DLD 도전과제 결과 보고서

구현 : CD + RD
사용 언어 : python

1. class Implicant, pi_chart

먼저 도전과제 구현 내용을 풀이하기에 앞서 전체적으로 어떤 방법으로 pi를 찾고 epi를 찾는지 설명 드릴 필요가 있을 것 같습니다.

저는 조금 효율성이 떨어지더라도 로직을 직관적으로 따라가고자 implicant 클래스를 만들어서 객체 하나하나에 커버하고 있는 민텀이 무엇인지 terms라는 멤버변수에 튜플로 나타내었고, 해당 임플리컨트를 출력할 때 어떤 바이너리 스트링이 되는지를 bstr이라는 변수에 저장했습니다. 이 외에 checked 나 onecnt 같은 경우는 도전과제 구현에는 필수적이지 않지만 finding PI 과제를 구현하면서 사용한 변수입니다.

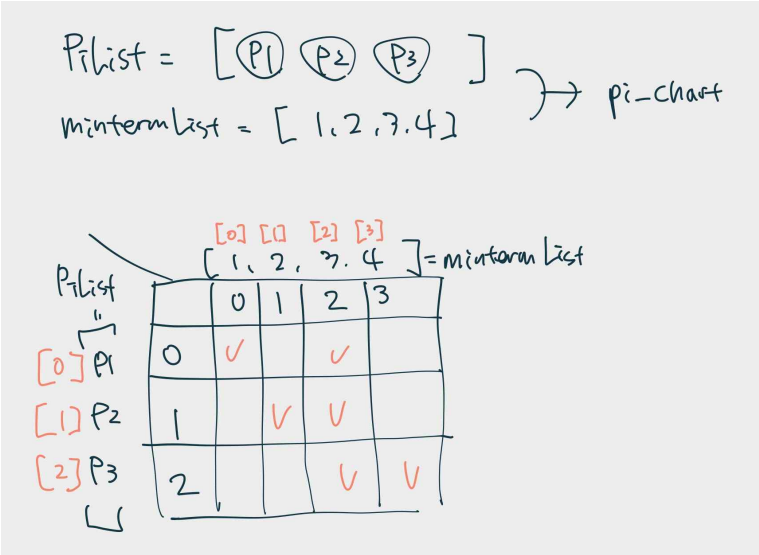
```
class Implicant:
    def __init__(self, terms, bstr, checked=False):
        self.terms = terms # 튜플
        self.bstr = bstr
        self.checked = checked
        self.onecnt=0 # 1의 개수
        for char in bstr:
            if(char=='1'):
                self.onecnt+=1

    def getterms(self):
        return self.terms
    def getbstr(self):
        return self.bstr
    def isChecked(self): # if checked == False -> PI
        return self.checked
```

findPI를 수행하면 PiList를 반환하는데 이는 Pi 객체가 담긴 리스트입니다.
findPI와 findEPI를 통해 일차적으로 epi를 찾아 커버합니다.
epi를 찾으면 해당 epi가 커버하고있는 minterm들을 더 이상 고려할 필요가 없기 때문에 mintermList에서 해당 minterm들을 제해주고, 찾은 epi도 PiList에서 제거합니다.

pi_chart는 2차원 배열로,
PiList와 mintermList 두 리스트의 각각의 인덱스가 pi_chart의 행, 열의 인덱스가 되어서 pi마다 커버하는 minterm에 True, 아닌 곳은 false를 표시한 2차원 배열의 형태를 가지게 됩니다.

```python
def makePiChart(PiList, mintermList):
    pi_chart=[]
    for pi_idx, pi in enumerate(PiList):
        pi_chart.append([])
        for i in range(len(mintermList)):
            pi_chart[pi_idx].append(False)
    for mt_idx, mt in enumerate(mintermList):
        for pi_idx, pi in enumerate(PiList):
            if mt in pi.getterms():
                pi_chart[pi_idx][mt_idx]= True
    return pi_chart
```



이해를 돕기위한 이미지입니다.

findPI, findEPI 등의 과정을 수행할 때 마다 PiList, mintermList의 내용이 달라지므로 pi_chart를 만드는 행위를 여러 번 거쳐야하기 때문에 함수로 구현하기로 결정했습니다.

2. 도전과제 구현 내용

처음 epi를 찾아 커버한 뒤 seconderyEPI를 찾는 과정을 반복합니다.
반복을 시작하기에 앞서, 만약 처음 EPI를 찾는 과정을 마쳤을 때 모든 minterm이 커버 되었다면 seconderyEPI를 찾을 필요가 없으므로 while문의 조건에 mintermList의길이가 0 이상인지 확인합니다. (mintermList의 길이가 0이라면 fully covered된 상태)

그 이후, 순서대로 eliminateDominatingColumns를 실행하여 ColumnDominance(CD), eliminateDominatedRows를 실행하여 RowDominance(RD)를 수행합니다.

```python
def solution(minterm):
    answer=[]
    PiList=[]
    mintermList = minterm[2:]
    n=minterm[0] # 변수개수

    # finding PI
    pi_str_list, PiList, mintermList = findPI(n, PiList, mintermList)
    answer+=pi_str_list

    # finding EPI
    answer.append("EPI")
    epi_str_list, PiList, mintermList = findEPI(PiList, mintermList)
    answer+=epi_str_list

    answer.append("secondery EPI")
    while len(mintermList)>0 :
        # eliminateDominatingColumns
        PiList, mintermList = eliminateDominatingColumns(PiList, mintermList)
        # eliminateDominatedRows
        PiList, mintermList = eliminateDominatedRows(PiList, mintermList)
        # find secondery EPI
        secondery_epi_str_list=[]
        secondery_epi_str_list, PiList, mintermList = findEPI(PiList, mintermList)

        if len(secondery_epi_str_list)>0: # secondery_epi 발생
            secondery_epi_str_list
            answer+=secondery_epi_str_list
        else : # secondery_epi가 생기지 않으면 chooseInterchangable
            secondery_epi_str_list, PiList, mintermList = chooseInterchangeable(PiList, mintermList)
            secondery_epi_str_list
            answer+=secondery_epi_str_list
```

3. CD

먼저 pi_chart와 딕셔너리를 만들어서 minterm, 즉 컬럼마다 True가 몇 개인지 개수를 셉니다. dic의 key는 mintermList에서 해당 minterm(컬럼)의 인덱스, value는 True의 개수입니다.

그런 뒤 이중 반복문을 통해 모든 minterm들중 해당 민텀보다 True의 개수가 적은, 즉 해당 minterm이 dominating 할 가능성이 있는 다른 minterm과 isDominating이라는 내장함수를 통해 dominating하는지의 여부를 검사합니다. 만약 해당 minterm이 dominating한다면 dominatingMtList에 해당 minterm을 삽입합니다.

모든 dominating하는 컬럼을 찾았을 때, dominatingMtList를 반복문을 통해 돌아가면서 mintermList에서 해당 minterm을 삭제합니다. 이는 pi_chart에서 해당 컬럼을 지우는것과 같습니다.

```
113    def eliminateDominatingColumns(PiList, mintermList):
114        # makePiChart
115        pi_chart = makePiChart(PiList, mintermList)
116
117        dic = {}
118        # 인덱스마다 체크 몇개인지 세기, key=mt '인덱스' in mintermList, value=체크개수
119        # make dic
120        for mt_idx, mt in enumerate(mintermList):
121            for i in range(len(PiList)):
122                if pi_chart[i][mt_idx] == True:
123                    if dic.get(mt_idx) == None:
124                        dic[mt_idx] =1
125                    else:
126                        dic[mt_idx] += 1
127
128        def isDominating(mt, next_mt):
129            for i in range(len(PiList)):
130                if pi_chart[i][next_mt] == True:
131                    if pi_chart[i][mt] == False:
132                        return False
133            return True
134
135        dominatingMtList=[]
136        # 돌아가면서 나보다 체크 적은것만 비교해서 내가 dominating 하는지 여부 확인 == make dominatingMtList
137        # mt는 mintermList의 '인덱스'
138        for mt, cnt in dic.items():
139            for next_mt, next_cnt in dic.items():
140                if(mintermList[mt]==mintermList[next_mt]):
141                    continue
142                if(cnt>next_cnt):
143                    if isDominating(mt, next_mt):
144                        dominatingMtList.append(mintermList[mt])
145        # mintermList 에서 dominating하는 mt 지우기 == eliminateDominatingColumns
146        for mt in dominatingMtList:
147            if mt in mintermList:
148                mintermList.remove(mt)
149        return PiList, mintermList
```

4. RD

이중 반복문을 통해 모든 Pi를 서로 비교하는데, 해당 pi보다 terms의 사이즈가 큰, 즉 해당 pi가 dominate당할 가능성이 있는 다른 pi와 isDominated라는 내장 함수를 통해 dominate 당하는지의 여부를 검사합니다. 만약 해당 pi가 dominated한다면 dominatedPiList에 해당 pi를 삽입합니다.

모든 dominated 당하는 행, 즉 pi를 찾았을 때, 반복문을 통해 돌아가면서 PiList에서 해당 pi들을 삭제합니다. 이는 pi_chart에서 해당 로우를 지우는 것과 같습니다.

```
152  def eliminateDominatedRows(PiList, mintermList):
153      def isDominated(pi, next_pi):
154          for mt in pi.getterms():
155              if not mt in next_pi.getterms():
156                  return False
157          return True
158
159      dominatedPiList=[]
160      # 돌아가면서 나보다 체크 많은것만 비교해서 내가 dominated 하는지 여부 확인 == make dominatedPiList
161      for pi_idx, pi in enumerate(PiList):
162          for next_pi_idx, next_pi in enumerate(PiList):
163              if (pi==next_pi):
164                  continue
165              if (len(pi.getterms()) < len(next_pi.getterms())):
166                  if isDominated(pi, next_pi):
167                      dominatedPiList.append(pi)
168      # PiList 에서 dominated 한 pi 지우기 == eliminateDominatedRows
169      for pi in dominatedPiList:
170          if pi in PiList:
171              PiList.remove(pi)
172      return PiList, mintermList
```

5. interchangeable

cd, rd를 수행한뒤 secondery epi가 발생하지 않는다면 interchangeable이 발생한 경우일 수 있습니다.

chooseInterchangeable 함수를 통해 interchangealbe중에서 임의로 한 pi를 secondery epi로 선택합니다.

PiList[0]을 임의로 선택한 뒤 선택한 pi가 커버하는 minterm을 같이 커버하는 다른 pi도 PiList에서 삭제합니다.
임의로 선택한 pi의 bstr을 반환하는데, 반환 형태는 findEPI와 같은 형태로 선택한 pi를 반환해주기 위해서 [srt]형태로 반환합니다.

```
174    def chooseInterchangeable(PiList, mintermList):
175        # PiList[0]을 임의로 결정
176        choosedInterchangeable = PiList[0]
177        str = choosedInterchangeable.getbstr()
178        # fix PiList, mintermList
179        covered_list=[]
180        for mt in choosedInterchangeable.getterms():
181            if mt in mintermList:
182                # choosedInterchangeable이 커버하는 minterm을 같이 커버하는 다른 pi도 PiList에서 삭제
183                for pi in PiList:
184                    if mt in pi.getterms():
185                        if not pi in covered_list:
186                            covered_list.append(pi)
187                # 해당 minterm 삭제
188                mintermList.remove(mt)
189        for pi in covered_list:
190            PiList.remove(pi) # pi에는 choosedInterchangeable도 포함되어있음
191        return [str], PiList, mintermList
192
```

6. 출력 방식과 테스트케이스 및 결과

```
        # print : 2->'-'
        for idx, str in enumerate(answer):
            str = list(str)
            for i in range(len(str)):
                if str[i]=='2':
                    str[i]='-'
            answer[idx]=''.join(str)
        return answer


    # minterm = [4,11 ,0,2,5,6,7,8,10,12,13,14,15]
    # minterm = [4,13 ,0,2,3,4,5,6,7,8,9,10,11,12,13]
    minterm = [4,6, 2,3,7,9,11,13]
    print(solution(minterm))
```

출력 방식은 PI, EPI 과제를 출력하는 방식을 참고하여 다음과 같은 출력 방식을 취합니다.
[‘pi’, … , “EPI”, ‘epi’, … ,“secondery EPI”, ‘secondery epi’, …]
먼저 pi들을 나열한 뒤 최초 epi를 출력하기 전 “EPI”를 출력합니다. (만약 찾은 epi가 없더라도 출력합니다.) 그 다음 CD, RD를 통해 찾은 secondery epi를 출력하기 전 “secondery EPI”를 출력합니다.

implicant 객체는 bstr 변수에 ‘0200’, ‘1220’ 등과 같이 표현되어있으므로 answer 리스트를 출력하기 전, ‘2’ 문자를 ‘-’로 바꾸어 ‘0-00’, ‘1—0’과 같은 형태로 바꾸어줍니다.

총 3 가지 테스트 케이스로 구현한 CD, RD가 잘 작동하는지 확인해보겠습니다.

1) minterm = [4,11 ,0,2,5,6,7,8,10,12,13,14,15]
결과 :
['11--', '1--0', '-0-0', '-11-', '-1-1', '--10', 'EPI', '-0-0', '-1-1', 'secondery EPI', '--10', '1—0']

2) minterm = [4,13 ,0,2,3,4,5,6,7,8,9,10,11,12,13]
결과 :
['01--', '0-1-', '0--0', '10--', '1-0-', '-01-', '-0-0', '-10-', '--00', 'EPI', 'secondery EPI', '0--0', '0-1-', '10--', '-10-']

3) minterm = [4,6, 2,3,7,9,11,13]
결과 :
['001-', '0-11', '10-1', '1-01', '-011', 'EPI', '001-', '0-11', '1-01', 'secondery EPI', '-011']