# 다중 조건 커버리지 기반의 논리식 동등성 검사기 개발

## (Development of a Logic Expression Equivalence Checker Based on Multiple Condition Coverage)

김예원*, 이로빈*, 남영호*
(Yewon Kim, Robin Lee, Youngho Nam)

요 약

MC/DC(Modified Condition/Decision Coverage)는 각 조건이 결정문 결과에 미치는 독립적 영향을 입증하도록 요구함으로써, MCC(Multiple Condition Coverage)대비 적은 테스트 케이스로 코드의 논리적 오류를 검출하는데 효율적인 코드 커버리지 기준이다. 안전 필수 시스템 검증에 널리 사용되는 MC/DC는 테스트 케이스 생성을 위한 다양한 연구가 진행되어 왔으나, 이들은 공통적으로 논리식 전처리 과정의 신뢰성이 보장된다는 가정하에 알고리즘의 효율성 향상에만 초점을 맞추어 왔다. 논리식의 전처리 과정의 동등성을 검증하지 않으면 원본 논리식과 다른 논리식을 테스트할 수 있는 잠재적인 위험을 내포하고 있다. 이러한 논리식 전처리 과정의 신뢰성 검증 문제를 해결하고자, 본 연구는 MCC 기반의 논리식 동등성 검사기를 개발한다. 개발한 논리적 동등성 검사기는 논리식의 각 변환 단계가 이전 단계와의 동등성을 유지하는지 순차적으로 검증하며, 불일치가 발견된 최초의 단계와 해당 반례를 제공하여 오류의 원인을 신속하게 파악하도록 돕는다. 이는 MCC를 만족하는 모든 입력 조합, 즉 전체 진리표를 생성하여 두 논리식의 결과를 비교하는 방식으로 이루어진다. 본 연구는 논리식 전처리 단계의 신뢰성 문제를 해결함으로써, MC/DC 연구 프로세스의 결과 신뢰도를 전반적으로 개선하는 데 기여할 것이다.

■ 중심어 : 소프트웨어 테스트 ; 논리식 ; 논리적 동등성 ; 수정 조건 결정 커버리지 ; 다중 조건 커버리지

**Abstract**

Modified Condition/Decision Coverage(MC/DC) is a widely adopted criterion in safety-critical systems, offering a more efficient alternative to Multiple Condition Coverage(MCC). However, existing research on MC/DC test generation has primarily prioritized algorithmic efficiency, often overlooking the reliability of the prerequisite boolean preprocessing step. This implicit assumption of reliability poses a significant risk, as errors during preprocessing can result in the testing of logically non-equivalent expressions. To ensure the integrity of the preprocessing phase, this paper proposes an MCC-based logical equivalence checker. This tool verifies equivalence sequentially at each step by comparing outcomes across all MCC combinations, thereby covering the complete truth table. If a discrepancy is detected, the tool identifies the initial failing stage and provides a corresponding counterexample. By guaranteeing the correctness of the logical expression preprocessing, this research contributes to enhancing the overall reliability of MC/DC testing.

■ keywords : Software test ; Boolean expression ; Logical equivalence; MC/DC ; MCC

## Ⅰ. 서 론

항공기, 자동차 등 안전 필수 시스템의 복잡성이 증가함에 따라, 소프트웨어의 안전성과 신뢰성 확보가 중요한 과제로 부각되고 있다. 이와 관련해 국제 안전 표준인 DO-178C(항공), ISO 26262(자동차)에서는 소프트웨어의 매우 엄격한 테스트 및 검증 절차를 요구하며, 특히 MC/DC (Modified Condition/Decision Coverage)를 준수

* 정회원, 경상국립대학교 컴퓨터공학부
이 연구는 2024년도 경상국립대학교 발전기금재단 재원으로 수행되었음.
접수일자 : 2025년 10월 28일
게재확정일 : 2025년 11월 21일
교신저자 : 남영호 e-mail : yhnam@gnu.ac.kr

하도록 요구한다[1-2]. 이는 MC/DC가 조건식에 포함된 모든 조건 조합을 검증하는 커버리지인 MCC(Multiple Condition Coverage) 대비 적은 수의 테스트 케이스로 높은 커버리지 값을 갖는 구조적 커버리지이기 때문이다[3-5]. MC/DC를 만족하는 테스트 케이스 생성을 목표로 한 기존 연구들은 제안하는 알고리즘 적용을 위해 전처리 단계를 요구한다. 전처리 단계는 원본 논리식에서 특정 논리식 형태로 변형을 통해 알고리즘에 적용하기 위한 과정이다. MC/DC를 만족하는 테스트 케이스를 효율적으로 생성하기 위한 기존 연구[6-15]들은 전처리된 논리식이 원본 논리식과 논리적으로 동등한지 확인 과정이 명확히 드러나지 않는다. 논리식을 변형 후 동등성 검증을 하지 않는 경우, 원본 논리식과 다른 논리식으로 테스트할 수 있는 잠재적 위험을 내포하고 있다.

이에 따라 전처리 과정에서 논리식의 동등성 보장을 위해, MCC를 활용한 논리식 동등성 검사기를 제안한다. 논리적 동등성 검사기는 각 변환 단계가 이전 단계와 동등성을 유지하는지 순차적으로 검증하며, 논리식의 전체 진리표를 생성하고 그 결과를 비교하여 논리적 동등성 여부를 판정한다. 또한 비동등 시 반례를 시각적으로 제시함으로써, 논리식 전처리 과정에서 발생할 수 있는 동등성 위배를 직관적으로 확인할 수 있다. 이를 통해 본 연구는 MC/DC 테스트 케이스 생성 연구의 신뢰성에 기여하는 것을 목표로 한다.

본 논문의 구성은 다음과 같다. Ⅱ장에서는 관련 연구를 살펴보고, Ⅲ장에서는 논리식 동등성 검사기의 설계 및 구현을 기술한다. Ⅳ장에서는 실험 설계 및 결과를 제시하고, Ⅴ장에서는 결론 및 한계점을 제시한다.

## Ⅱ. 관련 연구

MC/DC를 만족하기 위해 테스트 케이스를 생성하는 대다수의 연구들[6-15]은 입력된 원본 논리식을 적합한 형태로 변형하여 처리한다.

'Robin's Rule'[6]은 SBEs(Singular Boolean Expressions)에 대해 $n$개의 조건에 대하여 $n+1$개의 테스트 케이스를 생성하는 알고리즘이다. 이 알고리즘을 적용하기 위해서는 원본 논리식의 NOT 연산의 정규화와 구조적 재정렬과 같은 전처리가 선행되어야 한다. 이 연구에서는 논리식의 동등성 검증 절차는 별도로 제시되지 않았으며, 동등성이 보장되지 않는 경우 원본과 다른 논리식의 테스트 케이스를 생성하는 문제가 발생할 수 있다.

BDD(Binary Decision Diagram)를 활용하는 연구[7]에서는 원본 논리식을 BDD 구조로 변환한 뒤, MC/DC를 만족하는 테스트 케이스를 생성한다. BDD는 논리식을 압축적으로 표현하여 지수적 복잡도를 줄일 수 있다. 이러한 구조를 바탕으로, 독립성 조건을 만족하는 경로를 탐색하여 테스트 케이스를 도출한다. 위 접근법 역시 BDD로 변환된 결과물이 원본 논리식과 논리적으로 동등하다는 점을 전제로 한다.

SAT(Satisfiability)/SMT(Satisfiability Modulo Theories) Solver 기반 연구[8-12]는 원본 논리식을 CNF(Conjunctive Normal Form)와 같은 정규 형식으로 변환한 뒤, Solver가 이를 만족하는 입력 조합을 탐색하여 MC/DC를 만족하는 테스트 케이스를 생성한다. Kitamura[8]와 Yang[9] 등은 SAT 기반 접근을 통해 최소 테스트 집합을 산출하였다. 또한 SMT 및 BMC(Bounded Model Checking) 기법을 활용하는 방법이 제안되었다[10-12]. 이들 연구는 Solver 기반 효율성에 초점을 두고 있으나, 전처리 된 논리식의 동등성 검증은 다루지 않았다.

이외에도 Ufuktepe[13-14] 등은 시스템 요구사항을 부울 논리 기반의 원인 결과 그래프 형태로 전처리하여 테스트 케이스를 생성하였다. Awedikian[15]은 MC/DC 테스트 케이스 생성을 위한 전처리로, 소스 코드를 AST(Abstract Syntax Tree)로 파싱한 뒤 술어의 구조를

ADT(Abstract Decision Tree) 형태로 간결하게 변환하는 방식으로 전처리하였다.

요약하면, 기존 연구들은 MC/DC를 만족하는 테스트 케이스 생성을 위해 논리식을 다양한 형태로 변환하는 전처리 단계를 수행하지만, 논리식의 변환이 원본 논리식과 논리적으로 동등한지 검증하는 절차나 도구는 제시하지 않는다.

따라서 본 연구는 MCC 기반 논리식 동등성 검사기를 개발하여 원본 논리식과 변환된 논리식의 동등성을 보장하고자 한다. 논리식 동등성 검사기는 두 논리식의 전체 진리표를 생성 후 비교하여 동등성을 판정하고, 불일치 시 반례를 시각적으로 제시함으로써 MC/DC를 만족하는 테스트 케이스 생성 연구에서 전처리 과정의 신뢰성을 높인다.

## Ⅲ. 논리적 동등성 검사기 설계 및 구현

본 장에서는 Ⅱ장에서 제시한 논리식 전처리 과정의 동등성 검증을 위해 MCC 기반 논리적 동등성 검사기의 설계와 구현에 대해 기술한다.

### 1. 논리적 동등성 검사기 설계

논리적 동등성 검사기의 전체 시스템 구조는 그림 1과 같다. 시스템은 비교 대상이 되는 논리식들에 포함된 모든 고유 조건의 총 개수를 $n$이라 할 때, MCC에 해당하는 전체 진리표($2^n$)를 생성하고 비교한다. 각 논리식의 결과를 비교하는 과정을 효율적으로 수행하기 위해 Initializer(초기화), Comparator(비교/검증), Reporter(결과 확인)로 3단계 컴포넌트로 구성하였다.

#### 가. Initializer 컴포넌트

Initializer 컴포넌트는 전체 동등성 검사 작업을 시작하기 위한 전처리 및 환경 설정을 담당한다. 하위단계로는 CSV Load → Output Preparation → Work Sizing를 수행한다. 'CSV Load' 단계에서는 그림 2와 같이 사용자가 검증을 원하는 '원본 논리식'(1행) 1개와 '전처리 과정의 논리식'(2-5행) $L$개로 이루어진 CSV 파일을 로드한다.

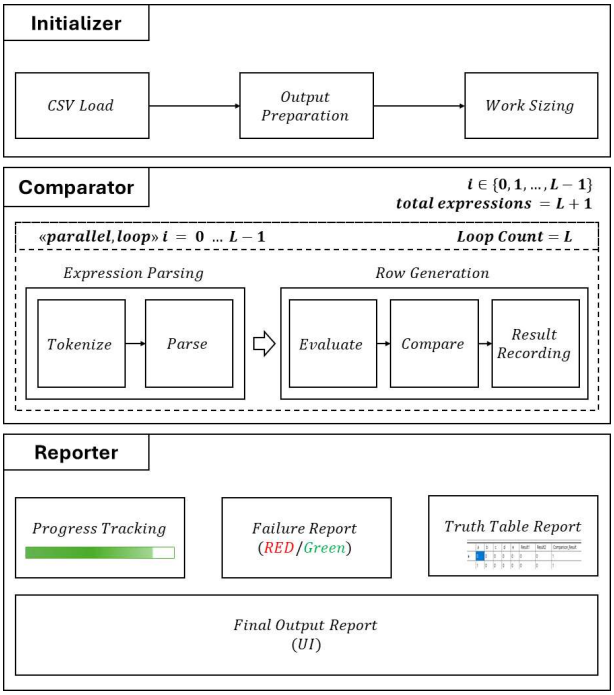


그림 1. 논리식 동등성 검사기 시스템 구조

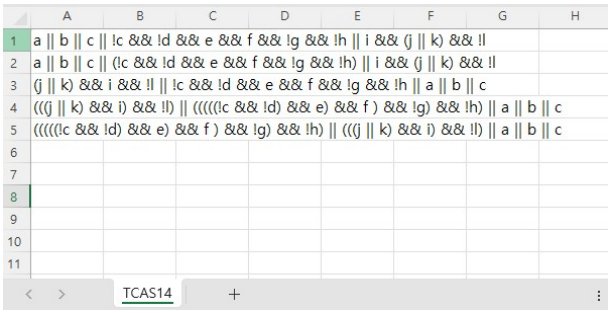


그림 2. 정상 논리식 집합 예시(TCAS14)

다음으로, 'Output Preparation' 단계에서는 Comparator가 생성할 최종 진리표가 기록될 CSV 파일을 저장할 디렉토리를 생성한다. 만약 이전 작업의 임시 파일이 있다면 초기화하여 검증 환경을 준비한다. 마지막으로, 'Work Sizing' 단계에서는 검증 대상이 되는 논리식 집합의 전체 논리식 개수($L+1$)를 파악한다. 본 검사기는 $A_i$와 $A_{i+1}$형태로 순차 비교하므로, 전체 비교 작업의 수(Loop Count)는 전처리된 논리식 개수인 $L$과 동일하다. 산출된 $L$값은 Comparator의 전체 비교 작업 수와 Reporter의 진행률 표시에 사용된다.

나. Comparator 컴포넌트

Comparator 컴포넌트는 논리적 동등성 검사기의 핵심으로, 동등성 검증 로직을 수행한다. 총 $L+1$개의 전체 논리식에 대해 인접한 두 논리식의 쌍$(A_i, A_{i+1})$을 비교하는 $L$개의 작업을 병렬 반복문으로 처리한다. 각 병렬 작업은 루프 인덱스 $i$로 구분되며, $i$의 범위는 ㉠으로 표현한다. 이와 같이 병렬로 연산함으로써 하나의 논리식 집합 전체의 검증 작업을 하나의 논리식 쌍을 검증하는 시간과 근사한 수준에서 수행이 가능하다.

$$i \in \{0, 1, \ldots, L-1\} \quad \cdots\cdots\cdots\cdots\cdots\cdots \quad ㉠$$

'Expression Parsing' 단계에서는 논리식을 토큰화하고 파싱하여 AST로 변환하는 과정이 수행된다. 다음으로 'Row Generation' 단계는 논리식 쌍$(A_i, A_{i+1})$에 대한 전체 진리표를 생성하고 동등성을 검증하는 핵심 과정이다. 논리식 쌍$(A_i, A_{i+1})$에 대한 모든 입력 조합을 생성하고, 각 조합의 진리값을 평가하여 두 논리식의 동등성을 검증한다. 먼저, 비교 대상인 두 논리식 $A_i$와 $A_{i+1}$에 포함된 모든 조건 변수들의 합집합 $V_i$는 식 ㉡과 같이 정의된다. 이에 따라 생성해야 할 전체 입력 조합의 수 $n$은 식 ㉢과 같이 결정되며, 이를 순회하기 위한 인덱스 $j$의 범위는 식 ㉣과 같다.

$$V_i = Vars(A_i) \cup Vars(A_{i+1}) \quad \cdots\cdots\cdots\cdots \quad ㉡$$

$$n = |V_i| \quad \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \quad ㉢$$

$$j \in \{0, 1, \ldots, 2^n - 1\} \quad \cdots\cdots\cdots\cdots\cdots\cdots \quad ㉣$$

Row Generation 단계의 로직을 의사코드로 표현하면 그림 3과 같다.

본 연구에서는 논리식 쌍$(A_i, A_{i+1})$의 동등성 검증을 위해, $n$개의 조건 변수에 대한 모든 입력 조합, 즉 $2^n$개의 테스트 케이스로 구성된 전체 진리표를 생성한다. 각 테스트 케이스는 이로빈[16]의 방법론을 따라, 0부터 $2^n-1$까지의 정수를 $n$비트 이진수로 표현한다. 이후 각 비트 값을 해당 조건 변수의 진리값에 매핑하여 테스트케이스를 생성한다.

**Algorithm 1:** Parallel row generation for pairwise logic equivalence.

**Input** : $[A_1, \ldots, A_{L+1}]$
**Output** : CSV file(s)
**Constant:** $B \leftarrow 4{,}096$ // Batch Size

**for** $i \leftarrow 0$ **to** $L-1$ ***(in parallel)*** **do**
  $V_i \leftarrow \mathrm{Vars}(A_i) \cup \mathrm{Vars}(A_{i+1})$
  $S \leftarrow \mathrm{SORT}(V_i)$, $n \leftarrow |V_i|$, TOTAL_ROWS $\leftarrow 2^n$
  CSV_WRITE([S, Result($A_i$), Result($A_{i+1}$), Comparison_Result])

  // predefined $B$ rows per batch
  NUM_BATCHES $\leftarrow \left\lceil \frac{\text{TOTAL\_ROWS}}{B} \right\rceil$

  **for** batch_id $\leftarrow 0$ **to** NUM_BATCHES $-1$ ***(in parallel)*** **do**
    BATCH_START $\leftarrow$ batch_id $\cdot B$
    BATCH_END $\leftarrow \min(\text{TOTAL\_ROWS}, \text{BATCH\_START} + B) - 1$

    **for** $j \leftarrow$ BATCH_START **to** BATCH_END **do**
      $x \in \{0,1\}^n$

      **for** $p \leftarrow 0$ **to** $n-1$ **do**
        $x[p] \leftarrow \mathrm{BIT}(j, n-1-p)$ // BIT → $S[p]$(sorted)

      $\mathrm{result}_i \leftarrow \mathrm{Eval}(A_i, S, x)$
      $\mathrm{result}_{i+1} \leftarrow \mathrm{Eval}(A_{i+1}, S, x)$
      $\mathrm{comparison} \leftarrow \mathbf{1}[\mathrm{result}_i = \mathrm{result}_{i+1}]$
      $\mathrm{row} \leftarrow [x[0], \ldots, x[n-1], \mathrm{result}_i, \mathrm{result}_{i+1}, \mathrm{comparison}]$

      CSV_WRITE(row)

그림 3. Row Generation 의사코드

각 논리식 쌍$(A_i, A_{i+1})$의 동등성 검증은 하나의 독립적인 태스크로 처리된다. $n$개의 조건 변수에 대한 $2^n$개의 모든 입력 조합을 검증하기 위해, 각 태스크는 이 전체 조합을 일정한 크기의 배치 단위로 분할한다. 그 후, 분할된 배치 작업들을 여러 스레드에 할당하여 병렬로 실행한다. 이렇게 병렬로 실행되는 각 배치 작업은 할당된 배치 크기 B만큼 다음의 세 단계를 반복 수행한다. 첫째, 'Evaluate(논리식 평가)' 단계에서는 이 입력 조합을 사용하여 비교 대상인 두 논리식 $A_i$와 $A_{i+1}$의 결괏값을 각각 연산한다. 둘째, 'Compare(결과 비교)' 단계에서 두 결괏값이 일치하는지 판정한다. 마지막으로, 'Result Recording' 단계에서 앞선 단계의 판정 결과를 하나의 레코드로 취합한다. 이 처리 과정을 $2^n$개의 테스트 케이스를 생성할 때까지 반복한다. $L$개의 모든 병렬 작업이 완료되면 출력 디렉토리에는 총 $L$개의 비교 결과가 생성된다.

다. Reporter 컴포넌트

Reporter 컴포넌트는 복잡한 검증 과정과 그 최종 결과를 사용자가 직관적으로 이해할 수 있도록 모니터링한다.

'Progress Tracking' 단계는 총 $L$개의 비교 작업 중 완료된 작업량을 실시간으로 시각화하여 전체 진행률을 나타낸다.

'Failure Report' 단계는 반례가 발견되는 즉시, 해당 논리식 쌍$(A_i, A_{i+1})$의 '비동등' 상태를 기록한다. 작업이 완료된 후 '테스트 대상 논리식 선택' 메뉴에서 동등하지 않은 논리식 쌍은 적색으로 표시된다.

'Truth Table Report' 단계는 개별 비교 결과를 표 형태로 제공한다. 사용자는 '테스트 대상 논리식 선택' 메뉴에서 특정 비교 쌍$(A_2, A_3)$을 선택하여 전체 진리표와 반례 값을 확인할 수 있다.

'Final Output Report(UI)' 단계는 모든 비교가 완료된 후 'Failure Report'의 기록을 바탕으로 각 논리식 쌍의 동등성 여부를 최종 판정한다. 그 결과를 '동등' / '비동등'으로 명확하게 요약하여 제시함으로써 모든 검증 절차를 마무리한다.

## 2. 논리적 동등성 검사기 구현

설계를 기반으로 구현한 논리적 동등성 검사기는 그림 4와 같다. 논리식 데이터 집합 중 하나의 논리식 집합 CSV 파일을 업로드하여 '비교 시작'을 클릭하여 수행하면 자동으로 논리식 집합 내 논리식 쌍을 비교하고 결과를 나타내며, 반례가 되는 테스트 케이스를 통해 동등/비동등을 확인할 수 있다.

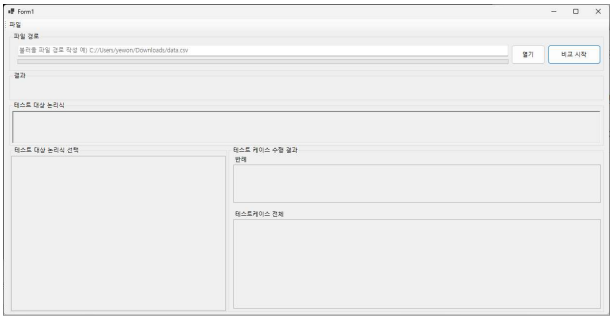

그림 4. 구현된 논리식 동등성 검사기

그림 5는 검사 대상인 논리식 데이터 집합을 선택하는 사례이다. 선택한 논리식 데이터 집합은 그림 6과 같이 검사기의 왼쪽 아래에 논리식 데이터 집합이 원본 논리식과 전처리된 논리식의 쌍으로 표시된다. 선택된 논리식 쌍은 파란색 텍스트로 표시되며 선택된 쌍은 '테스트 대상 논리식' 상자에서 논리식 데이터를 확인할 수 있다.

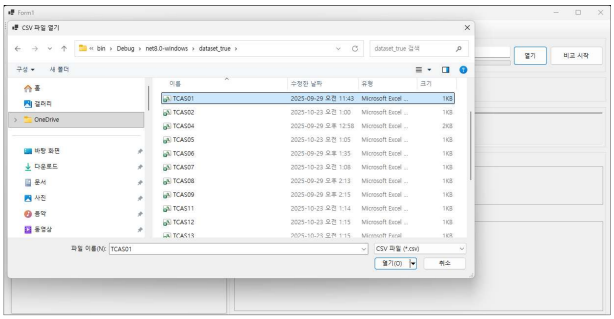

그림 5. 논리식 집합 CSV 파일 로드

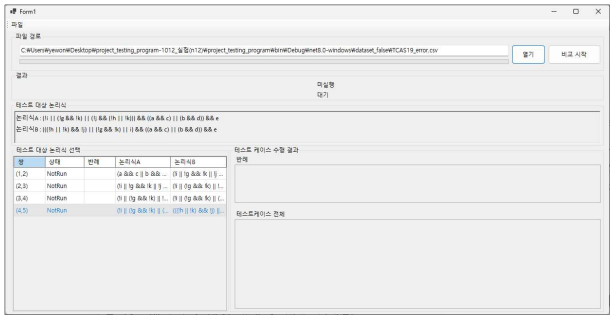

그림 6. 논리식 집합 CSV 파일을 불러온 실행 화면

## 3. 논리식 동등성 비교 기능

그림 7은 논리식 동등성 비교 기능의 구현 로직을 나타낸 것이다. 이 로직은 입력으로 주어진 논리식 집합$[A_1, A_2, \ldots, A_{L+1}]$에 대해, 인접한 두 논리식 쌍$(A_i, A_{i+1})$을 병렬로 처리하여 동등성을 평가한다. 각 쌍은 먼저 Tokenizer을 통해 식을 토큰 단위로 분리하고, Parser를 거쳐 AST로 변환된다. 다음으로 Evaluator가 논리식 쌍의 모든 조건 변수의 합집합 $X$에 대한 두 식의 평가 결과를 산출한다. 이후 Compare 모듈이 두 결과의 일치 여부를 판단한다. 모든 평가 결과는 [*Result1, Result2, Comparison_Result*] 형식으로 각 논리식 쌍별로 CSV 파일(output{i}.csv)에 저장된다.

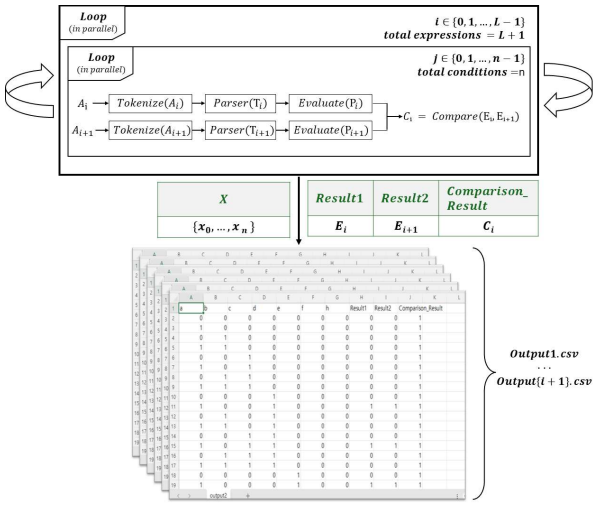


그림 7. 논리식 동등성 검사기 비교 검증 구현 로직

## Ⅳ. 실 험

본 실험은 MCC 기반 논리식 동등성 검사기가 정상적인 논리식 쌍을 올바르게 '동등'으로 판정하고, 의도적으로 오류가 주입된 논리식 쌍을 정확히 '비동등'으로 탐지하며 유효한 반례를 제시하는지 평가한다.

### 1. 실험 환경

본 실험은 표 1의 환경에서 수행하였다. 본 연구에서 개발한 '논리식 동등성 검사기'는 Visual Studio 2022와 .NET 6.0 SDK를 사용하여 WinForms GUI로 구현하였다.

표 1. 실험 환경

| 구분 | 사양 |
|---|---|
| CPU | Intel Core i7-12700K @ 5.0 GHz |
| Memory | 16 GB DDR5 |
| OS/.NET | Windows 11 / 6.0 SDK |
| IDE | Visual Studio 2022 |

### 2. 실험 데이터 셋 구성

개발된 논리식 동등성 검사기의 논리식 동등성 평가를 검증하기 위해, 실제 항공기 충돌 방지 시스템인 TCAS-II(Traffic Collision Avoidance System II)의 논리식을 기반으로 표 2와 같이 실험 데이터 집합을 구축하였다[17]. TCAS-II 논리식은 20개 조건식 중에 음영으로 표현된 3, 10, 15, 19번 행을 선택하여, 해당 논리식 데이터 집합에 오류가 있는 데이터를 임의로 삽입하였다.

표 2. TCAS-II 데이터 집합 구성

| No | TCAS II-Specipication(원본 논리식) |
|---|---|
| 1 | !(a && b) && (d && !e && !f \|\| !d && e && !f \|\| !d && e && !f) && (a && c && (d \|\| e) && h \|\| a && (d \|\| e) && !h \|\| b && (e \|\| f)) |
| 2 | !(a && b) && (c && !d && !e \|\| !f && g && !h \|\| !i && !j && !k) && (l && m && (n \|\| o) && p \|\| q && (r \|\| s) && !t \|\| u && (v \|\| w)) |
| 3 | (a && ((c \|\| d \|\|e) && g \|\| a && f \|\| c && (f \|\| g \|\| h \|\| i)) \|\| (a \|\| b) && (c \|\| d \|\| e) && i) && !(a && b) && !(c && d) && !(c && e) && !(d && e) && !(f && g) && !(f && h) && !(f && i) && !(g && h) && !(h && i) |
| 4 | (a && (d \|\| e \|\| d && e && !(f && g && h && i \|\| g && h && i) && !(f && g && l && k \|\| g && i && k)) \|\| !(f && g && h && i \|\| g && h && i) && !(f && g && l && k \|\| g && i && k) && (b \|\| c && m \|\| f)) && (a && b && c \|\| a && b && c && a && b && c) |
| 5 | a && (!b \|\| !c) && d \|\| e |
| 6 | a && (!b \|\| !c \|\| b && c && !(!f && g && h && !i \|\| !g && h && i) && !(!f && g && l && k \|\| !g && !i && k)) \|\| f |
| 7 | (!a && b \|\| a && !b) && !(c && d) && (f && !g && !h \|\| !f && g && !h \|\| !f && !g && !h) && !(j && k) && ((a && c \|\| b && d) && e && (f \|\| (i && (g && j \|\| h && k)))) |
| 8 | (!a && b \|\| a && !b) && !(c && d) && !(g && h) && !(j && k) && ((a && c \|\| b && d) && e && (!i \|\| !g && !k \|\| !j && (!h \|\| !k))) |
| 9 | (!a && b \|\| a && !b) && !(c && d) && !(g && h) && ((a && c \|\| b && d) && e && (f && g \|\| !f && h)) |
| 10 | !(c && d) && (!e && f && !g && !a && (b && c \|\| !b && d)) |
| 11 | a && !b && !c && d && !e && f && (g \|\| !g && (h \|\| i)) && !(j && k \|\| !j && l \|\| m) |
| 12 | a && !b && !c && !((f && (g \|\| !g && (h \|\| i))) \|\| f && (g \|\| !g && (h \|\| i)) && !d && !e) && !(j && k \|\| !j && l && !m) |
| 13 | a && !b && !c && (f && (g \|\| !g && (h \|\| i)) && (!e && !n \|\| d ) \|\| !n) && (j && k \|\| !j && l && !m) |
| 14 | a \|\| b \|\| c \|\| !c && !d && e && f && !g && !h \|\| i && (j \|\| k) && !l |
| 15 | a && c && (d \|\| e) && h \|\| a && (d \|\| e) && !h \|\| b && (e \|\| f) |
| 16 | a && ((c \|\| d \|\| e) && g \|\| a && f \|\| c && (f \|\| g \|\| h \|\| i)) \|\| (a \|\| b) && (c \|\| d \|\| e) && i |
| 17 | a && (!d \|\| !e \|\| d && e && !(!f && g && h && !i \|\| !g && h && i) && !(!f && g && l && k \|\| !g && !i && k) ) \|\| !(!f && g && h && !i \|\| !g && h && i) && !(!f && g && l && k \|\| !g && !i && k) && (b \|\| c && !m \|\| f) |
| 18 | (a && c \|\| b && d) && e && (f \|\| (i && (g && j \|\| h && k))) |
| 19 | (a && c \|\| b && d) && e && (!i \|\| !g && !k \|\| !j && (!h \|\| !k)) |
| 20 | (a && c \|\| b && d) && e && (f && g \|\| !f && h) |

본 연구에서 사용한 정상 및 오류 논리식 데이터 집합은 모두 4단계의 순차적 변형을 일괄 적용하여, 0번 행에 원본 조건식이, 1-4번 행에 순차적으로 변형된 논리식이 포함된 구조를 갖는다. 이 중 표 3의 오류 논리식 데이터 집합은 표 2의 3, 10, 15, 19번 TCAS-II 원본 조건식을 기반으로 생성하였다. 오류 유형은 Kobayashi[18]의 변수 부정 오류(VNF; Variable Negation

Fault), 연산자 참조 오류(ORF; Operator Reference Fault), 연관적 이동 오류(ASF; Associative Shift Fault)로 분류하였다. VNF는 TCAS03, TACS19 ORF는 TCAS10, TCAS15 ASR은 TCAS10 조건식에 각각 반영하였다.

표 3. TCAS-II 오류가 있는 논리식 데이터 집합

| TCAS03_error.csv | | |
|---|---|---|
| 원본 논리식 | | (a && ((c \|\| d \|\|e) && g \|\| a && f \|\| c && (f \|\| g \|\| h \|\| i)) \|\| (a \|\| b) && (c \|\| d \|\| e) && i) && !(a && b) && !(c && d) && !(c && e) && !(d && e) && !(f && g) && !(f && h) && !(f && i) && !(g && h) && !(h && i) |
| 변환 단계 | 1단계 (VNF) | (a && ((c \|\| d \|\|e) && g \|\| a && f \|\| c && (f \|\| g \|\| h \|\| i)) \|\| (a \|\| b) && (c \|\| d \|\| e) && i) && (!a \|\| !b) && (!c \|\| !d) && (!c \|\| !e) && (!d \|\| !e) && (!f \|\| !g) && (!f \|\| !h) && (!f \|\| !i) && (!g \|\| !h) && **(!h && !i)** |
| | 2단계 | ((a && ((((c \|\| d) \|\|e) && g) \|\| (a && f) \|\| (c && (((f \|\| g) \|\| h)\|\| i)))) \|\| (((a \|\| b) && ((c \|\| d) \|\| e)) && i)) && (!a \|\| !b) && (!c \|\| !d) && (!c \|\| !e) && (!d \|\| !e) && (!f \|\| !g) && (!f \|\| !h) && (!f \|\| !i) && (!g \|\| !h) && (!h && !i) |
| | 3단계 | (((((((f \|\| g) \|\| h)\|\| i) && c ) \|\| (((c \|\| d) \|\| e) && g) \|\| (a && f)) && a ) \|\| ((((c \|\| d) \|\| e) && (a \|\| b) ) && i)) && (!a \|\| !b) && (!c \|\| !d) && (!c \|\| !e) && (!d \|\| !e) && (!f \|\| !g) && (!f \|\| !h) && (!f \|\| !i) && (!g \|\| !h) && (!h && !i) |
| | 4단계 | ((((((((!a \|\| !b) && (!c \|\| !d)) && (!c \|\| !e)) && (!d \|\| !e)) && (!f \|\| !g)) && (!f \|\| !h)) && (!f \|\| !i)) && (!g \|\| !h)) && (!h && !i)) && (((((f \|\| g \|\| h \|\| i) && c) \|\| ((c \|\| d \|\| e) && g) \|\| (a && f)) && a) \|\| (((c \|\| d \|\| e) && (a \|\| b)) && i)) |
| TCAS10_error.csv | | |
| 원본 논리식 | | !(c && d) && (!e && f && !g && !a && (b && c \|\| !b && d)) |
| 변환 단계 | 1단계 | (!c \|\| !d) && (!e && f && !g && !a && (b && c \|\| !b && d)) |
| | 2단계 (ASR) | (!e && f && !g && !a \|\| (b && c \|\| !b && d)) && **!c \|\| !d** |
| | 3단계 (ORF) | (!e && (f && (!g && (!a **&&** ((b && c) \|\| (!b && d)))))) && !c \|\| !d |
| | 4단계 | ((((((b && c) \|\| (!b && d)) && !a) && !g ) && f) && !e) && !c \|\| !d |
| TCAS15_error.csv | | |
| 원본 논리식 | | a && c && (d \|\| e) && h \|\| a && (d \|\| e) && !h \|\| b && (e \|\| f) |
| 변환 단계 | 1단계 (ORF) | a && c && (d \|\| e) && h \|\| a && (d \|\| e) && !h \|\| b && (e **&&** f) |
| | 2단계 | (a && c && (d \|\| e) && h) \|\| (a && (d \|\| e) && !h) \|\| b && (e && f) |
| | 3단계 | (d \|\| e) && c && a && h \|\| (d \|\| e) && a && !h \|\| (e && f) && b |
| | 4단계 | ((((d \|\| e) && c) && a) && h) \|\| (((d \|\| e) && a) && !h) \|\| ((e && f) && b) |

| TCAS19_error.csv | | |
|---|---|---|
| 원본 논리식 | | (a && c \|\| b && d) && e && (!i \|\| !g && !k \|\| !j && (!h \|\| !k)) |
| 변환 단계 | 1단계 | (!i \|\| !g && !k \|\| !j && (!h \|\| !k)) && (a && c \|\| b && d) && e |
| | 2단계 | (!i \|\| (!g && !k) \|\| !j && (!h \|\| !k)) && (a && c \|\| b && d) && e |
| | 3단계 | (!i \|\| (!g && !k) \|\| (!j && (!h \|\| !k))) && ((a && c) \|\| (b && d)) && e |
| | 4단계 (VNF) | (((!h \|\| !k) && !j) \|\| (!g && !k) \|\| **i**) && ((a && c) \|\| (b && d)) && e |

## 3. 실험 결과

본 실험은 개발한 논리식 동등성 검사기의 성능과 신뢰성을 종합적으로 평가하기 위해, 표 4와 같이 총 20개의 논리식 집합(정상 16개, 오류 4개)을 대상으로 수행되었다. 논리식 동등성 검사기는 TCAS-II의 실험 데이터에서 정상과 오류가 있는 논리식 쌍에 대해 100%의 판별 정확도를 달성하였으며, 오류가 삽입된 논리식 집합에서도 오류가 발생한 변환 단계를 모두 정확히 식별하였다.

표 4. 전체 데이터 집합 검증 결과 요약

| 구분 | | 판별 성공률 (%) | 검증 시간 (ms) | 논리식 내 조건의 수 (n) |
|---|---|---|---|---|
| 정상 논리식 집합 | 1 | 100 | 94.4 | 7 |
| | 2 | 100 | 10,332.5 | 23 |
| | 4 | 100 | 42.3 | 12 |
| | 5 | 100 | 9.5 | 5 |
| | 6 | 100 | 6.7 | 9 |
| | 7 | 100 | 11.3 | 11 |
| | 8 | 100 | 8.6 | 10 |
| | 9 | 100 | 6.7 | 8 |
| | 11 | 100 | 29.1 | 13 |
| | 12 | 100 | 19.8 | 13 |
| | 13 | 100 | 31.7 | 14 |
| | 14 | 100 | 8.9 | 12 |
| | 16 | 100 | 5.0 | 9 |
| | 17 | 100 | 16.9 | 12 |
| | 18 | 100 | 5.9 | 11 |
| | 20 | 100 | 4.5 | 8 |
| 오류 논리식 집합 | 3 | 100 | 5.1 | 9 |
| | 10 | 100 | 5.6 | 7 |
| | 15 | 100 | 3.8 | 7 |
| | 19 | 100 | 4.9 | 10 |

그림 8은 논리식 동등성 검사기의 실행 결과 화면으로, 오류 논리식 집합(TCAS19_error)의 검증 결과를 예시로 제시하였다. 이 예시에서는 '!'연산자 누락으로 발생한 반례 값을 통해 논리적 불일치를 발견하여 적색으로 표시되었다. 사용자가 해당 항목을 선택하면 그림 9와 같이 두 논리식의 출력이 달라지는 반례 입력과 전체 진리표를 함께 표시한다. 논리식 동등성 검사기는 단순한 불일치 검출을 넘어, 전처리 단계에서의 오류 위치와 원인을 직관적으로 파악할 수 있다.

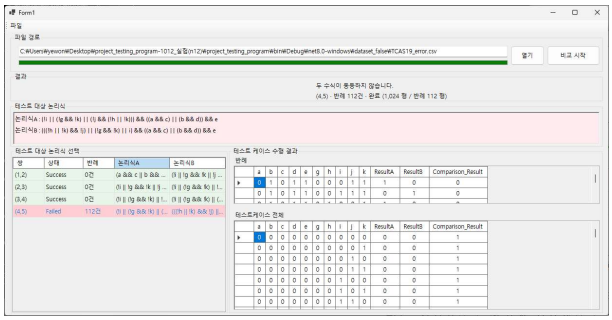

그림 8. 오류가 있는 논리식 집합 실행 결과 화면

테스트 케이스 수행 결과

반례

| | a | b | c | d | e | g | h | i | j | k | ResultA | ResultB | Comparison_Result |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ▶ | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 |

그림 9. 논리식 동등성 검사기 검증 결과 반례 테이블

## 4. 실험 결과 분석

대부분의 논리식 데이터 집합은 조건의 수가 증가할수록 검증 시간이 비례적으로 증가하는 경향을 보였다. 특히 정상 논리식 집합 2번은 10,332.5ms의 시간이 소요되어, $n$이 일정 수준 이상 커지자 시간이 지수적으로 증가함을 확인하였다. 표 2의 정상 논리식 집합 1번은 동일 변수가 여러 조건에서 반복되어 평가 및 입출력 횟수가 누적되어 검증 시간이 증가하였다. 정상 논리식 집합 9번과 정상 논리식 집합 4번은 단축 평가(Short-Circuit Evaluation)가 거의 발생하지 않는 깊은 AND 구조로 인해 검증 시간이 크게 증가하였다. 반면 정상 논리식 집합 14번, 18번과 오류 논리식 집합 19번은 논리식 구조가 단순하고 단축 평가가 자주 일어나 검증 시간이 짧게 나타났다. 또한 정상 논리식 집합 5번과 오류 논리식 집합 10번은 최상위 식의 평가 순서상 단축 평가 이점이 제한되면서 검증 시간이 길어졌다. 따라서 검증 시간은 주요 변수인 조건의 수뿐만 아니라, 논리식의 구조적 중복성 및 단축 평가 효율성과 같은 구조적 요인에 의해 영향을 받는 것으로 분석된다.

# V. 결 론

본 연구는 MC/DC 테스트 케이스 생성 과정에서 발생하는 논리식 전처리의 신뢰성 확보에 초점을 두었다. 이를 위해 MCC 기반 논리식 동등성 검사기를 개발하였다. 이 검사기는 논리식 쌍의 전체 진리표를 자동 생성하여 논리적 동등성을 검증한다. 또한, 비동등 발생 시 결과가 달라지는 반례 입력 조합을 시각적으로 제시하여 오류의 근거를 직관적으로 확인할 수 있도록 하였다. 결과적으로, 본 연구는 논리식 전처리 단계의 동등성을 자동으로 보장하는 실용적 도구를 제시함으로써 MC/DC 기반 테스트 케이스 생성 과정의 신뢰성과 검증 효율성을 향상시켰다.

다만, 조건 변수의 수가 증가함에 따라 시간 복잡도가 지수적으로 증가하는 구조적 한계는 여전히 존재한다. 향후 연구에서는 본 시스템을 MC/DC 테스트 생성 도구와 연동가능한 '전처리 모듈'로 확장하고자 한다. 구체적으로는 BDD, CNF, AST 등 논리식의 전처리 형태를 지원하여, EQ-Robin[19]과 같은 최신 연구의 동등성 검증을 내재적으로 지원하고자 한다. 또한 대규모 논리식 처리에 대한 연산 효율성을 제고할 계획이다. 이를 통해 논리식 전처리 과정을 전반적으로 지원함으로써, 자동화된 MC/DC 테스트 케이스 생성 연구[20]의 신뢰성과 효율성 향상에 기여할 것이다.

# REFERENCES

[1] RTCA, "Software Considerations in Airborne

Systems and Equipment Certification," *RTCA DO-178C*, Dec. 2011.
[2] ISO, "Road Vehicles – Functional Safety – Part 6: Product Development at the Software Level," *ISO 26262-6:2018,* 2nd ed., 2018.
[3] K. J. Hayhurst, S. P. Miller, J. J. Chilenski and L. K. Rierson, "A Practical Tutorial on Modified Condition/Decision Coverage," *NASA/TM 2001 210876*, May 2001.
[4] J. J. Chilenski, "An Investigation of Three Forms of the Modified Condition/Decision Coverage(MC/DC) Criterion," *Federal Aviation Administration Technical Report DOT/FAA/AR-01/18*, Apr. 2001.
[5] J. J. Chilenski and S. P. Miller, "Applicability of Modified Condition/Decision Coverage to Software Testing," *Software Engineering Journal*, vol. 9, no. 5, pp. 193 - 200, Sep. 1994.
[6] R. Lee and Y. Nam, "An Efficient Algorithm for Generating Minimal Unique-Cause MC/DC Test Cases for Singular Boolean Expressions," arXiv preprint arXiv:2507.14687, Jul. 2025.
[7] F. Ahishakiye, J. I. Requeno Jarabo, L. M. Kristensen, and V. Stolz, "MC/DC Test Cases Generation Based on BDDs," *Dependable Software Engineering. Theories, Tools, and Applications (SETTA 2021)*, vol. 13071, Springer, pp. 178 - 197, Cham, 2021.
[8] T. Kitamura, Q. Maissonneuve, E.-H. Choi, C. Artho, and A. Gargantini, "Optimal Test Suite Generation for Modified Condition Decision Coverage Using SAT Solving," *Computer Safety, Reliability, and Security(SAFECOMP 2018)*, vol. 11093, Springer, pp. 123 - 138, Cham, 2018.
[9] L. Yang, J. Yan, and J. Zhang, "Generating Minimal Test Set Satisfying MC/DC Criterion via SAT-Based Approach," *Proceedings of the 33rd Annual ACM Symposium on Applied Computing(SAC '18), ACM*, pp. 1899 - 1906, New York, USA, 2018.
[10] S. Godboley, J. Jaffar, R. Maghareh, and A. Dutta, "Toward Optimal MC/DC Test Case Generation," *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis(ISSTA 2021),* ACM, pp. 505 - 516, New York, USA, 2021.
[11] S. Kangoye, A. Todoskoff, M. Barreau, and P. Germanicus, "MC/DC Test Case Generation Approaches for Decisions," *Proceedings of the ASWEC 2015 24th Australasian Software Engineering Conference(ASWEC '15 Vol. II),* ACM, pp. 74 - 80, New York, USA, 2015.
[12] M. R. Golla and S. Godboley, "Automated SC-MCC Test Case Generation Using Bounded Model Checking for Safety-Critical Applications," *Expert Systems with Applications*, vol. 238, pp. 122033-122052, Mar. 2024.
[13] D. K. Ufuktepe, E. Ufuktepe, and T. Ayav, "A Metric for Measuring Test Input Generation Effectiveness of Test Generation Methods for Boolean Expressions," *Proceedings of the 2021 15th Turkish National Software Engineering Symposium (UYMS), IEEE*, pp. 1 - 6, Izmir, Turkey, 2021.
[14] D. K. Ufuktepe, "Test Case Generation from Cause - Effect Graphs," *İzmir Institute of Technology Master of Science Thesis in Computer Engineering*, Dec. 2016.
[15] Z. Awedikian, K. Ayari, and G. Antoniol, "MC/DC Automatic Test Input Data Generation," *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation(GECCO '09), ACM*, pp. 1657 - 1664, New York, USA, Jul. 2009.
[16] 이로빈, 남영호, "2진수를 활용한 MCC 테스트 케이스 생성기 설계 및 구현," *스마트미디어저널*, 제13권, 제8호, 9-15쪽, 2024년 8월
[17] E. Weyuker, T. Goradia, and A. Singh, "Automatically Generating Test Data from a Boolean Specification," *IEEE Transactions on Software Engineering, IEEE*, vol. 20, no. 5, pp. 353 - 363, May 1994.
[18] N. Kobayashi, T. Tsuchiya, and T. Kikuno, "Non-Specification-Based Approaches to Logic Testing for Software," *Information and Software Technology*, vol. 44, no. 2, pp. 113 - 121, Feb. 2002.
[19] R. Lee and Y. Nam, "EQ-Robin: Generating Multiple Minimal Unique-Cause MC/DC Test Suites," arXiv preprint arXiv:2509.26458, Sep. 2025.
[20] S. Shekhawat, A. Iqbal, U. Srinivsan, and P. Menon, "Automation of MC/DC Coverage Test Case Suite Deploying the Optimal Strategic Choice for Tool Development," *Proceedings of the ICT with Intelligent Applications, Smart Innovation, Systems and Technologies(SIST)*, vol. 248, Springer, pp. 433 - 443, Singapore, 2022.

## 저 자 소 개


**김예원(정회원)**

2023년 경상국립대학교
컴퓨터과학과 학사 졸업.
현재 경상국립대학교
컴퓨터공학과 석사과정.

<주관심분야 : 임베디드 시스템, 소프트웨어 테스트>

**이로빈(정회원)**

2014년 경상국립대학교
컴퓨터공학과 학사 졸업.
2016년 경상국립대학교
정보과학과 석사 졸업.
2025년 경상국립대학교
컴퓨터과학과 박사 졸업.

<주관심분야 : 소프트웨어공학, 소프트웨어 테스트>

**남영호(정회원)**

1989년 경상국립대학교
전자계산통계학과 학사 졸업.
1991년 중앙대학교
전자계산학과 석사 졸업.
1994년 중앙대학교 전자계학과
박사 졸업.
현재 경상국립대학교
컴퓨터공학부교수 재직.

<주관심분야 : 시스템 프로그래밍, 컴퓨터 과학 교육,
프롬프트 엔지니어링>