Intro to Software Testing

Chapter 8.2

Syntactic Logic Coverage
Disjunctive Normal Form (DNF)

Dr. Brittany Johnson-Matthews
(Dr. B for short)

https://go.gmu.edu/SWE637
Adapted from slides by Jeff Offutt and Bob Kurtz
What is DNF?

Disjunctive Normal Form (DNF) is a common representation for Boolean functions.

Slightly different notation and terminology

**Literal**: a clause or the negation of a clause: $a, \bar{a}$

**Term**: is a set of literals connected by logical and, represented by adjacency, for example:

- $a \land b$ becomes $ab$
- $\neg a \land b$ becomes $\bar{a}b$
- $\neg a \land \neg b$ becomes $\bar{a}\bar{b}$

Terms are also called **implicants**, because if a single term is true, it implies that the entire predicate is true.

**Predicate**: a set of terms connected by or, which is represented by $+$, for example:

- $a \lor b$ becomes $a + b$
**DNF Fault Classes**

There are 9 types of syntactic faults on DNF predicates; we want criteria that are guaranteed to find them.

<table>
<thead>
<tr>
<th>Fault Class</th>
<th>Intended Expression</th>
<th>Faulty Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENF: expression negation fault</td>
<td>$f = ab + c$</td>
<td>$f = \overline{ab} + c$</td>
</tr>
<tr>
<td>TNF: term negation fault</td>
<td>$f = ab + c$</td>
<td>$f = \overline{ab} + c$</td>
</tr>
<tr>
<td>TOF: term omission fault</td>
<td>$f = ab + c$</td>
<td>$f = ab$</td>
</tr>
<tr>
<td>LNF: literal negation fault</td>
<td>$f = ab + c$</td>
<td>$f = a\overline{b} + c$</td>
</tr>
<tr>
<td>LRF: literal reference fault</td>
<td>$f = ab + bcd$</td>
<td>$f = ad + bcd$</td>
</tr>
<tr>
<td>LOF: literal omission fault</td>
<td>$f = ab + c$</td>
<td>$f = a + c$</td>
</tr>
<tr>
<td>LIF: literal insertion fault</td>
<td>$f = ab + c$</td>
<td>$f = ab + bc$</td>
</tr>
<tr>
<td>ORF+: operator reference fault</td>
<td>$f = ab + c$</td>
<td>$f = abc$</td>
</tr>
<tr>
<td>ORF*: operator reference fault</td>
<td>$f = ab + c$</td>
<td>$f = a + b + c$</td>
</tr>
</tbody>
</table>
DNF Fault Class Subsumption

LIF: Literal Insertion Fault

LRF: Literal Reference Fault

LOF: Literal Omission Fault

LNF: Literal Negation Fault

ORF+: Operator Reference Fault

TOF: Term Omission Fault

TNF: Term Negation Fault

ENF: Expression Negation Fault

Together, these subsume all others

If we can find LIF and LOF faults, we will find all faults
Implicant Coverage

An obvious coverage thought is to make each implicant (term) evaluate to true. This only tests true cases for the predicate $f$, so we include DNF negation of the entire predicate $f$.

**Examples:**

$$f = ab + b\bar{c}, \quad \bar{f} = \bar{b} + \bar{a}c$$

Implicants: \{ab, b\bar{c}, \bar{b}, \bar{a}c\}

Possible test set: \{TTF, FFT\}

IC is a relatively weak criterion, not guaranteed to find any of the DNF fault classes.
Improving on Implicant Coverage

Additional definitions:

**Proper subterm:** a term with one or more clauses removed

\[ abc \] has proper subterms, \( a, b, c, ab, ac, bc \)

**Prime implicant:** an implicant such that no proper subterm is an implicant

Given \( f = ab + abc, ab \) is a prime implicant, but \( abc \) is not, because proper subterm \( ac \) is an implicant (because the predicate can be simplified to \( f = ab + ac \), and we’ll soon see how to determine that)

**Redundant implicant:** an implicant that can be removed without changing the value of the predicate

Given \( f = ab + ac + b\bar{c} \), implicant \( ab \) is redundant because the predicate can be simplified to \( ac + b\bar{c} \) (again, we’ll soon see how to determine that)
Simplifying Predicates

We can use Karnaugh maps (K-maps) to simplify DNF predicates

Given predicate \( f = ab + ac + b\overline{c} \)

Group clauses into pairs (or one pair and one single clause) and populate the possible values of the clauses.

Values use Grey code ordering (rather than binary counting) where only one truth value changes at a time across columns or down rows.

Populate the truth table where true values are listed as “t”, false values are (by convention) simply left blank.
Simplifying Predicates

We can use Karnaugh maps (K-maps) to simplify DNF predicates

Given predicate $f = ab + ac + b\bar{c}$

Group clauses into pairs (or one pair and one single clause) and populate the possible values of the clauses.

Values use Grey code ordering (rather than binary counting) where only one truth value changes at a time across columns or down rows.

Populate the truth table where true values are listed as “t”; false values are (by convention) simply left blank.
Simplifying Predicates

We can use Karnaugh maps (K-maps) to simplify DNF predicates

Given predicate $f = ab + ac + b\bar{c}$

Group clauses into pairs (or one pair and one single clause) and populate the possible values of the clauses.

Values use Grey code ordering (rather than binary counting) where only one truth value changes at a time across columns or down rows.

Populate the truth table where true values are listed as “t”; false values are (by convention) simply left blank.
Simplifying Predicates

We can use Karnaugh maps (K-maps) to simplify DNF predicates.

Given predicate \( f = ab + ac + b\bar{c} \)

Group clauses into pairs (or one pair and one single clause) and populate the possible values of the clauses.

Values use Grey code ordering (rather than binary counting) where only one truth value changes at a time across columns or down rows.

Populate the truth table where true values are listed as “t”; false values are (by convention) simply left blank.
We can use Karnaugh maps (K-maps) to simplify DNF predicates. Given predicate $f = ab + ac + b\overline{c}$, simplifies to $f = ac + b\overline{c}$. 

![Karnaugh Map](chart.png)
Simplifying Predicates

We can use Karnaugh maps (K-maps) to simplify DNF predicates.

Given predicate $f = ab + ac + b\overline{c}$

Simplifies to $f = ac + b\overline{c}$

Select maximal rectangles in the table, sized $2^m \times 2^n$ (1x1, 1x2, 2x2, 2x4, 4x4, 4x8, etc.); it's okay if they overlap.
Simplifying Predicates

We can use Karnaugh maps (K-maps) to simplify DNF predicates

Given predicate \( f = ab + ac + b\overline{c} \)
Simplifies to \( f = ac + b\overline{c} \)

Select maximal rectangles in the table, sized \( 2^m \) by \( 2^n \) (1x1, 1x2, 2x2, 2x4, 4x4, 4x8, etc.); it’s okay if they overlap.
K-Maps are Toroidal

K-Maps are a torus, not a plane

The bottom row wraps around to the top row

The right column wraps around to the left column

By Jochen Burghardt - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=28286441
K-MAPS ARE TOROIDAL

Given the predicate $f = \overline{bd}$

Draw the K-map
Given the predicate $f = bd$

Draw the K-map

These 4 true values are a single 2x2 rectangle!
Prime Implicants

Given the predicate $f = abc + ab\bar{d} + \bar{a}bcd + ab\bar{c}d + a\bar{c}d$

Draw the K-map
Prime Implicants

Given the predicate $f = abc + ab\bar{d} + \bar{a}bcd + \bar{a}\bar{b}d + \bar{a}c\bar{d}$

Draw the K-map

Not prime implicants: $ab\bar{d}$ (part of $\bar{a}d$), $\bar{a}bcd$ (part of $b\bar{c}d$), $\bar{a}c\bar{d}$ (part of $\bar{a}d$), $\bar{a}d$ (part of $\bar{a}d$).

All these have proper subterms that are implicants.

Minimal DNF representation: $f = a\bar{d} + bcd$
A **minimal DNF representation** is one with only prime, non-redundant implicants.

Not minimal: \( f = abc + ab\overline{d} + \overline{a}bcd + \overline{a}bc\overline{d} + a\overline{c}\overline{d} \)

Minimal (simplified) equivalent from previous slide: \( f = a\overline{d} + bcd \)
Determination

Given predicate $f = b + \overline{ac} + ac$, suppose we want to identify when $b$ determines $f$

Draw K-map
Determination

Given predicate \( f = b + \overline{ac} + ac \), suppose we want to identify when \( b \) determines \( f \).

Draw K-map

If two cells adjacent to the boundary have different values for \( f \), then \( b \) determines \( f \) for those two cells.

\( b \) determines \( f \) for \( ac + \overline{ac} \)
Given predicate $f = ab + bc$, suppose we want to negate $f$.

Draw the K-map for $f$.

Negate all the cells in the K-map.

Write down the result: $\overline{f} = \overline{b} + \overline{a}c$
True and False Points

Given $f = ab + cd$

**True points** are those cells in the K-map where the value of the predicate is true.

**False points** are those where the value is false.
Unique True Points

A unique true point (UTP) with respect to a given implicant is an assignment of truth values such that

- The given implicant is true
- All other implicants are false

Thus a unique true point test focuses on only one implicant.
Unique True Points (UTPs)

Given \( f = ab + cd \)

<table>
<thead>
<tr>
<th>cd</th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>t</td>
<td>t</td>
<td>t</td>
<td></td>
</tr>
<tr>
<td>01</td>
<td></td>
<td>t</td>
<td>t</td>
<td>t</td>
</tr>
<tr>
<td>11</td>
<td>t</td>
<td>t</td>
<td>t</td>
<td>t</td>
</tr>
<tr>
<td>10</td>
<td>t</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Unique true points for \( ab \)

TTFF, TTFT, TTTF

Unique true points for \( cd \)

FFTT, FTTT, TFTT

TTTT is a true point, but not a unique true point
Multiple Unique True Point Coverage

A minimal representation guarantees the existence of at least one unique true point for each implicant.

**Multiple Unique True Point Coverage (MUTP)** – Given a minimal DNF representation of a predicate \( f \), for each implicant \( i \), choose unique true points (UTPs) such that clauses not in \( i \) are true and false.
**Multiple Unique True Points**

Given \( f = ab + cd \)

Choose unique true points for each implicant such that literals not in the implicant take on values true and false.

For implicant \( ab \), choose \( \text{TTFT} \) and \( \text{TTTF} \)

For implicant \( cd \), choose \( \text{FTTT} \) and \( \text{TFTT} \)

MUTP test set:
\[ \{ \text{TTFT, TTTF, FTTT, TFTT} \} \]
Given the predicate \( f = ab + b\overline{c} \)

Implicants are \( \{ ab, bc \} \)

Both implicants are prime

Neither implicant is redundant
**MUTP Infeasibility**

Unique true points required by MUTP

**ab:** \{TTT\} causes **ab** to be true and **b\bar{c}** to be false

But there’s no way to also make clause **c** both true and false while keeping the implicants true and false as required by MUTP, so MUTP is infeasible

**b\bar{c}:** \{FTF\} causes **ab** to be false and **b\bar{c}** to be true

But there’s no way to also make clause **a** both true and false while keeping the implicants true and false as required by MUTP, so MUTP is infeasible
MUTP Fault Detection

When feasible, MUTP finds all literal insertion faults (LIFs).

Now we need a way to find all literal omission faults (LOFs) and/or operator reference faults (ORF*s).
Near False Points and CUTPNFP

A near false point (NFP) with respect to a clause \( c \) in implicant \( i \) is an assignment of truth values such that \( f \) is false, but if \( c \) is negated and all other clauses are left unchanged, then \( i \) and thus \( f \) evaluates to true.

At a near false point, \( c \) determines \( f \)

**Corresponding Unique True Point and Near False Point Pair Coverage (CUTPNFP)** – Given a minimal DNF representation of a predicate \( f \), for each clause \( c \) in each implicant \( i \), \( TR \) contains a unique true point for \( i \) and a near false point for \( c \) such that the points differ only in the truth value of \( c \).
CUTPNFP Example

Given \( f = ab + cd \)

For each literal \( c \) in each implicant \( i \), choose a unique true point for \( i \) and a near false point for \( c \) in \( i \) such that only the value of \( c \) changes.

For clause \( a \) in \( ab \), choose UTP and NFP:
- \( TTFF \) and \( FTFF \), or
- \( TTFT \) and \( FTFT \), or
- \( TTTF \) and \( FTTF \)

For clause \( b \) in \( ab \), choose UTP and NFP:
- \( TTFF \) and \( TFFF \), or
- \( TTFT \) and \( TFFT \), or
- \( TTTF \) and \( TFTF \)

We don’t have to pick the same UTP for \( a \) and \( b \), but we can to reduce test cases.
Given $f = ab + cd$

For each literal $c$ in each implicant $i$, choose a unique true point for $i$ and a near false point for $c$ in $i$ such that only the value of $c$ changes.

<table>
<thead>
<tr>
<th>ab</th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>t</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>01</td>
<td></td>
<td>t</td>
<td>t</td>
<td>t</td>
</tr>
<tr>
<td>11</td>
<td>t</td>
<td>t</td>
<td>t</td>
<td>t</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>t</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

For clause $c$ in $cd$, choose UTP and NFP
- FTTT and FFFT, or
- FTTT and FTFT, or
- TFTT and TFTF

For clause $d$ in $cd$, choose UTP and NFP
- FTTT and FFFT, or
- FTTT and FTFT, or
- TFTT and TFTF

We don’t have to pick the same UTP for $c$ and $d$, but can to reduce test cases.
CUTPNFP EXAMPLE (CONT’D)

Given \( f = ab + cd \)

For each literal \( c \) in each implicant \( i \), choose a unique true point for \( i \) and a near false point for \( c \) in \( i \) such that only the value of \( c \) changes.

<table>
<thead>
<tr>
<th></th>
<th>( ab )</th>
<th>( cd )</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>t</td>
<td>t</td>
</tr>
<tr>
<td>01</td>
<td></td>
<td>t</td>
</tr>
<tr>
<td>11</td>
<td>t</td>
<td>t</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>t</td>
</tr>
</tbody>
</table>

For clause \( a \) in \( ab \), choose UTP and NFP

- TTFT and FTFT

For clause \( b \) in \( ab \), choose UTP and NFP

- TTFT and TFFT

For clause \( c \) in \( cd \), choose UTP and NFP

- FTTT and FTFT

For clause \( d \) in \( cd \), choose UTP and NFP

- FTTT and FTTF

\( TR = \{ TTFT, FTFT, TFFT, FTTT, FTTF \} \)
CUTPNFP Fault Detection

When feasible, CUTPNFP finds all literal omission faults (LOFs).

Is there a way to increase the feasibility so that more predicates can be adequately tested?
We saw earlier that MUTP can easily be infeasible in its entirety, and the same is true of CUTPNFP.

**Multiple Near False Point Coverage (MNFP)** – Given a minimal DNF representation of a predicate $f$, for each clause $c$ in each implicant $i$, TR contains near false points for $c$ such that the clauses not in $i$ take on values true and false.
Given $f = ab + cd$

For each literal $c$ in each implicant $i$, choose near false points such that the clauses not in $i$ take on values true and false.

For clause $a$ in $ab$, choose NFP FTFT and NFP FTTF.

For $b$ in $ab$, choose TFTF and TFTF.

For $c$ in $cd$, choose FTFT and TFFT.

For $d$ in $cd$, choose FTTF and TFTF.

MNFP test set: \{ TFTF, TFFT, FTTF, TFTF \}
We can combine the previous three criteria (MUTP, CUTPNFP, and MNFP) This combination detects all fault classes even when one (or more) of the constituent criteria are infeasible

DEFINITION

**MUTP, MNFP, and CUTPNFP Coverage (MUMCUT)** – Given a minimal DNF representation of a predicate $f$, apply MUTP, CUTPNFP, and MNFP.

This combination detects all fault classes even when one (or more) of the constituent criteria are infeasible

However, this is a very expensive criterion
Minimal-MUMCUT uses feasibility analysis, and adds CUTPNFP and MNFP only when necessary.

Guarantees detection of LIF, LRF, and LOF fault types, thus covers all 9 fault types.

For each term
- MUTP Feasible?
  - YES: Test set = MUTP + MNFP
  - NO: For each literal in term
    - CUTPNFP Feasible?
      - YES: Test set = MUTP + CUTPNFP
      - NO: Test set = MUTP + MNFP