논리회로 설계 과제 보고서
20191660 전병우

row dominance column dominance petrick method 구현

아래의 코드는 전체 코드입니다.
epi와 pi 구현은 과제 제출 하였으니 설명은 생략을 하겠습니다.
전체 코드 후에 Row dominance, column dominance, petrick method 순으로 설명하겠습니다.

```
def findone(numlist, a):
    arr = []
    for i in range(a):
        arr.append([])
    for i in range(len(numlist)):
        arr[numlist[i].count('1')].append(numlist[i])
    return arr
def makecheck(arr):
    check = []
    for i in range(len(arr)):
        check.append([])
        if len(arr[i]) == 0:
            continue
        for j in range(len(arr[i])):
            check[i].append("0")
    return check
def findpi(numlist, a):
    answer = []
    b = a+1
    while 1:
        arr = findone(numlist, b)
        numlist = []
        b -= 1
        #체크 배열 만들기
        check = makecheck(arr)
        for i in range(len(arr) - 1):
            if len(arr[i]) == 0:
                continue
            if len(arr[i+1]) == 0:
                continue
            for j in range(len(arr[i])):
                for k in range(len(arr[i+1])):
                    cnt = ""#합쳐지는 것 문자열
                    cnt1 = 0#하나만 다른지 확인하는 방법
                    for l in range(a):
                        if arr[i][j][l] != arr[i + 1][k][l]:
                            cnt += "2"
                            cnt1 += 1
                        if arr[i][j][l] == arr[i + 1][k][l] and arr[i]
[j][l] == "0":
```

```
                        cnt += "0"
                    if arr[i][j][l] == arr[i + 1][k][l] and arr[i]
[j][l] == "2":
                        cnt += "2"
                    if arr[i][j][l] == arr[i + 1][k][l] and arr[i]
[j][l] == "1":
                        cnt += "1"
                if cnt1 == 1:
                    if cnt not in numlist:
                        numlist.append(cnt)
                    check[i][j] = "1"
                    check[i + 1][k] = "1"
        checkcnt = 0
        for i in range(len(check)):
            if len(check[i]) == 0:
                continue
            for j in range(len(arr[i])):
                if check[i][j] == "0":
                    answer.append(arr[i][j])
                else:
                    checkcnt += 1
        if checkcnt == 0:
            break
    return answer
def findepi(answer, xlist, n,a):
    epi = []
    for i in range(len(answer)+1):
        epi.append([])
        for j in range(n+1):
            epi[i].append("-")
    for i in range(1,n+1):
        epi[0][i] = xlist[i-1]
    for j in range(1,len(answer)+1):
        epi[j][0] = answer[j-1]
    for i in range(1,len(answer)+1):
        checkinglist = []
        checkinglist.append(epi[i][0])
        checkinglist = checking(checkinglist,a)
        for j in range(1,n+1):
            if epi[0][j] in checkinglist:
                epi[i][j] = "1"
    realepi = []
    for j in range(1,n+1):
        count = []
        for i in range(1,len(answer)+1):
            if epi[i][j] == "1":
                count.append(epi[i][0])
        if len(count) == 1 and count[0] not in realepi:
            realepi.append(count[0])
    return realepi
def checking(checkinglist,a):
```

```
    checkinglist1 = []
    for i in range(len(checkinglist)):
        for j in range(a):
            if checkinglist[i][j] == '-':
                checkinglist1.append(checkinglist[i][:j]
+"0"+checkinglist[i][j+1:a])
                checkinglist1.append(checkinglist[i][:j] + "1" +
checkinglist[i][j + 1:a])
                break
    checkinglist = checkinglist1
    #재귀 끝내는 것
    checkbreak = 0
    for i in range(len(checkinglist)):
        for j in range(a):
            if checkinglist[i][j] == '-':
                checkbreak += 1
    if checkbreak == 0:
        return checkinglist
    return checking(checkinglist,a)
def reducedlist(epii, a, answer, xlist):
    reducedxlist = checking(epii,a)
    result = []
    for i in xlist:
        if i not in reducedxlist:
            result.append(i)
    xlist = result
    # print("---------")
    # print(answer)
    #print(epii)
    if len(epii)!=0:
        answer.remove(epii[0])
    print(answer)
    #reducedlist 만들기
    relist = []
    n = len(xlist)
    for i in range(len(answer) + 1):
        relist.append([])
        for j in range(n + 1):
            relist[i].append("-")
    for i in range(1, n + 1):
        relist[0][i] = xlist[i - 1]
    for j in range(1, len(answer) + 1):
        relist[j][0] = answer[j - 1]
    for i in range(1, len(answer) + 1):
        checkinglist = []
        checkinglist.append(relist[i][0])
        checkinglist = checking(checkinglist, a)
        for j in range(1, n + 1):
            if relist[0][j] in checkinglist:
                relist[i][j] = "1"
    return relist
```

```
def inlist(rest_list,rest_list1):
    answer = 1
    for i in rest_list1:
        if i not in rest_list:
            answer = 0
    return answer
def rowdominance(relist):
    row = []
    row_number = []
    changed = 0
    for i in range(1, len(relist)):
        rest_list = list(filter(lambda x: relist[i][x] == '1',
range(1, len(relist[i]))))
        for j in range(1, len(relist)):
            if j == i:
               continue
            rest_list1 = list(filter(lambda x: relist[j][x] == '1',
range(1, len(relist[j]))))
            if inlist(rest_list, rest_list1):
                row.append(relist[j][0])
                row_number.append(j)
    row_number = list(set(row_number))
    row = list(set(row))
    row_number.sort(reverse=True)
    print(row_number)
    for i in row_number:
        del relist[i]
        changed = 1
    return relist,row, changed
    #print(relist)
    #for i in range(1,len(relist)-1):
    #    if relist[i][-1] !=1:
def columndominance(relist):
    column = []
    column_number = []
    changed = 0
    for i in range(1, len(relist[0])):
        rest_list = list(filter(lambda x: relist[x][i] == '1',
range(1, len(relist))))
        for j in range(1, len(relist[0])):
            if j == i:
                continue
            rest_list1 = list(filter(lambda x: relist[x][j] == '1',
range(1, len(relist))))
            if inlist(rest_list, rest_list1):
                column.append(relist[0][i])
                column_number.append(i)
    column = list(set(column))
    column_number = list(set(column_number))
    column_number.sort(reverse=True)
    for i in column_number:
```

```
        for j in range(len(relist)):
            del relist[j][i]
            changed = 1
    for i in range(1,len(relist)):
        cnt1 = 0
        for j in range(1,len(relist[0])):
            if relist[i][j]=='1':
                cnt1+=1
        if cnt1==0:
            del relist[i]
    return relist, column, changed
def secondaryepi(relist):
    # for i in range(1,len(relist)-1):
    #     cnt22 = 0
    #     for j in range(1,len(relist[0])):
    #         if relist[i][j]=='1':
    #             cnt22+=1
    #     if cnt22 == 0:
    #         del relist[i]
    # for i in range(len(relist)):
    #     print(relist[i])
    secondaryepi = []
    idx = []
    for j in range(1,len(relist[0])):
        count = []
        idxcount = []
        for i in range(1,len(relist)):
            if relist[i][j] == "1":
                count.append(relist[i][0])
                idxcount.append(i)
        if len(count) == 1:
            secondaryepi.append(count[0])
            idx.append(idxcount[0])
    idx.sort(reverse=True)
    print("--------------")
    print(idx)
    for i in range(len(relist)):
        print(relist[i])
    for i in idx:
        del relist[i]
    # for i in range(1,len(relist)):
    #     for j in secondaryepi:
    #         if relist[i][0] in secondaryepi:
    #             del relist[i]
    # print(secondaryepi)
    # for i in range(len(relist)):
    #     print(relist[i])
    # print("\n")
    return secondaryepi,relist
def interchangable(relist):
    for i in range(1,len(relist)-1):
```

```
        cnt22 = 0
        for j in range(1,len(relist[0])):
            if relist[i][j]=='1':
                cnt22+=1
        if cnt22==0:
            del relist[i]
    while(1):
        checklist=[]
        changed = 0
        for i in range(1,len(relist)):
            checklist = []
            isdelete = 0
            for j in range(i+1,len(relist)):
                if relist[i][1:-1]==relist[j][1:-1]:
                    checklist.append(j)
                    changed = 1
            checklist.sort(reverse=True)
            for k in checklist:
                del relist[k]
                isdelete = 1
            if isdelete == 1:
                break
        if changed == 0:
            break
    return relist
def patric(relist):
    patric = []
    for i in range(1,len(relist[0])):
        patric_row = []
        for j in range(1,len(relist)):
            if relist[j][i] == '1':
                patric_row.append(j)
        patric.append(patric_row)
    # 가능한 조합 리스트 만들기
    # print(patric)
    list1 = []
    notselect = []
    for i in range(len(patric)):
        insert = []
        covercount = []
        for j in range(len(patric[0])):
            if (patric[i][j] in list1):
                covercount.append(patric[i][j])
            if (patric[i][j] not in list1) and (patric[i][j] not in
notselect):
                insert.append(patric[i][j])
        if len(insert) == 1 and len(covercount)==1:
            notselect.append(insert[0])
            continue
        if len(insert)==1:
            list1.append(insert[0])
```

```
        if len(insert)==2:
            list1.append(insert[0])
            notselect.append(insert[1])
    list1.sort(reverse=True)
    # print("list1----------------------")
    # print(list1)
    # for i in range(len(relist)-1,0,-1):
    #     if i not in list1:
    #         del relist[i]
    # print("relist without list1------------------")
    # for i in range(len(relist)):
    #     print(relist[i])
    patricanswer = []
    for i in list1:
        patricanswer.append(relist[i][0])
    return patricanswer
def solution(minterm):
    a = minterm[0]
    n = minterm[1]
    numlist = []
    for i in range(n):
        x = minterm[i + 2]
        bin_num = bin(x)[2:].zfill(a)
        numlist.append(bin_num)
    xlist = numlist
    answer = findpi(numlist, a)
    answer = set(answer)
    answer = list(answer)
    answer.sort()
    for i in range(len(answer)):
        answer[i] = answer[i].replace('2','-')
    epii=findepi(answer, xlist, n,a)
    relist = reducedlist(epii, a, answer, xlist)
    for i in range(len(relist)):
        print(relist[i])
    relist = interchangable(relist)
    for i in range(1,len(relist)-1):
        cnt22 = 0
        for j in range(1,len(relist[0])):
            if relist[i][j]=='1':
                cnt22+=1
        if cnt22==0:
            del relist[i]
    # for i in range(len(relist)):
    #     print(relist[i])
    nothingchange = 0
    secondepi = []
    while(1):
        if len(relist)==1:
            print("finished and second epi : ")
            print(secondepi)
```

```
            break
        relist, row, changed2 = rowdominance(relist)
        # for i in range(len(relist)):
        #     print(relist[i])
        relist, column, changed1 = columndominance(relist)
        # for i in range(len(relist)):
        #     print(relist[i])
        if (changed1==1) or (changed2==1):
            second,relist= secondaryepi(relist)
            secondepi = secondepi+second
        elif (changed1!=1) and (changed2!=1):
            nothingchange = 1
            break
    if nothingchange==1:
        print("---------------")
        for i in range(len(relist)):
            print(relist[i])
        print("petrick : ")
        print(patric(relist))
        # print("---------------")
        # for i in range(len(relist)):
        #     print(relist[i])
    # for i in range(len(relist)):
    #     print(relist[i])
    # relist, row,changed2 = rowdominance(relist)
    # print(changed2)
    #print(row)
    # for i in range(len(relist)):
    #     print(relist[i])
    # relist,second = secondaryepi(relist)
    # print(second)
    # print(relist)
    answer.append("EPI")
    answer = answer+epii
    return answer
# print(solution([4,11,0,2,5,6,7,8,10,12,13,14,15]))
# print(solution([3,6,0,1,2,5,6,7]))
print(solution([4,8,0,4,8,10,11,12,13,15]))
```

1. row dominance
row dominance 설명에 앞서 solution 에 대한 설명이 있어야 함으로 solution,reducedlist 에 대한 설명부터 하겠습니다.

```
def solution(minterm):
    a = minterm[0]
    n = minterm[1]
    numlist = []
    for i in range(n):
        x = minterm[i + 2]
        bin_num = bin(x)[2:].zfill(a)
        numlist.append(bin_num)
    xlist = numlist
    answer = findpi(numlist, a)
    answer = set(answer)
    answer = list(answer)
    answer.sort()
    for i in range(len(answer)):
        answer[i] = answer[i].replace('2','-')
    epii=findepi(answer, xlist, n,a)
    relist = reducedlist(epii, a, answer, xlist)
    relist = interchangable(relist)
설명 1로 이동
    #for i in range(1,len(relist)-1):
    #    cnt22 = 0
    #    for j in range(1,len(relist[0])):
    #        if relist[i][j]=='1':
    #            cnt22+=1
    #    if cnt22==0:
    #        del relist[i]
    nothingchange = 0
    secondepi = []
    while(1):
        if len(relist)==1:
            print("finished and second epi : ")
            print(secondepi)
            break
        relist, row, changed2 = rowdominance(relist)
설명 2로 이동
        # for i in range(len(relist)):
        #     print(relist[i])
설명 3으로 이동
       relist, column, changed1 = columndominance(relist)
        # for i in range(len(relist)):
        #     print(relist[i])
        if (changed1==1) or (changed2==1):
설명 4로 이동
            second,relist= secondaryepi(relist)
            secondepi = secondepi+second
        elif (changed1!=1) and (changed2!=1):
            nothingchange = 1
            break
설명 5로 이동
      if nothingchange==1:
        for i in range(len(relist)):
            print(relist[i])
        print("petrick : ")
        print(patric(relist))

    answer.append("EPI")
    answer = answer+epii
```

```
    return answer
print(solution([4,8,0,4,8,10,11,12,13,15]))
```

설명 1

relist = reducedlist(epii, a, answer, xlist)

이 코드는 수업에서 배운대로 기존에 구한 epi 와 pi 를 통해 축소된 리스트를 만드는 과정입니다.

아래의 코드는 reducedlist 입니다.

```
def reducedlist(epii, a, answer, xlist):
    reducedxlist = checking(epii,a)
    result = []
    for i in xlist:
        if i not in reducedxlist:
            result.append(i)
    xlist = result
    #print(epii)
    if len(epii)!=0:
        answer.remove(epii[0])
    print(answer)
    #reducedlist 만들기
    relist = []
    n = len(xlist)
    for i in range(len(answer) + 1):
        relist.append([])
        for j in range(n + 1):
            relist[i].append("-")
```

체크가 안된 칸은 잘 보이기 위해 – 로 채워넣었습니다.

```
    for i in range(1, n + 1):
        relist[0][i] = xlist[i - 1]
    for j in range(1, len(answer) + 1):
        relist[j][0] = answer[j - 1]
    for i in range(1, len(answer) + 1):
        checkinglist = []
        checkinglist.append(relist[i][0])
        checkinglist = checking(checkinglist, a)
```

checking 함수는 ‘-0-1’ 같이 문자열로 표현 된 이진수를 가능한 십진수로 만들어주는 함수 입니다.

```
        for j in range(1, n + 1):
            if relist[0][j] in checkinglist:
                relist[i][j] = "1"
```

구한 checking list 를 통해 해당되는 minterm 에 ‘1’로 체크를 해주었습니다.

```
    return relist
```

후에 interchangable 한 pi 를 없애기 위해 interchangable 함수를 사용하였습니다.

Interchangable 함수를 간단하게 설명해 드리겠습니다.

```
def interchangable(relist):
    for i in range(1,len(relist)-1):
        cnt22 = 0
        for j in range(1,len(relist[0])):
            if relist[i][j]=='1':
                cnt22+=1
```

```
        if cnt22==0:
            del relist[i]

    while(1):
        checklist=[]
        changed = 0
        for i in range(1,len(relist)):
            checklist = []
            isdelete = 0
            for j in range(i+1,len(relist)):
                if relist[i][1:-1]==relist[j][1:-1]:
                    checklist.append(j)
                    changed = 1
            checklist.sort(reverse=True)
            for k in checklist:
                del relist[k]
                isdelete = 1
            if isdelete == 1:
                break
        if changed == 0:
            break
```

위 함수는 간단하게 같은 minterm을 표현하는 pi들을 없애주는 것입니다. 만약 같은 것이 있을 시에 처음 나오는 pi는 살리고 뒤에 나오는 pi를 처리하는 식으로 구현 하였습니다.

위 과정들을 거치면 아래의 그림이 나옵니다.

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

위의 결과가 reducedlist 함수를 통해 리턴되는 relist입니다. 교제와 똑같이 구현을 위해 2차원 배열을 써서 표현 하였습니다.

설명 2

아래는 rowdominance 코드입니다

```
def rowdominance(relist):
    row = []
    row_number = []
    changed = 0
```

changed는 변화가 있었는지 알려 줍니다. row 배열은 row dominance로 인해 없어지는 row를 저장합니다.

```
    for i in range(1, len(relist)):
        rest_list = list(filter(lambda x: relist[i][x] == '1',
range(1, len(relist[i]))))
```

filter 함수를 통해서 '1'(체크)의 인덱스 배열을 저장하는 rest_list를 만들었습니다.

```
        for j in range(1, len(relist)):
            if j == i:
               continue
```

이 코드가 있으므로서 자신과 같은 것을 row dominance로 인식하지 않습니다.

```
            rest_list1 = list(filter(lambda x: relist[j][x] ==
'1', range(1, len(relist[j]))))
```

이 함수도 마찬가지로 나머지 row의 1의 인덱스를 저장합니다.

```
            if inlist(rest_list, rest_list1):
                row.append(relist[j][0])
                row_number.append(j)
```

처음에 구한 rest_list안에 비교하는 대상인 rest_list1가 있는지 확인하며 값이 참이면 row배열에 해당 pi를 넣어줍니다. 그리고 relist에서 row dominance로 인해 지워지는 pi를 지우기 위해 row_number에 인덱스 값을 넣어줍니다.

```
    row_number = list(set(row_number))
    row = list(set(row))
    row_number.sort(reverse=True)
```

이는 인덱스를 역으로 정렬을 해서 인덱스 오류가 나지 않게 하기 위함입니다.

```
    for i in row_number:
        del relist[i]
        changed = 1
```

만약 row dominance에 해당하면 해당 pi를 지워주며 변화가 있었음으로 changed를 1로 만들어 줍니다.

```
    return relist,row, changed
relist :
['-', '1010', '1011', '1101', '1111']
['101-', '1', '1', '-', '-']
['11-1', '-', '-', '1', '1']
['1-11', '-', '1', '-', '1']
row :
['10-0', '110-']
```

설명 3. column dominance

구현 방법은 row dominance와 같습니다. 원리 가 같으며 단지 바뀐 것은 이제 column에서의 지배를 찾아야 하기 때문에 구현 과정에서 행과 열의 순서와 지배하는 쪽이 지워져야 한다는 것만 다릅니다.

```
def columndominance(relist):
    column = []
    column_number = []
    changed = 0
    for i in range(1, len(relist[0])):
        rest_list = list(filter(lambda x: relist[x][i] == '1',
range(1, len(relist))))
        for j in range(1, len(relist[0])):
            if j == i:
                continue
            rest_list1 = list(filter(lambda x: relist[x][j] ==
'1', range(1, len(relist))))
            if inlist(rest_list, rest_list1):
                column.append(relist[0][i])
                column_number.append(i)
    column = list(set(column))
```

```
    column_number = list(set(column_number))
    column_number.sort(reverse=True)

    for i in column_number:
        for j in range(len(relist)):
            del relist[j][i]
            changed = 1

    for i in range(1,len(relist)):
        cnt1 = 0
        for j in range(1,len(relist[0])):
            if relist[i][j]=='1':
                cnt1+=1
        if cnt1==0:
            del relist[i]
    return relist, column, changed
```

아래는 columndominance를 취한 뒤의 결과 값입니다.

```
after columndominance
['-', '1010', '1101']
['101-', '1', '-']
['11-1', '-', '1']
```

두번의 과정을 통해 나온 reduced list의 secondary epi를 구해 줍니다. 다음으로 secondary epi를 찾는 함수를 설명 드리겠습니다.

설명 4

```
def secondaryepi(relist):

    secondaryepi = []
    idx = []
    for j in range(1,len(relist[0])):
        count = []
        idxcount = []
        for i in range(1,len(relist)):
            if relist[i][j] == "1":
                count.append(relist[i][0])
                idxcount.append(i)
        if len(count) == 1:
            secondaryepi.append(count[0])
            idx.append(idxcount[0])
```

기존에 epi를 찾을 때의 과정과 같게 minterm을 커버하는 1의 갯수가 1인 pi를 찾습니다. 그렇게 해서 찾은 pi의 인덱스를 저장하여 relist에서 없애줍니다.

```
    idx.sort(reverse=True)
    for i in idx:
        del relist[i]
    return secondaryepi,relist
```

아래는 secondaryepi 와 relist의 값입니다.
second epi :
['101-', '11-1']
relist :
[['-', '1010', '1101']]

여기서는 앞에서의 과정을 통해 petrick method를 쓸 수 없으므로 petrick method는 다른 예시로 설명 드리겠습니다.

이 예시에서의 결과

finished and second epi :
['101-', '11-1']
['101-', '10-0', '110-', '11-1', '1-11', 'EPI', '--00']

설명 5

petrick method 를 사용할 조건을 만들기 위하여
solution([3,6,0,1,2,5,6,7])으로 진행하였다.
이 코드에서는 row dominance와 column dominance가 적용 되지 않으므로
row dominance와 column dominance가 적용 되지 않았다는 말은 reduced list 가 변경되지 않았다는 의미 임으로 solution의 elif (changed1!=1) and (changed2!=1): 조건이 충족 됨으로 patric 함수가 호출된다.
reduced list 는 아래와 같이 나온다.
['-', '000', '001', '010', '101', '110', '111']
['00-', '1', '1', '-', '-', '-', '-']
['0-0', '1', '-', '1', '-', '-', '-']
['11-', '-', '-', '-', '-', '1', '1']
['1-1', '-', '-', '-', '1', '-', '1']
['-01', '-', '1', '-', '1', '-', '-']
['-10', '-', '-', '1', '-', '1', '-']

Patric 함수

```
def patric(relist):
    patric = []
    for i in range(1,len(relist[0])):
        patric_row = []
        for j in range(1,len(relist)):
            if relist[j][i] == '1':
                patric_row.append(j)
        patric.append(patric_row)
```

위의 코드는 수업에서 배운대로 각 민텀을 커버하는 pi를 2중 리스트를 통하여 구현하였다. 구현 결과는 다음과 같다.
[[1, 2], [1, 5], [2, 6], [4, 5], [3, 6], [3, 4]]

```
    list1 = []
    notselect = []
```

```
    for i in range(len(patric)):
        insert = []
        covercount = []
        for j in range(len(patric[0])):
            if (patric[i][j] in list1):
                covercount.append(patric[i][j])
            if (patric[i][j] not in list1) and (patric[i][j] not
in notselect):
                insert.append(patric[i][j])
        if len(insert) == 1 and len(covercount)==1:
            notselect.append(insert[0])
            continue
        if len(insert)==1:
            list1.append(insert[0])
        if len(insert)==2:
            list1.append(insert[0])
            notselect.append(insert[1])
```

위의 코드는 patric을 돌면서 list1 에 선택된 것들 그리고 notselect에는 선택되지 않은 것들을 넣는다. 그리고 계속 for 문을 돌면서 if len(insert) == 1 and len(covercount)==1: if len(insert)==1: if len(insert)==2: 조건에 따라 처리 해준다.

```
    list1.sort(reverse=True)
    patricanswer = []
    for i in list1:
        patricanswer.append(relist[i][0])
    return patricanswer
```

이런 과정을 거치면

```
petrick :
['-10', '1-1', '00-']
```

이런 결과가 나옵니다.