mc-cluskey algorithm 결과 보고서.

row dominance와 column dominance 구현했음,

구현을 위해 만든 파일들: bin.h(.cpp), myfunc.h(.cpp), testFunc.h(.cpp), main.cpp

1. bin: Bin 클래스를 구현.

-Bin class

Bin 클래스는 pi와 minterm들을 하나로 묶어놓은 클래스로서, pi는 string _binary로 minterm을 담는 변수는 set<int> _nums로 구현하였다.

예시) _binary = "111-" -> _nums = { 14, 15 }

지원되는 연산에는 + (1000 + 0000 = -000), == 등이 있다.

2. myfunc.h(cpp): findPI, findEPI, findRowDominance, findColumnDominance 등을 구현하였다.

3. testFunc.h(cpp): myfunc에서 만든 함수들을 테스트하고 MC-Cluskey 알고리즘의 핵심로직 findEPI->findColumnDominance->findRowDominance을 차례로 재귀적으로 실행하는 함수를 구현하였다.

4. main.cpp: 테스트케이스를 입력받고 testFunc의 핵심로직을 실행하는 함수를 호출한다.

-핵심로직 설명:

함수호출은 다음과 같이 된다. 1. findPI -> 2. findEPI -> 3. refineMinterms -> 4. findNEPI -> 5. findColumnDominance -> 6. findRowDominance -> 2~6반복...

설명을 하자면,

1. 입력이 vector<int> input으로 들어온다. input으로 minterm들만 분리하여 vector<minterms>을 만들고 이것으로 다시 vector<Bin> bins를 만든다. 예를 들어 input이 [2, 2, 0, 2]라면 minterms에는 { 0, 2 }, bins에는 { _binary: "00", _nums: { 0 } }, { _binary: "10", _nums: { 2 } }와 같이 저장된다(일부 변수를 생략하였다.). findPI에서 이 bins를 묶을 수 있는지 판단하여 pi로 만들어준다. 방금과 같은 경우에는 { _binary: "-0", _nums: { 0, 2 } } 라는 새로운 Bin객체가 vector<Bin> pi에 추가되는 것이다.
2. pi들을 담고 있는 vector<Bin> pi와 minterm들을 담고 있는 vector<int> minterms 이용해서 findEPI에서 vector<Bin> epi를 찾아낸다. Epi는 특정 minterm을 혼자서만 포함하고 있

는 pi를 말한다. 따라서 minterms안의 원소들을 bins._nums이 포함하는지 찾아보면서 찾을 때마다 count를 1 증가시키고 그 Bin객체를 따로 저장한다. 만약 count == 1이라면 그 Bin객체가 epi라는 의미임으로 vector<Bin> epi에 그 객체를 추가한다.

3. 우선 epi는 알고리즘에서 무조건 선택되는 pi이기 때문에 epi에 속하는 minterm들은 밑에서 판단할 때 필요하지 않다. 따라서 refineMinterms 함수를 통해 minterms에서 epi에 속하는 minterm들을 제외시킨다. Row domiance와 Column dominance는 굳이 판단할 필요 없는 nepi와 minterms를 지워나가면서 효율적인 pi를 찾는 과정이다. 따라서 우선 이를 시행하기 전에 vector<Bin> nepi를 찾아내야 한다. 아까 찾은 epi를 pi에서 제외시키면 nepi다. 이 일을 findNEPI에서 처리한다.

4. 이제 findColumnDominance를 시행한다. findCoulmnDominance에서는 특정 minterm을 가지고 있는 pi집합이 다른 minterm을 가지고 있는 pi집합을 포함할 때 포함하는 집합을 가지고 있는 minterm을 제외하는 과정이다. 따라서 minterm: { pi1, pi2, .. } 와 같은 자료구조로 표현할 수 있다. 이를 구현하기 위해 map<int, set<stirng>> table객체를 만들어 주었다. 이 객체는 다음과 같이 데이터를 저장한다. minterm(key값) : { minterm을 포함하는 pi들 }(value값 (set<int>형)). 이제 table에서 각 key값을 돌면서 각 value값의 포함관계를 확인하는데 이는 c++ STL에 includes 함수를 사용해서 확인한다. 만약 포함관계가 나타날 경우 다른 집합을 포함하는 집합을 갖고 있는 minterm을 table에서 삭제한다. Interchable에 경우엔 임의의 key를 제외시킨다.

5. rowDominance를 시행한다. Row dominance는 특정 pi가 갖고 있는 minterm들이 다른 pi가 갖고 있는 minterm들을 포함하는 것을 말한다. 이 때 포함되는 minterm 집합을 갖고 있는 pi는 제외되어야 한다. 이 때 포함관계를 확인할 때 사용되는 _nums 속 minterm들은 이전에 제외된 minterm일 경우 포함관계를 판단하는데 사용되면 안된다. 따라서 현재 남은 minterms와 두 Bin객체가 가지고 있는 _nums의 합집합을 구하여 포함관계를 확인하는 isRowDominance 함수를 따로 만들어주었다. True가 반환될 경우, 포함되는 집합을 가지고 있는 Bin객체를 vector<Bin> nepi에서 제외시킨다.

6. 이렇게 해서 vector<Bin> nepi와 vector<int> minterms가 도출된다. 이 값을 가지고 다시 2~5의 과정을 반복한다. 만약 minterms의 원소 개수가 0이 되거나 2~5의 과정을 거쳐도 minterms의 개수에 변화가 없을 경우 루프를 빠져나온다.

```
findEPI:
       9   13  15  17  25  26  27  31
0-1--  -   O   O   -   -   -   -   -
0--0-  O   O   -   -   -   -   -   -
110-1  -   -   -   -   O   -   O   -
11-1-  -   -   -   -   -   O   O   O
--001  O   -   -   O   O   -   -   -
--11-  -   -   O   -   -   -   -   O

epi: ['--001', '11-1-']
after del epi minterms: 13, 15,
nepi:['0-1--', '0--0-', '110-1', '--11-']
----------------------------
coulmn dominance start:
pi: ['0-1--', '0--0-', '110-1', '--11-']
minterms: 13, 15,
    0-1--  0--0-  110-1  --11-
13  O      O      -      -
15  O      -      -      O

after column dominance minterms: 13, 15,
----------------------------
row dominance:
       13  15
0-1--  O   O
0--0-  O   -
110-1  -   -
--11-  -   O
```

1. epi 찾기. 특정 minterm을 유일하게 감싸고 있는 pi를 찾는다. epi와 nepi를 출력한다.

```
coulmn dominance start:
pi: ['0-1--', '0--0-', '110-1', '11-1-', '-0-01', '-1-10', '--001', '--11-']
minterms: 1, 9, 13, 15, 17, 25, 26, 27, 30, 31,
    0-1--  0--0-  110-1  11-1-  -0-01  -1-10  --001  --11-
1   -      O      -      -      O      -      O      -
9   -      O      -      -      -      -      O      -
13  O      O      -      -      -      -      -      -
15  O      -      -      -      -      -      -      O
17  -      -      -      -      O      -      O      -
25  -      -      O      -      -      -      O      -
26  -      -      -      O      -      O      -      -
27  -      -      O      O      -      -      -      -
30  -      -      -      O      -      O      -      O
31  -      -      -      O      -      -      -      O

1>=9
30>=26
after column dominance minterms: 9, 13, 15, 17, 25, 26, 27, 31,
```

2. column dominance

column dominance를 찾는다. 표의 좌측에 minterm들이 놓여 있고 상단에 pi가 놓여 있기 때문에 눈으로 볼 때는 가로로 감싸는 형태를 찾으면 된다. 1>=9는 1이 9를 포함한다는 뜻이다. 부분집합 기호를 비슷하게 표현하였다.

두 집합 중 하나가 다른 하나에 포함되는 형태일 때, 포함하는 쪽 집합을 제거한다. 즉 1>=9이면 pi들을 담고 있는 집합 1을 제거하는 것이다. 맨 아랫줄에 보면 1과 30이 사라졌다는 것을 알 수 있다.

```
row dominance:
       9   13  15  17  25  26  27  31
0-1--  -   O   O   -   -   -   -   -
0--0-  O   O   -   -   -   -   -   -
110-1  -   -   -   -   O   -   O   -
11-1-  -   -   -   -   -   O   O   O
-0-01  -   -   -   O   -   -   -   -
-1-10  -   -   -   -   -   O   -   -
--001  O   -   -   O   O   -   -   -
--11-  -   -   O   -   -   -   -   O

11-1->=-1-10
--001>=-0-01
after row dominance nepi: ['0-1--', '0--0-', '110-1', '11-1-', '--001', '--11-']
------start-----------------
pi: ['0-1--', '0--0-', '110-1', '11-1-', '--001', '--11-']
minterms: 9, 13, 15, 17, 25, 26, 27, 31,
----------------------------
```

3. row dominance

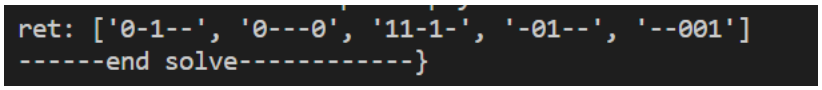
column dominance와 마찬가지로 눈으로 볼 땐 가로로 감싸는 형태를 찾으면 된다.

row dominance에서는 minterm들을 담고 있는 집합이 있을 때, 포함되는 쪽 집합을 가진 pi를 제거한다. 11-1->=-1-10이면 pi 11-1-이 pi –1-10를 포함하고 있다는 것이다. 따라서 이 경우엔 –1-10를 second epi가 될 수 있는 후보들을 담고 있는 pi array에서 제외시킨다.

```
C:\Users\jch61\ps\mccluskey-algorithm>bin
["00-", "0-0", "11-", "1-1", "-01", "-10"]
bins is empty.
["0--"]
["0--"]
["---"]
["---"]
["-"]
["-"]
bins is empty.
bins is empty.
["01----", "0-0---", "0--00-", "0--11-", "0--1-1", "0---01"]
["0-0---", "0--00-", "0--11-", "01----"]
["101-", "10-0", "110-", "11-1", "1-11", "--00"]
["--00"]
00- 0-0 11- 1-1 -01 -10 EPI
0-- EPI 0--
--- EPI ---
- EPI -
EPI
01---- 0-0--- 0--00- 0--11- 0--1-1 0---01 EPI 0-0--- 0--00- 0--11- 01----
101- 10-0 110- 11-1 1-11 --00 EPI --00
```

1, 2, 3의 과정은 재귀 호출을 통해서 계속 진행된다.

minterms의 원소의 개수가 0이거나, 1, 2, 3의 과정을 거쳐도 minterms의 개수가 변하지 않는다면 (더 이상 표 size를 줄일 수 없다면) 루프를 벗어난다.



이렇게 루프를 벗어나면 최종적으로 최적화를 위해 사용할 수 있는 epi와 secondary epi들을 담는 vector<Bin>을 리턴한다.