

Tabular method

사용 언어



Python



PyQt

기능

값을 입력해주세요

값은 ,로 구분됩니다

input size :

4

minterm : 0,2,5,6,7,8,10,12,13,14,15

don' care :

입력

PI TABLE

	# of 1s	Minterm	Binary	Combined	
1	0	[0]	0000	v	
2	1	[2]	0010	v	
3	1	[8]	1000	v	
4	2	[5]	0101	v	
5	2	[6]	0110	v	
6	2	[10]	1010	v	
7	2	[12]	1100	v	
8	3	[7]	0111	v	
9	3	[13]	1101	v	
10	3	[14]	1110	v	
11	4	[15]	1111	v	

->

	# of 1s	Minterm	Binary	Combined	
1	0	[0, 2]	00-0		
2	0	[0, 8]	-000		
3	1	[2, 6]	0-10		
4	1	[2, 10]	-010		
5	1	[8, 10]	10-0		
6	1	[8, 12]	1-00		
7	2	[5, 7]	01-1		
8	2	[5, 13]	-101		
9	2	[6, 7]	011-		
10	2	[6, 14]	-110		
11	2	[10, 14]	1-10		
12	2	[12, 13]	110-		
13	2	[12, 14]	11-0		

next

다음 버튼을 누르면 어떤 것을 할 지

1. EPI 찾기
2. Column dominance
3. Row dominance

를 반복

EPI 찾기

이전 상태

		0	2	5	6	7	8	10	12	13
1	P1 = -0-0	v	v				v	v		
2	P2 = --10		v		v			v		
3	P3 = 1--0						v	v	v	
4	P4 = -1-1			v		v				v
5	P5 = -11-				v	v				
6	P6 = 11--								v	v
7										

minterm

EPI TABLE

바뀐 상태

EPI List

찾은 EPI가 보임

다음

EPI TABLE

column dominance 찾기

	1	2	4	5	7	10	11	12	13	14
1 P1 = 00-10-				v				v	v	
2 P2 = 0-010-				v						
3 P3 = 00-010		v				v				
4 P4 = 0-0010		v								
5 P5 = 000-01	v			v						
6 P6 = 0001-1				v	v					
7 P7 = 001-10						v				v

EPI는 노란줄로 표시

	1	2	4	5	7	10	11	12	13	14
1 P1 = 00-10-				v				v	v	
2 P2 = 0-010-				v						
3 P3 = 00-010		v				v				
4 P4 = 0-0010		v								
5 P5 = 000-01	v			v						
6 P6 = 0001-1				v	v					
7 P7 = 001-10						v				v

EPI List

1	P5 = [1, 5] 000-0
2	P6 = [5, 7] 0001-1
3	P1 = [4, 5, 12, 13] 00...
4	P4 = [2, 18] 0-0010
5	P2 = [4, 5, 20, 21] 0-...

Don't care는 초록색으로 표시

R&C dominance에서 무조건
커버되는 PI, minterm의 경우
보라색으로 표시

다음

모두 커버된 경우, Finished 라고 쓰고,
아직 커버되지 않은 minterm이 있는 경우
Patrick method로 이동

NEPI들로 뽑아낸 식

$$(P4+P5)(P6+P7)(P4+P6)(P8+P9)(P5+P8)(P7+P9)=$$

Petrick Method

$$(P5*P6*P9)+(P4*P7*P8)+(P5*P6*P7*P8)+(P4*P6*P8*P9)+(P4*P5*P7*P9)$$

	1	2
1	P1	[0, 2, 4, 6]
2	P2	[0, 2, 8, 10]
3	P3	[0, 4, 8, 12]
4	P4	[2, 3, 6, 7]
5	P5	[2, 3, 10, 11]
6	P6	[4, 5, 6, 7]
7	P7	[4, 5, 12, 13]
8	P8	[8, 9, 10, 11]
9	P9	[8, 9, 12, 13]

Petrick method 결과

모든 PI에 대한 정보

Finish

구현

구현 - PI 찾기

```
for i in range(len(data)-1):
    for j in range(i+1, len(data)):
        arr_tmp = data[i][0].numbers + data[j][0].numbers

        if self.__get_hd(data[i][0].binary, data[j][0].binary) <= 1 and \
            len(arr_tmp) == len(set(arr_tmp)) and \
            len(data[i][0].numbers) == len(data[j][0].numbers):
            self.is_prime = False
            t = copy.deepcopy(data[i])
            t[0].combineNum(data[j][0].numbers)
            self.data[i][2] = True
            self.data[j][2] = True
            t[2] = False
            result.append(t)
```

1. 해밍거리가 1이고
2. PI가 같은 Minterm을
커버하지 않는지

구현 - EPI 찾기

```
def get_epi(self, currentData):
    for i in range(len(currentData[0])):
        count = sum(row[i] for row in currentData)

        if count == 1 and self.isCoveredMintermList[i]==False and \
            self.tableHeaderList[i] not in self.dontcareList: # column count가 1이면
            for j in range(len(currentData)):
                self.isChanged=True
                # column count가 1인 minterm을 포함하는 PI를 찾음
                if currentData[j][i] == 1:
                    self.isCoveredMintermList[i]=True
                    self.isCoveredPIList[j] = 1 # epi 찾음
                    # currentData[j]=[0 for i in range(len(self.tableHeaderList))] # 0으로 초기화

            # 해당 PI가 가지고 있는 minterm들도 모두 cover된 것으로 체크
            for k in range(len(self.data[j][0].numbers)):
                idx = self.tableHeaderList.index(str(self.data[j][0].numbers[k]))
                self.isCoveredMintermList[idx] = True

                for k2 in range(len(currentData)):
                    currentData[k2][idx] = 0

            self.epiList.append(self.data[j])
            break
```

1. 어떤 Minterm을 포함하는 PI가 한개라면 EPI
2. 해당 PI가 커버하는 minterm들 체크

구현 - column dominance

```
def check_column_dominance(self, currentData):  
    result = currentData  
    updateCover = []  
    for i in range(len(currentData[0])):  
        for j in range(len(currentData[0])):  
            if not self.isCoveredMintermList[j] and i != j: #  
                for k in range(len(currentData)):  
                    if currentData[k][j] == 1 and currentData[k][i] == 0:  
                        break  
                else: #j가 superset  
                    self.isCoveredMintermList[i] = True  
                    updateCover.append(i)  
                    self.isChanged = True  
  
    for i in range(len(updateCover)):  
        for j in range(len(currentData)):  
            currentData[j][updateCover[i]] = 0  
    return result
```

i가 0, j가 1인 경우가 없으면
i는 j를 지배한다.

$i = 0 \ \&\& \ j = 1$ 인 경우는 없음

			i	j								
	4	8	10	11	12	15	=>	A	B	C	D	
$m(4,12)^*$	X				X		=>	-	1	0	0	
$m(8,9,10,11)$		X	X	X			=>	1	0	-	-	
$m(8,10,12,14)$		X	X		X		=>	1	-	-	0	
$m(10,11,14,15)^*$			X	X		X	=>	1	-	1	-	

구현 - row dominance

```
def check_row_dominance(self, currentData):
    result = currentData
    updateCover = []
    for i in range(len(currentData)):
        for j in range(len(currentData)):

            if self.isCoveredPIList[i] == 0 and self.isCoveredPIList[j] == 0 and i != j:
                for k in range(len(currentData[0])):
                    if currentData[j][k] == 1 and currentData[i][k] == 0:
                        break
                else:
                    # print(i,j,"rd")
                    self.isCoveredPIList[j] = 2
                    updateCover.append(j)
                    self.isChanged = True

    emptyList = [0 for i in range(len(self.tableHeaderList))]
    for i in range(len(updateCover)):
        currentData[updateCover[i]] = emptyList
    return result
```

i가 0, j가 1인 경우가 없으면
i는 j를 지배한다.

구현 - Petrick method

```
for i in range(1, len(self.NEPIList)):
    p = []
    for j in range(len(self.NEPIList[i])):
```

각 단항식에 추가

```
        for k in range(len(polynomial[i-1])):
```

```
            s = set(list(polynomial[i-1][k]) + [self.NEPIList[i][j].getName()])
```

```
            p.append(s)
```

```
        polynomial.append(p)
```

```
polynomial = polynomial[len(self.NEPIList)-1]
```

```
deduplicate = set(map(lambda x: frozenset(x), polynomial))
```

$X * X = X$

$X + X = X$

모든 괄호를 풀고 set을 이용해서 중복제거

구현 - Petrick method

```
idx = 0
while len(deduplicate) > idx:
    # subset 제거
    deleteList = []
    for i in range(idx+1, len(deduplicate)):
        if deduplicate[idx].issubset(deduplicate[i]):
            deleteList.append(deduplicate[i])
    # 삭제
    for i in range(len(deleteList)):
        deduplicate.remove(deleteList[i])

    idx += 1
```

$$X*(1+Y)=X$$

$$X+XY=X$$

Subset 제거

느낀 점
