2022-1 논리회로설계

도전과제 : Row dominance, Column dominance

목차

1. (Prime) Implicant

```
class Implicant {
    String binary;
    int num, number_Of_1=0;
    boolean combined, isEPI, dominated;
    ArrayList<Integer> mintermList = new ArrayList<>();

    public Implicant(String binary, ArrayList<Integer> list1, ArrayList<Integer> list2) {
        this.binary = binary;
        setNumber_Of_1(binary);
        mintermList.addAll(list1);
        mintermList.addAll(list2);
    }

    public Implicant(String binary, int m) {
        this.binary = binary;
        setNumber_Of_1(binary);
        mintermList.add(m);
    }

    void setNumber_Of_1(String binary) {
        for(int i=0; i<binary.length(); i++) {
            if(binary.charAt(i) == '1') number_Of_1++;
        }
    }
}
```

기존 과제(PI구하기)에서 Implicant는 클래스로 구현하였다.

클래스로 구현된 Implicant의 필드 중에서 이번 과제인 EPI찾기, Row dominance, Column dominance에 관여하는 것은 다음과 같다.

1. 자신이 커버하는 minterm들을 원소로 가진 리스트 :
   `ArrayList<Integer> mintermList = new ArrayList<>();`
2. 자신이 EPI(secondary epi등을 포함한) 인지 나타내는 값 :

   boolean isEPI
3. 자신이 Row dominance로 지배 받는지를 나타내는 값 :

   boolean dominated
4. PI를 정렬하고 난 뒤 순서대로 부여한 번호 :

   Int num

2. Minterm

```
class Minterm {
    int number;
    ArrayList<Integer> coveredBy = new ArrayList<>();
    boolean inEPI, dominate;
    public Minterm(int number) {
        this.number = number;
    }
}
```

테스트 케이스에서 주어진 minterm들 각각을 클래스로 구현하였다.

클래스로 구현된 minterm의 필드 중 이번 과제인 EPI찾기, Row dominance, Column dominance에 관여하는 것은 다음과 같다.

1. Minterm 의 번호를 나타내는 정수형 변수 number
2. 자신을 커버하는 PI 목록을 원소로 가진 리스트 :

```
ArrayList<Integer> coveredBy = new ArrayList<>();
```

3. 자신이 EPI에 속한 minterm인가를 나타내는 값 :

   boolean inEPI

4. 자신이 Column dominance로 지배하는 지를 나타내는 값 :

   Boolean dominated

3. Finding EPI

```
void solution_EPI() {
    for(int i=0; i<minterm.size(); i++) {
        int count=0, pos=0;
        for(int j=0; j<pi.size(); j++) {
            if(pi.get(j).mintermList.contains(minterm.get(i).number)) {
                pos = j;
                count++;
                minterm.get(i).coveredBy.add(pi.get(j).num);
            }
        }

        if(count == 1) {
            pi.get(pos).isEPI = true;
            for(int j=0; j<minterm.size(); j++) {
                if(pi.get(pos).mintermList.contains(minterm.get(j).number))
                    minterm.get(j).inEPI = true;
            }
        }
    }
    System.out.println("After finding EPI");
}
```

Minterm 객체들이 들어있는 minterm 리스트에 대해서 Implicant 객체가 들어있는 pi 리스트를 순회한다.

만약 해당 Minterm이 Implicant의 mintermlist에 포함되면, count++을 하고, Minterm객체의 coveredBy리스트에 Implicant번호를 추가한다.

최종 count 값이 1이면, 해당 Implicant는 EPI이다. 따라서 해당 Implicant가 커버하는 모든 Minterm객체의 inEPI 값을 true로 바꾸어준다.

4. Row dominance

```
boolean row_Dominance() {
    boolean simplification = false;
    for(Implicant i : pi) {
        if(!i.dominated) {
            for(Implicant i2 : pi) {
                if(i!=i2 && i.mintermList.containsAll(i2.mintermList)) {
                    i2.dominated = true;
                    simplification = true;
                }
            }
        }
    }
    System.out.println("After Row-Dominance");
    return simplification;
}
```

Implicant 객체들을 원소로 갖는 pi리스트를 이중으로 순회한다.

만약 지배관계가 나타나면, 지배 받는 Implicant에 대하여 dominated 필드를 true로 바꾸어준다.

순회 중 dominated 필드가 true로 되어있는 경우는 순회하지 않기 때문에, interchangeable 한 PI는 여기서 하나만 빼고 제거된다.

지배 관계가 한 번이라도 나타났다면 , true를 리턴한다.

5. Column dominance

```
boolean col_Dominance() {
    boolean simplification = false;
    for(Minterm m : minterm) {
        for(Minterm m2 : minterm) {
            if(m!=m2 && m.coveredBy.containsAll(m2.coveredBy)) {
                if(m.coveredBy.retainAll(m2.coveredBy) || m2.dominate==false)
                m.dominate = true;
                simplification = true;
            }
        }
    }
    System.out.println("After Column-Dominance");
    return simplification;
}
```

Minterm 객체를 원소로 갖는 minterm 리스트를 이중으로 순회한다.

만약 지배관계가 나타나면, 지배하는 Minterm에 대하여 dominate 필드를 true로 바꾸어준다. 이때, 서로가 서로를 지배하는 관계가 나타날 때는 한쪽만 적용한다.

지배 관계가 한 번이라도 나타났다면, true를 리턴한다.

6. Remove

```
void remove() {
    pi.removeIf(Implicant -> Implicant.isEPI==true);
    minterm.removeIf(Minterm -> Minterm.inEPI==true);
    minterm.removeIf(Minterm -> Minterm.dominate==true);
    for(Minterm m : minterm) {
        m.coveredBy.removeIf(n -> pi_copy.get(n).dominated);
    }

    // col dominance를 통해 쓸모없게 된 PI 삭제
    loop :
    for(Implicant i : pi) {
        for(Minterm m : minterm) {
            if(i.mintermList.contains(m.number))
            continue loop;
        }
        i.dominated = true;
    }

    pi.removeIf(Implicant -> Implicant.dominated==true);
}
```

Implicant 객체 중 EPI인 것을 제거한다.

Minterm 객체 중 EPI로부터 커버되는 것(inEPI == true)을, 제거한다.

Minterm 객체 중 다른 Minterm객체를 지배하는 것(dominate == true)을 제거한다.

Minterm 객체 중 Row dominance로 제거된 PI로부터 커버된 것을 제거한다.

Implicant 객체 중 Column dominance를 통해 자신이 커버하는 모든 Minterm이 제거된 경우의 것을 제거한다.

Implicant 객체 중 다른 Implicant 객체로부터 지배당하는 것(dominated == true)을 제거한다.

7. 테스트케이스

1번 테스트 케이스 :

```
int[] minterm1 = {4, 11, 0, 1, 2, 6, 7, 8, 9, 10, 11, 13, 15}; //rd, cd
```

실행 결과 :

```
          0   1      2      6      7      8      9      10     11     13     15
P1(0-10) |             v      v
P2(011-) |                    v      v
P3(-111) |                           v                                      v
P4(-00-) | v    v                                v      v
P5(-0-0) | v           v                        v             v
P6(10--) |                                      v      v      v      v
P7(1--1) |                                             v             v      v      v
Cover :

After finding EPI
          2    6      7      10
P1(0-10) | v    v
P2(011-) |      v      v
P3(-111) |             v
P5(-0-0) | v                  v
P6(10--) |                    v
Cover : P4(-00-)  P7(1--1)

After Row-Dominance
          2    6      7      10
P1(0-10) | v    v
P2(011-) |      v      v
P5(-0-0) | v                  v
Cover : P4(-00-)  P7(1--1)

After Column-Dominance
          7    10
P2(011-) | v
P5(-0-0) |      v
Cover : P4(-00-)  P7(1--1)

After finding EPI
*** All minterms have been covered ***
Cover : P2(011-)  P4(-00-)  P5(-0-0)  P7(1--1)
```

Row_dominance에서 나타난 지배관계는 P2→P3, P5→P6 이므로 P3와 P6가 제거된다.

Col-dominance에서 나타난 지배관계는 6→7, 2→10 이므로 6과 2가 제거된다.

최종결과 :

```
After finding EPI
*** All minterms have been covered ***
Cover : P2(011-)  P4(-00-)  P5(-0-0)  P7(1--1)
```

2번 테스트 케이스 :

```
int[] minterm2 = {4, 11, 0, 1, 2, 5, 4, 8, 9, 10, 11, 14, 15}; //interchangeable
```

실행 결과 :

```
          0   1      2      5      4      8      9      10     11     14     15
P1(0-0-) | v   v             v      v
P2(-00-) | v   v                           v      v
P3(-0-0) | v          v                  v             v
P4(10--) |                                v      v      v      v
P5(1-1-) |                                              v      v      v      v
Cover :

After finding EPI
          9
P2(-00-) | v
P4(10--) | v
Cover : P1(0-0-)  P3(-0-0)  P5(1-1-)

After Row-Dominance
          9
P2(-00-) | v
Cover : P1(0-0-)  P3(-0-0)  P5(1-1-)

After Column-Dominance
          9
P2(-00-) | v
Cover : P1(0-0-)  P3(-0-0)  P5(1-1-)

After finding EPI
*** All minterms have been covered ***
Cover : P1(0-0-)  P2(-00-)  P3(-0-0)  P5(1-1-)
```

EPI를 테이블에서 제거하고 난 뒤에 P2와 P4가 interchangeable한 관계이다.

Row-dominance에서, interchangeable한 PI들은 하나만 남기고 삭제된다.

최종 결과 :

```
After finding EPI
*** All minterms have been covered ***
Cover : P1(0-0-)  P2(-00-)  P3(-0-0)  P5(1-1-)
```

3번 테스트 케이스 :

```
int[] minterm3 = {3, 6, 0, 2, 3, 4, 5, 7}; //Petrick's Method
```

실행 결과 :

```
          0    2       3       4       5       7
P1(0-0) | v    v
P2(-00) | v                    v
P3(01-) |      v       v
P4(10-) |                      v       v
P5(-11) |              v                       v
P6(1-1) |                              v       v
Cover :

After finding EPI
          0    2       3       4       5       7
P1(0-0) | v    v
P2(-00) | v                    v
P3(01-) |      v       v
P4(10-) |                      v       v
P5(-11) |              v                       v
P6(1-1) |                              v       v
Cover :

After Row-Dominance
          0    2       3       4       5       7
P1(0-0) | v    v
P2(-00) | v                    v
P3(01-) |      v       v
P4(10-) |                      v       v
P5(-11) |              v                       v
P6(1-1) |                              v       v
Cover :

After Column-Dominance
          0    2       3       4       5       7
P1(0-0) | v    v
P2(-00) | v                    v
P3(01-) |      v       v
P4(10-) |                      v       v
P5(-11) |              v                       v
P6(1-1) |                              v       v
Cover :

*** Try Petrick's Method ***
```

NEPI가 아직 남아있는 상태에서 Row-dominance와 Col-dominance를 실행한 결과 변화가 없었으므로, 이 테스트 케이스는 Petrick's Method를 필요로 한다.

최종 결과 :

```
*** Try Petrick's Method ***
```