논리회로설계
도전과제

도전과제	• Row dominance • Column dominance
문서 제목	결과보고서
작성자	20213038 이보현

목 차

1. 설계 목표
    1.1. epi 탐색
    1.2. row dominance
    1.3. column dominance

2. 개발 내용 및 결과
    2.0 사전 설명
    2.1. epi 탐색
    2.2. row dominance
    2.3. column dominance
    2.4. 최종 결과 출력

1. 설계 목표

1.1 epi 탐색

epi 탐색의 주 목표는 minterm을 단독으로 커버하는 pi를 찾아내 epi로 구분하는 것이다.

- 프로그램의 최종 결과를 출력하기 위해, 찾아낸 epi를 list에 저장한다.
- epi와 해당 epi가 커버하는 모든 minterm을 자료에서 제외한다.
- 프로그램 종료를 위한 조건 판단에 사용하기 위해, epi 탐색과 제외 과정 이후 자료의 상태를 저장한다.
- 해당 과정에서 탐색된 epi를 확인하기 위해 출력한다.
- 현재 자료의 상태를 확인하기 위해 출력한다.
- nepi의 여부를 확인해, 프로그램의 종료 또는 진행을 결정한다.

1.2 column dominance

table의 모든 열을 서로 비교하여 다른 열을 지배하는(완전히 포함하는) 열을 찾아 제거한다.

- 과정을 확인하기 위해 지배자와 피지배자를 출력한다.
- 해당 과정에서 제거되는 열을 확인하기 위해 출력한다.
- 해당 열의 minterm을 모든 자료에서 제거한다.
- 현재 자료의 상태를 확인하기 위해 출력한다.

1.3 row dominance

table의 모든 행을 서로 비교하여 다른 행을 지배하는(완전히 포함하는) 행을 찾아 제거한다.

- 과정을 확인하기 위해 지배자와 피지배자를 출력한다.
- 해당 과정에서 제거되는 행을 확인하기 위해 출력한다
- 해당 행의 pi를 모든 자료에서 제거한다.
- 현재 자료의 상태를 확인하기 위해 출력한다.
- 1.1의 과정에서 저장해둔 자료와 현재 상태를 비교해 프로그램의 종료 여부를 결정한다.

2. 개발 내용 및 결과

2.0 사전 설명

pi 탐색은 이 코드의 column dominance, row dominance와는 연관이 적으므로 설명을 생략하도록 합니다.

탐색 대상의 입력은 전체 코드의 첫 번째 줄에서 minterm의 리스트에 다음과 같은 형식으로 받습니다.

minterm = [size, 1인 minterm의 수, 1인 minterm의 번호 . . .]

아래 설명에 나오는 table이라는 용어는 다음과 같이 pi 탐색 이후 사용되는 table을 뜻합니다.

Prime implicants	Minterm A	B	C	D
P1	v		v	
P2	v	v		
P3			v	v
P4		v		v

2.1 epi 탐색

pi 탐색 이후 epi, column dominance, row dominance를 묶는 while문의 시작 부분입니다.

```
epi =[]
epi_full_list =[]
loop_count =0
while 1 :
   loop_count +=1
   print ("loop_count: ",loop_count)
```

프로그램이 실행되는 동안 찾아낸 epi를 모두 저장하는 epi_full_list와 그 내용을 정해진 형식으로 변환해 저장하고 출력하는 데 사용되는 epi list를 빈 list로 생성합니다.

epi, column dominance, row dominance를 while문의 무한 루프로 묶어 조건이 성립할 경우에만 빠져나가 프로그램을 끝내도록 합니다.

이 while문의 실행 횟수를 나타내는 int 변수 loop_count를 0으로 선언하고 while문을 돌 때마다 1씩 증가시켜 출력합니다.

```
epi_list =[]
   for x in coldic :
     if (len(coldic.get(x))==1)& (coldic.get(x)[0] not in epi_list ):
       epi_list.append(coldic.get(x)[0])
       epi_full_list.append(coldic.get x)[0])
```

epi_list는 현재 while문을 돌 때 생겨난 epi를 담는 list로 매 순환마다 빈 list로 초기화됩니다.
coldic은 table을 세로로 읽어 저장한 dictionary이며 key 값으로는 minterm이 int로, value로는 해당 minterm을 커버하는 pi가 string list에 담겨 저장되어 있습니다.
for문으로 이러한 coldic을 돌며 coldic의 value list의 길이가 1인지 확인합니다. 즉, 해당 minterm을 커버하는 pi가 오직 하나인 경우를 골라내는 것입니다. value list의 길이가 1이고, 그 pi가 epi_list에 들어있지 않다면 epi_list와 epi_full_list에 담아줍니다.

```
rowdic_get =[]
   for y in epi_list :
     for r in rowdic.get(y):
       rowdic_get.append(r)
     for g in rowdic_get :
       if (g in coldic.keys()):
         del coldic[g]
       for k in rowdic.keys():
         if g in rowdic.get(k):
           rowdic.get(k).remove(g)
     rowdic_get =[]
     del rowdic [y]
```

이 코드 속 rowdic은 table을 세로로 저장한 coldic과 달리 table을 가로로 저장해, key로는 pi가, value로는 해당 pi가 커버하는 minterm의 list가 저장되어 있습니다.
앞의 과정을 통해 epi_list에 저장된 pi, 즉 epi들을 돌며 해당 epi가 커버하는 minterm를 rowdic_get list에 저장합니다. 이후, 이 rowdic_get을 돌며 epi가 커버하는 minterm들을 codlic에서 삭제합니다. 또 rowdic.keys()을 for문으로 돌아 각 rowdic의 value list에 접근해 해당 list에 담겨있는 minterm도 삭제합니다. 마지막으로 rowdic의 epi를 삭제하고 rowdic_get을 빈 리스트로 초기화하며, epi와 epi가 커버하는 minterm을 자료에서 완벽히 제거해 줍니다.

```
row_check ={}
col_check ={}
for x in rowdic :
  row_check[x]=rowdic.get(x)
for x in coldic :
  col_check[x]=coldic.get(x)
```

row_check와 col_check는 while문의 마지막 부분에서 simplification을 판단해, break를 통한 프로그램의 종료, 또는 순환을 결정하기 위한 dictionary입니다. for문으로 rowdic과 coldic을 각각 돌며, epi 탐색과 제거를 끝낸 후의 rowdic, coldic의 상태를 row_chcek와 col_check에 담아 저장합니다.

```
print ("<epi_list>")
print (epi_list)
print ("\n")
print ("<rowdic>\n", dict(sorted(rowdic.items())))
print ("\n")
print ("<coldic>\n", dict(sorted(rowdic.items())))
print ("\n")
```

탐색된 epi와 현재 rowdic, coldic의 상태를 사용자에게 알리기 위해 출력합니다.

```
if (rowdic=={}):
     print("break!")
     epi_full_list =sorted(epi_full_list)
     for i in epi_full_list :
       epi.append(i.replace('2','-'))
     break
```

마지막으로 rowdic의 상태에 따라 break를 적용해 프로그램을 종료합니다. rowdic이 빈 dictionary일 경우, 남은 nepi가 없다는 뜻으로 QM method에 따라 전체 탐색을 종료합니다. 그전에, 종료 시에 출력되는 최종 결과를 위해 epi_full_list에 담긴 epi를 지정된 형식으로 바꾸어 epi list에 저장합니다.

2.1 column dominance

```
remove_list =[]
   for x in coldic :
     for y in coldic :
       if (x !=y ):
          if list (sorted (set (coldic .get (x )) &set (coldic .get (y
))))==list (sorted (set (coldic .get (y )))):
           if list (sorted (set (coldic .get (x )) &set (coldic .get (y
))))==list (sorted (set (coldic .get (x )))):
             print (x , " dominate ", y )
             if ((x not in remove_list ) &(y not in remove_list )):
               remove_list .append (x )
           else :
             print (x , " dominate ", y )
             if (x not in remove_list ):
               remove_list .append (x )
```

remove_list는 다음과 같은 과정을 통해 제거 대상으로 선정된 minterm을 저장하는 list입니다.

우선, 이중 for문을 통해 서로 다른 coldic의 value list를 서로 비교해 지배자와 피지배자를 찾습니다. 지배자와 피지배자를 찾는 방법은 두 value list의 교집합과 두 대상을 비교해 주는 것으로, 교집합과 동일한 value list를 지닌 minterm은 피지배자, 다른 하나는 지배자가 됩니다. column dominance에서는 지배 minterm을 제거하기 때문에, 지배 minterm을 remove_list에 담습니다. 이때, 비교 대상인 두 coldic의 value list가 동일한 값을 가지고 있다면 상황에 맞추어 remove_list에 넣을 수 있도록 합니다. 지배 피지배 관계는 출력을 통해 사용자에게 알립니다.

```
coldic_get =[]
   for x in remove_list :
     for r in coldic .get (x ):
       coldic_get .append (r )
     for y in coldic_get :
       if (x in rowdic .get (y )):
         rowdic .get (y ).remove(x )
     coldic_get =[]
     del coldic [x ]
```

앞의 과정을 통해 remove_list에 저장된 minterm을 돌며 해당 minterm을 커버하는 pi를 coldic_get list에 저장합니다. 이후, 이 coldic_get을 돌며 pi의 rowdic value의 list에서 해당 minterm을 삭제합니다. 마지막으로 coldic에서도 해당 minterm을 삭제하고 coldic_get을 빈 리스트로 초기화하며 column dominance를 마칩니다.

```
print ("col_remove: ", remove_list )
print ("\n")
print ("<rowdic>\n ", dict (sorted (rowdic .items ())))
print ("\n")
print ("<coldic>\n ", dict (sorted (coldic .items ())))
print ("\n")
```

column dominance로 삭제된 minterm과 현재 rowdic, coldic의 상태를 사용자에게 알리기 위해 출력합니다.

2.1 row dominance

```
remove_list =[]
   for x in rowdic :
     for y in rowdic :
       if (x !=y ):
          if list (sorted (set (rowdic .get (x ))&set (rowdic .get (y
))))==list (sorted (set (rowdic .get (y )))):
           if list (sorted (set (rowdic .get (x ))&set (rowdic .get (y
))))==list (sorted (set (rowdic .get (x )))):
             print (x , " dominate ", y )
             if ((x not in remove_list ) &(y not in remove_list )):
               remove_list .append (y )
           else :
             if ( y not in remove_list ):
               remove_list .append (y )
               print (x , " dominate ", y )
```

remove_list는 column dominance에서와 같은 역할을 하는 list이며, raw dominance의

메커니즘은 column dominance와 매우 유사합니다. 이중 for문을 통해 서로 다른 rowdic의 value list를 서로 비교해 지배자와 피지배자를 찾습니다. 지배자와 피지배자를 찾는 방법은 두 value list의 교집합과 두 대상을 비교해 주는 것으로 column dominance와 같습니다. 하지만 row dominance에서는 피지배 minterm을 제거하기 때문에, 피지배 minterm을 remove_list에 담습니다. 이때도 역시, 비교 대상인 두 rowdic의 value list가 동일한 값을 가지고 있다면 상황에 맞추어 remove_list에 넣을 수 있도록 합니다. 지배 피지배 관계는 출력을 통해 사용자에게 알립니다.

```
rowdic_get =[]
   for x in remove_list :
     for r in rowdic .get (x ):
       rowdic_get .append (r )
     for y in rowdic_get :
       if (x in coldic .get (y )):
         coldic .get (y ).remove(x )
     rowdic_get =[]
     del rowdic [x ]
```

앞의 과정을 통해 remove_list에 저장된 pi를 돌며 해당 pi가 커버하는 minterm을 rowdic_get list에 저장합니다. 이후, 이 rowdic_get을 돌며 coldic value list의 해당 pi을 삭제합니다. 마지막으로 rowdic에서도 해당 pi를 삭제하고 rowdic_get을 빈 리스트로 초기화하며 row dominance를 마칩니다.

```
print ("row_remove: ", remove_list )
print ("\n")
print ("<rowdic>\n", dict (sorted (rowdic .items ())))
print ("\n")
print ("<coldic>\n", dict (sorted (coldic .items ())))
print ("\n")
```

row dominance로 삭제된 pi와 현재 rowdic, coldic의 상태를 사용자에게 알리기 위해 출력합니다.

```
if ((rowdic ==row_check ) &(coldic ==col_check )):
     epi_full_list =sorted (epi_full_list )
     for i in epi_full_list :
       epi .append (i .replace('2','-'))
     print ("break!")
     print (epi )
     break
```

epi 탐색, column dominance, row dominance를 모두 거친 후에는 simplification을 판단해 프로그램의 중지 또는 반복을 결정합니다. 이를 위해 epi 탐색 이후의 rowdic, coldic의 상태를 저장해두었던 row_check와 col_check를 현재 rowdic, coldic과 비교해 줍니다.
만약, simplification으로 인해 위의 if문이 false가 되었다면 이 프로그램은 다시 while문의 처음 즉, epi 탐색으로 돌아가 진행됩니다. 하지만 반대로, epi 탐색 이후 column dominance와 row dominance에서 삭제된 자료가 없어 if문이 true가 된다면 epi_full_list의 값을 정해진 형식으로 변환해 epi list에 저장한 후 break를 통해 while문을 빠져나오게 됩니다.

2.4 최종 결과 출력

```
print ("answer: ", pi_list + ["EPI"] + epi )
```

while문을 빠져나온 후에는 지정된 형식으로 변환된 pi와 epi를 출력해 주며 프로그램이 끝나게 됩니다.