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.

Row dominance가 한번이라도 진행됐으면, 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, 지배 하는 Minterm에 대하여 dominate 필드를 true로 바꾸어준다. 이때, 서로가 서로를 지배하는 관계가 나타날 때는 한쪽만 적용한다.

Column dominance가 한번이라도 진행됐으면, 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 커버된다면(inEPI), 제거한다.

Minterm 객체 중 지배하는 쪽이면, 제거한다.

Minterm 객체 중 Row dominance로 제거된 PI로부터 커버됐다면, 제거한다.

Implicant 객체 중 Column dominance를 통해 자신이 커버하는 모든 Minterm이 제거됐다면, 제거한다.

Implicant 객체 중 지배당하는 쪽이면, 제거한다.