 2200을 만드는 데 사용됐다는 것을 의미한다.

○참고: PI에서 2는 minterm을 합칠 때 0과 1이 만나서 나오는 '-'에 해당한다. 추후 정렬을 위해 일단 2로 표기하여 저장한다.

예를 들어 강의자료의 다음과 같은 테이블을 piConsist와 mintermConsist를 사용해서 표현한다면 다음과 같을 것이다.



piConsist = {'1020': [8, 10], '1012': [10, 11], '1102': [12, 13], '1211': [11, 15], '1121': [13, 15], '2200': [0, 4, 8, 12]}

mintermConsist = {0: ['2200'], 4: ['2200'], 8: ['1020', '2200'], 10: ['1020', '1012'], 11: ['1012', '1211'], 12: ['1102', '2200'], 13: ['1102', '1121'], 15: ['1211', '1121']}

따라서 테이블을 축소시킨다는 것은, piConsist와 mintermConsist에서 특정 값(PI 또는 minterm)을 삭제한다는 것을 의미한다.

1. EPI 찾기

mintermConsist에서 어떤 minterm이 PI를 만드는데 한 번 밖에 사용되지 않았다면, 이 PI는 EPI이다. 즉, len(mintermConsist[minterm]) == 1이라면 mintermConsist[minterm]의 리스트에 있는 단 하나의 PI가 EPI이다. 이를 통해 EPI를 판정하고 찾아낸 epi를 epi리스트에 넣는다.

2. Column Dominance(CD) 찾기

다음은 Column Dominance를 구현한 함수이다. deletedColumn을 반환하는데, 이는 삭제된 컬럼이 모인 리스트이다. 한편 mintermConsist와 deletedColumn을 인자로 받는데, 이에서 알 수 있듯 재귀함수이다.

```
91  #ColumnDominance 구현, return값: 삭제된 column이 모인 배열
92  def columnDominance(mintermConsist, deletedColumn):
93      tempMC = copy.deepcopy(mintermConsist)
94      mintermConsistKeys = list(mintermConsist.keys())
95      for i in range(len(mintermConsist)-1):
96          for j in range(i+1, len(mintermConsist)):
97              l = tempMC[mintermConsistKeys[i]]
98              l2 = tempMC[mintermConsistKeys[i+1]]
99              tempSet = set(l+l2) #합친 후 중복 제거
100             if(len(l) == len(tempSet) or len(l2) == len(tempSet)): # 
101                 if(len(l) >= len(l2)):
102                     deletedColumn.append(mintermConsistKeys[i])
103                     del mintermConsist[mintermConsistKeys[i]]
104                 else:
105                     deletedColumn.append(mintermConsistKeys[j])
106                     del mintermConsist[mintermConsistKeys[j]]
107                 return columnDominance(mintermConsist, deletedColumn)
108     return deletedColumn
```

2중 for문을 통해 mintermConsist에서 가능한 모든 2가지 minterm간의 조합을 검사한다. 이 코드에서 주목해야할 점은 99번과 100번 라인이다.

예를 들어, 현재 minterm 0과 8을 검사하는데 mintermConsist에서 0: ['1020', '2200']이고 8: ['1020', '2200', '2220']이라고 하자. 그렇다면 97,98번 라인에서 l은 ['1020', '2200']이 되고 l2는 ['1020', '2200', '2220']이 된다. 먼저 이 두개의 리스트를 합친다. 그리고 set()을 활용하여 집합으로 만든다. 이를 통해 중복되는 값을 제거한다. 그 결과 99번 라인에서 tempSet = {'1020', '2200', '2220'}이 될 것이다. 그 후 100번 라인에서 tempSet의 크기와 l, l2의 크기를 비교한다. 이 때 tempSet이 두개의 리스트 중 어느 하나와 크기가 같다면, CD가 존재한다는 뜻이 된다. 여기서는 tempSet과 l2의 크기가 같다. 만약 CD가 존재하지 않는다면 tempSet의 크기는 l, l2의 크기보다 클 수밖에 없다.

이렇게 해서 CD를 판정했다면, 크기가 더 큰 리스트를 가지는 minterm을 지운다. 앞선 예에서는 l2의 크기가 l보다 더 크므로 minterm 8이 mintermConsist의 키 값에서 제거되고 deletedColumn에 8이 추가된다.

이 때 또다른 CD가 아직 존재할 수 있으므로 107번째 라인을 통해 재귀함수로 구현한다. 이렇게 모든 판정이 끝나면 삭제된 minterm들이 모두 모인 리스트 deletedColumn이 반환된다. 이를 통해 테이블을 축소시킨다.

3. Row Dominance(RD) 찾기

```
#RowDominance 구현, return값: 삭제된 row가 모인 배열
def rowDominance(piConsist, deletedRow):
    tempPC = copy.deepcopy(piConsist)
    piConsistKeys = list(piConsist.keys())
    for i in range(len(piConsist)-1):
        for j in range(i+1, len(piConsist)):
            l = tempPC[piConsistKeys[i]]
            l2 = tempPC[piConsistKeys[j]]
            tempSet = set(l+l2)
            if(len(l) == len(tempSet) or len(l2) == len(tempSet)):
                if(len(l) >= len(l2)):
                    deletedRow.append(piConsistKeys[j])
                    del piConsist[piConsistKeys[j]]
                else:
                    deletedRow.append(piConsistKeys[i])
                    del piConsist[piConsistKeys[i]]
                return rowDominance(piConsist, deletedRow)
    return deletedRow
```

앞서 설명한 ColumnDominance를 찾는 과정과 원리는 같다. 다른 점 하나는 CD에서는 크기가 더 큰 리스트를 가지는 minterm을 지웠다면, RD에서는 크기가 더 작은 리스트를 가지는 PI를 지운다는 것이다. CD를 구현한 코드에서 102~106라인과 위 코드를 비교하면 i와 j의 위치가 반대가 되었음을 확인할 수 있을 것이다.

만약 함수 columnDominance가 반환한 리스트와 함수 rowDominance가 반환한 리스트의 크기가 모두 0이라면 더 이상 Dominance가 존재하지 않는다는 뜻이므로 while문을 빠져나간다. 그리고 찾은 epi들이 모두 모인 리스트를 반환한다.