Introduction to Software Testing

Syntactic Logic Coverage (Ch. 8.2)

Software Testing & Maintenance

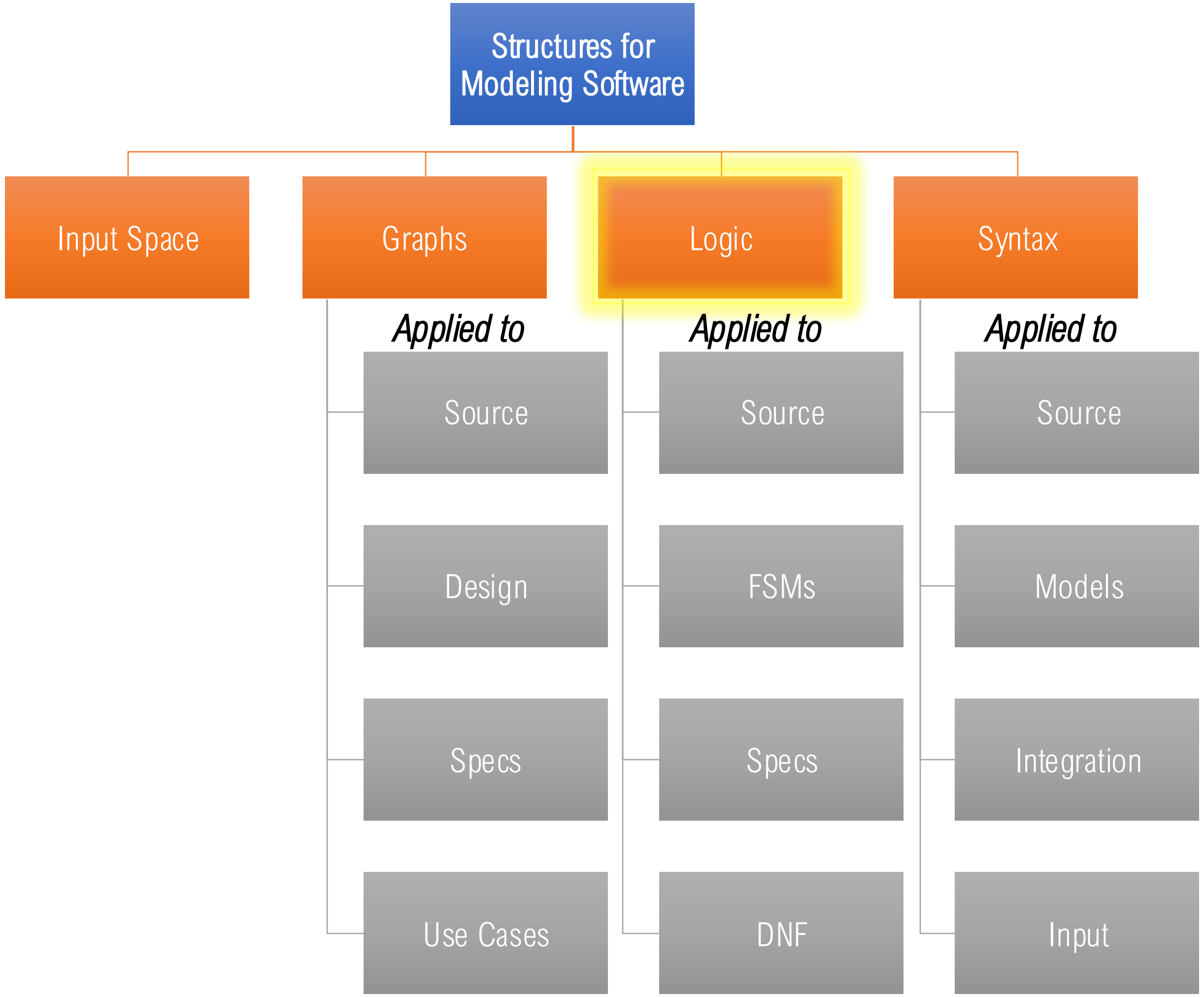
SWE 437/637

go.gmu.edu/SoftwareTestingFall24

Dr. Brittany Johnson-Matthews

(Dr. B for short)

Logic Coverage



Disjunctive Normal Form (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 operator *and* (∧), represented by adjacency.
  - $\boldsymbol{a} \wedge \boldsymbol{b}$ becomes $\boldsymbol{ab}$
  - $\neg\boldsymbol{a} \wedge \boldsymbol{b}$ becomes $\overline{\boldsymbol{a}}\boldsymbol{b}$
  - $\neg\boldsymbol{a} \wedge \neg\boldsymbol{b}$ becomes $\overline{\boldsymbol{ab}}$
- 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

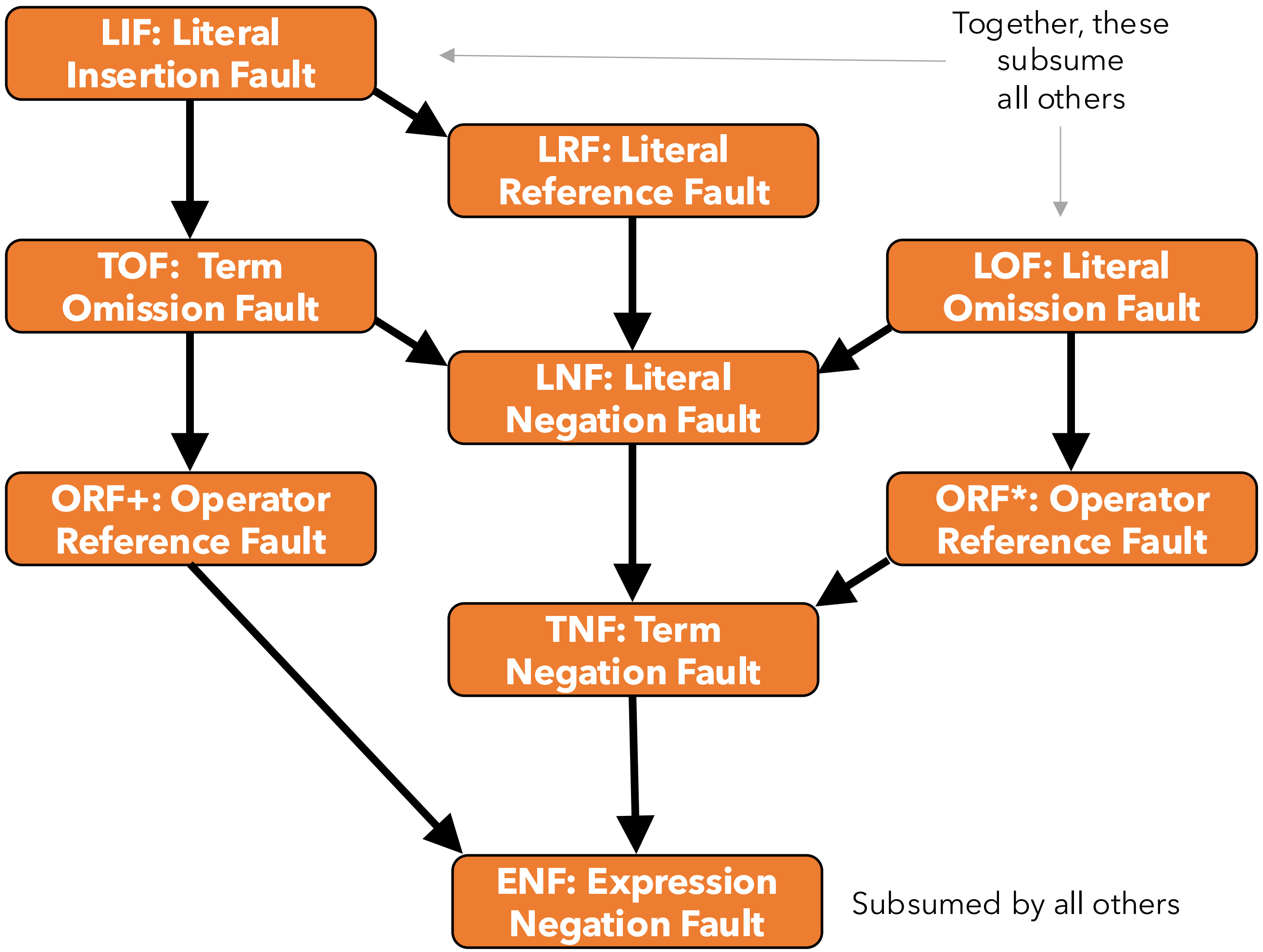
DNF Fault Classes

There are 9 types of syntactic faults on DNF predicates; we want

them.

Fault Class	Intended Expression	Faulty Expression
ENF: expression negation fault	$f = ab + c$	$f = \overline{ab + c}$
TNF: term negation fault	$f = ab + c$	$f = \overline{ab} + c$
TOF: term omission fault	$f = ab + c$	$f = ab$
LNF: literal negation fault	$f = ab + c$	$f = a\overline{b} + c$
LRF: literal reference fault	$f = ab + bcd$	$f = ad + bcd$
LOF: literal omission	$f = ab + c$	$f = a + c$

DNF Fault Class Subsumption


LIF: Literal Insertion Fault
LRF: Literal Reference Fault
TOF: Term Omission Fault
LOF: Literal Omission Fault
LNF: Literal Negation Fault
ORF+: Operator Reference Fault
ORF*: Operator Reference Fault
TNF: Term Negation Fault
ENF: Expression Negation Fault
Together, these subsume all others
Subsumed by 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 tes F negation of the entire

DEFINITION

Implicant Coverage (IC) – Given DNF representation of a predicate $\boldsymbol{f}$ and its negation $\bar{\boldsymbol{f}}$, for each implicant in $\boldsymbol{f}$ and $\bar{\boldsymbol{f}}$, ***TR*** contains the requirement that the implicant evaluate to true.

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

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

Possible test set: { TTF, FFT }

Improving on Implicant Coverage

Additional definitions:

Proper subterm: a term with one or more clauses removed

$\boldsymbol{abc}$ has proper subterms, $\boldsymbol{a},\ \boldsymbol{b},\ \boldsymbol{c}, \boldsymbol{ab}, \boldsymbol{ac}, \boldsymbol{bc}$

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

Given $\boldsymbol{f} = \boldsymbol{ab} + \boldsymbol{a\bar{b}c}, \boldsymbol{ab}$ is a prime implicant, but $\boldsymbol{a\bar{b}c}$ is not, because proper subterm $\boldsymbol{ac}$ is an implicant (because the predicate can be simplified to $\boldsymbol{f} = \boldsymbol{ab} + \boldsymbol{ac}$, 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 $\boldsymbol{f} = \boldsymbol{ab} + \boldsymbol{ac} + \boldsymbol{b\bar{c}}$, implicant $\boldsymbol{ab}$ is redundant because the

Simplifying Predicates

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

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

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

ab

c	00	01	11	10
0			t	
1			t	

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 $\boldsymbol{f = ab + ac + b\bar{\ }}$

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

ab

		00	01	11	10
c	0			t	
	1			t	t

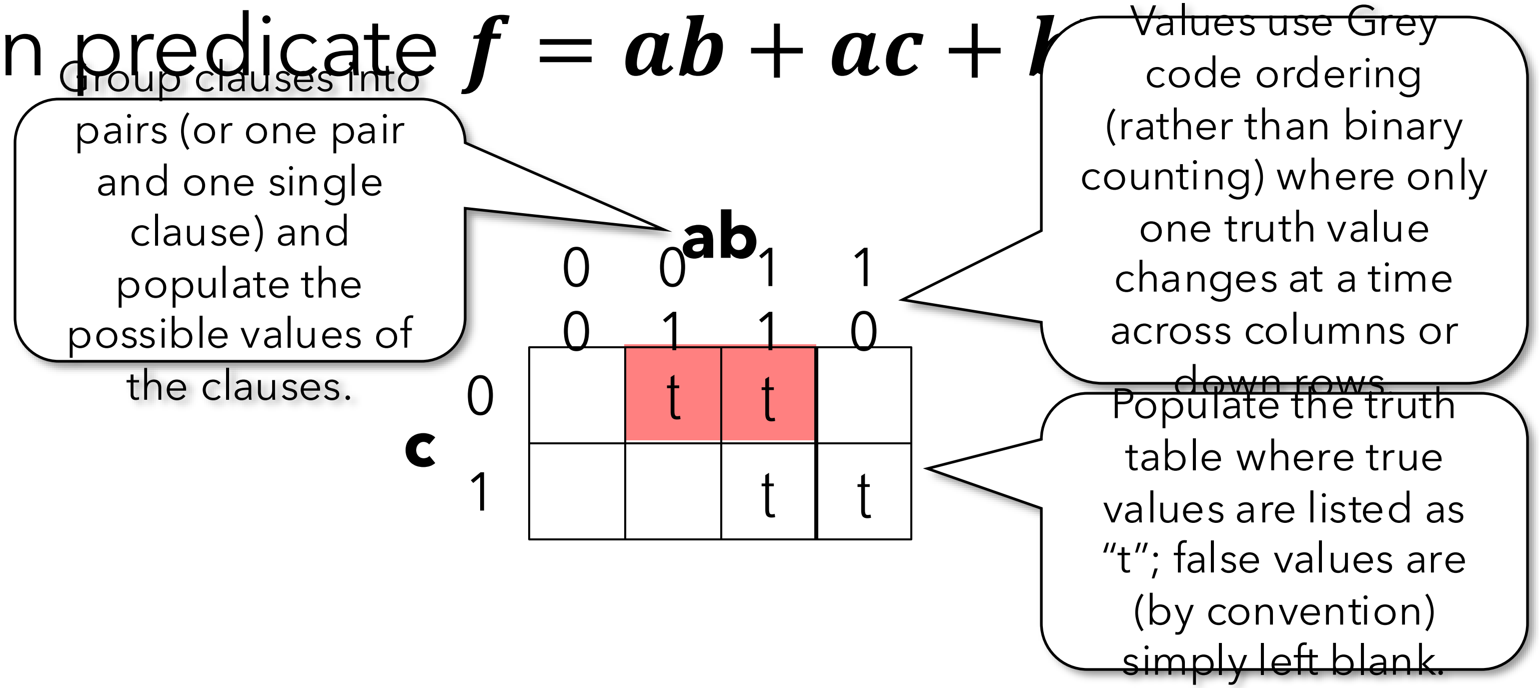
Values use Grey *code* ordering (rather than binary counting) where only one truth value changes at a time across columns or down rows

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 $\boldsymbol{f = ab + ac +}$



		ab			
		00	01	11	10
c	0		t	t	
	1			t	t

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

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

ab

c	00	01	11	10
0		t	t	
1			t	t

Simplifying Predicates

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

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

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

		ab			
		00	01	11	10
c	0		t	t	
	1			t	t

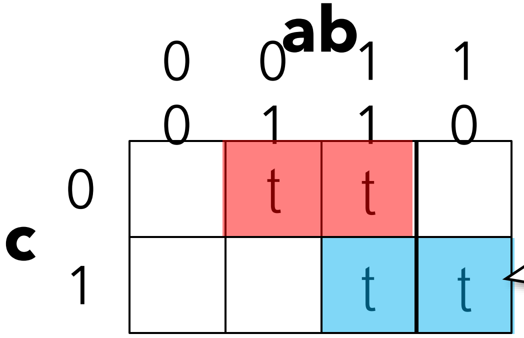
Select maximal rectangles in the table, sized 2^m by 2^n (1x1, 1x2, 2x2, 2x4, 4x4, 4x8, etc.); it's okay if they overlap

Simplifying Predicates

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

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

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



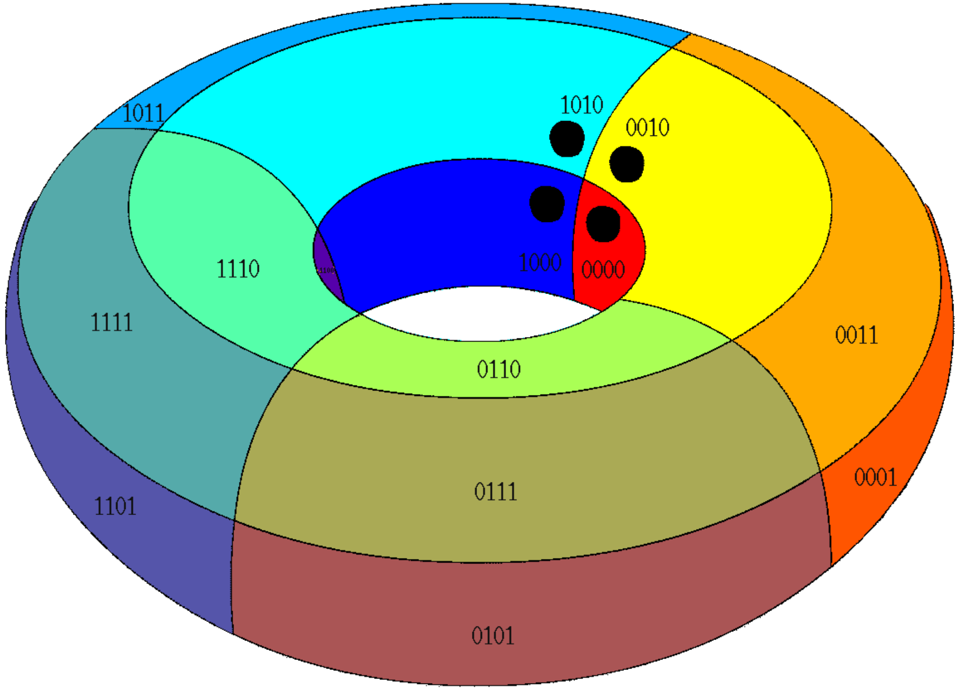
		ab			
		00	01	11	10
c	0		t	t	
	1			t	t

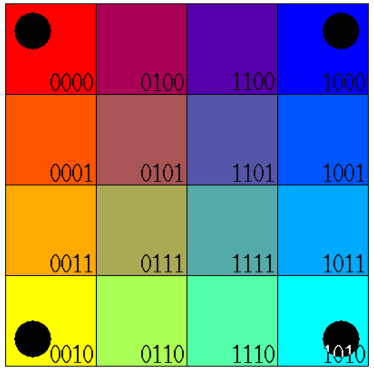
Select maximal rectangles in the table, sized 2^m by 2^n (1x1, 1x2, 2x2, 2x4, 4x4, 4x8, etc.); it's okay if they overlap

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





● 0000	0100	1100	● 1000
0001	0101	1101	1001
0011	0111	1111	1011
● 0010	0110	1110	● 1010

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 $\boldsymbol{f = \overline{bd}}$

Draw the K-map

ab

cd	00	01	11	10
00	t			t
01				
11				
10	t			t

K-Maps are toroidal!

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

Draw the K-map

ab

cd	00	01	11	10
00	t			t
01				
11				
10	t			t

These 4 true values are a single 2x2 rectangle!

Prime Implicants

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

Draw the K-map

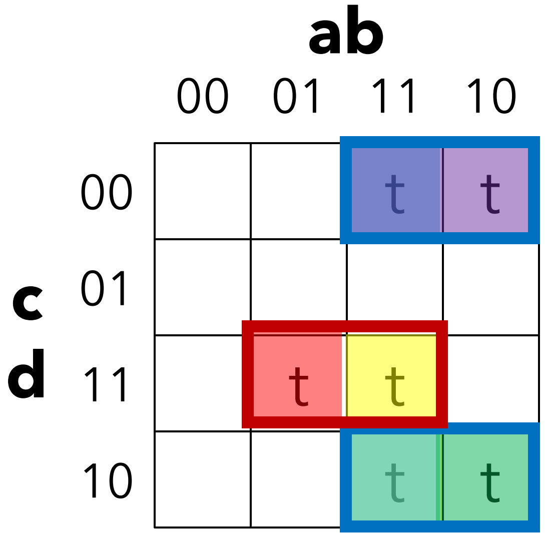
ab

		00	01	11	10
c d	00			t	t
	01				
	11		t	t	
	10			t	t

Prime Implicants

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

Draw the K-map



cd \ ab	00	01	11	10
00			t	t
01				
11		t	t	
10			t	t

Not prime implicants:

- $ab\bar{d}$ (part of $a\bar{d}$)
- $\bar{a}bcd$ (part of bcd)
- $a\bar{b}c\bar{d}$ (part of $a\bar{d}$)
- $a\overline{cd}$ (part of $a\bar{d}$)

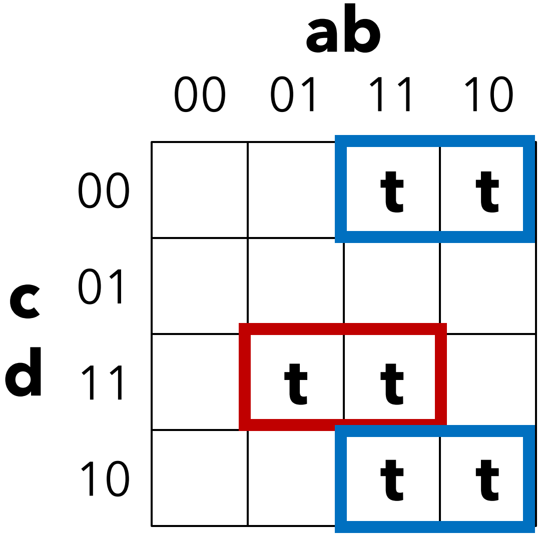
All these have proper subterms that are implicants

Minimal DNF representation: $f = a\bar{d} + bcd$

Minimal representation

A **minimal DNF representation** is one with only *prime*, *non-redundant* implicants

- Not minimal: $\boldsymbol{f = abc + ab\bar{d} + \bar{a}bcd + a\bar{b}c\bar{d} + a\overline{cd}}$
- Minimal (simplified) equivalent from previous slide: $\boldsymbol{f = a\bar{d} + bcd}$



	ab			
cd	00	01	11	10
00			**t**	**t**
01				
11		**t**	**t**	
10			**t**	**t**

Determination

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

Draw K-map

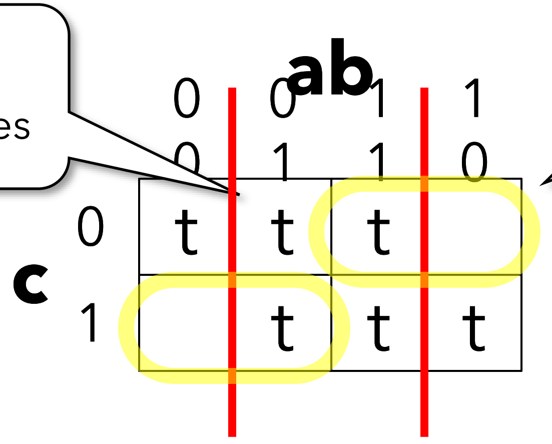
		ab			
		00	01	11	10
c	0	t	t	t	
	1		t	t	t

Determination

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

Draw K-map

Identify the boundaries where b changes value.



c \ ab	00	01	11	10
0	t	t	t	
1		t	t	t

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

b determines f for $a\bar{c} + \bar{a}c$

Predicate Negation

Given predicate $f = ab + bc$, suppose we want to negate f

Draw the K-map for f.

ab

c	00	01	11	10
0			t	
1		t	t	

Negate all the cells in the K-map.

ab

c	00	01	11	10
0	t	t		t
1	t			t

Write down the result: $\overline{f} = \overline{b} + \overline{ac}$

True and False Points

Given $f = ab + cd$

cd \ ab	00	01	11	10
00			t	
01			t	
11	t	t	t	t
10			t	

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 (UTPs)

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 $f = ab + cd$

ab

cd	00	01	11	10
00			t	
01			t	
11	t	t	t	t
10			t	

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 UTP Coverage

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

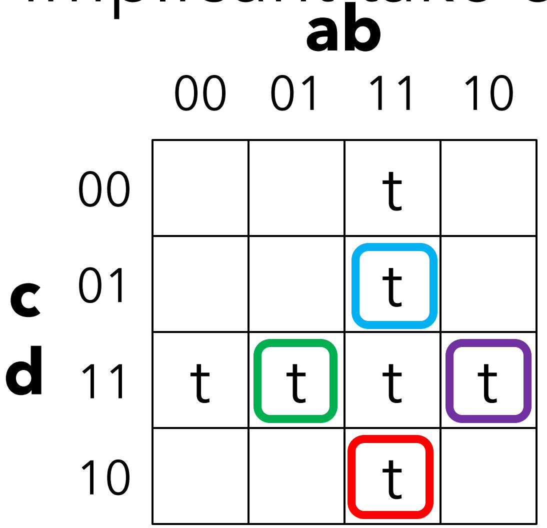
DEFINITION

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 UTP Coverage

Given $f = ab + cd$

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



cd \ **ab**	00	01	11	10
00			t	
01			t	
11	t	t	t	t
10			t	

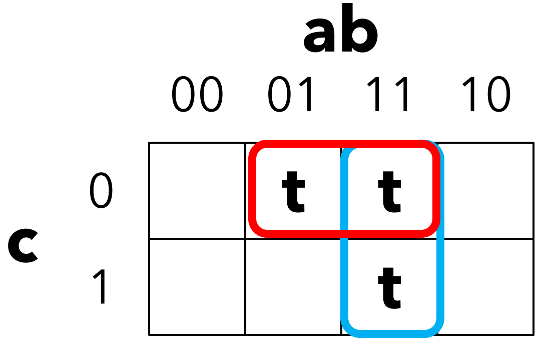
For implicant ab, choose **TTFT** and **TTTF**

For implicant cd, choose **FTTT** and **TFTT**

MUTP Infeasibility

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

- Implicants are { $\boldsymbol{ab}, \boldsymbol{b\bar{c}}$ }
- Both implicants are prime
- Neither implicant is redundant

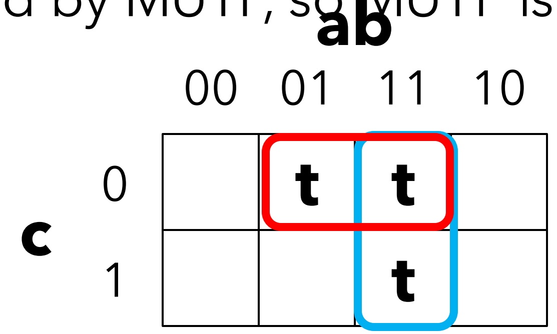


c \ ab	00	01	11	10
0		t	t	
1			t	

MUTP Infeasibility

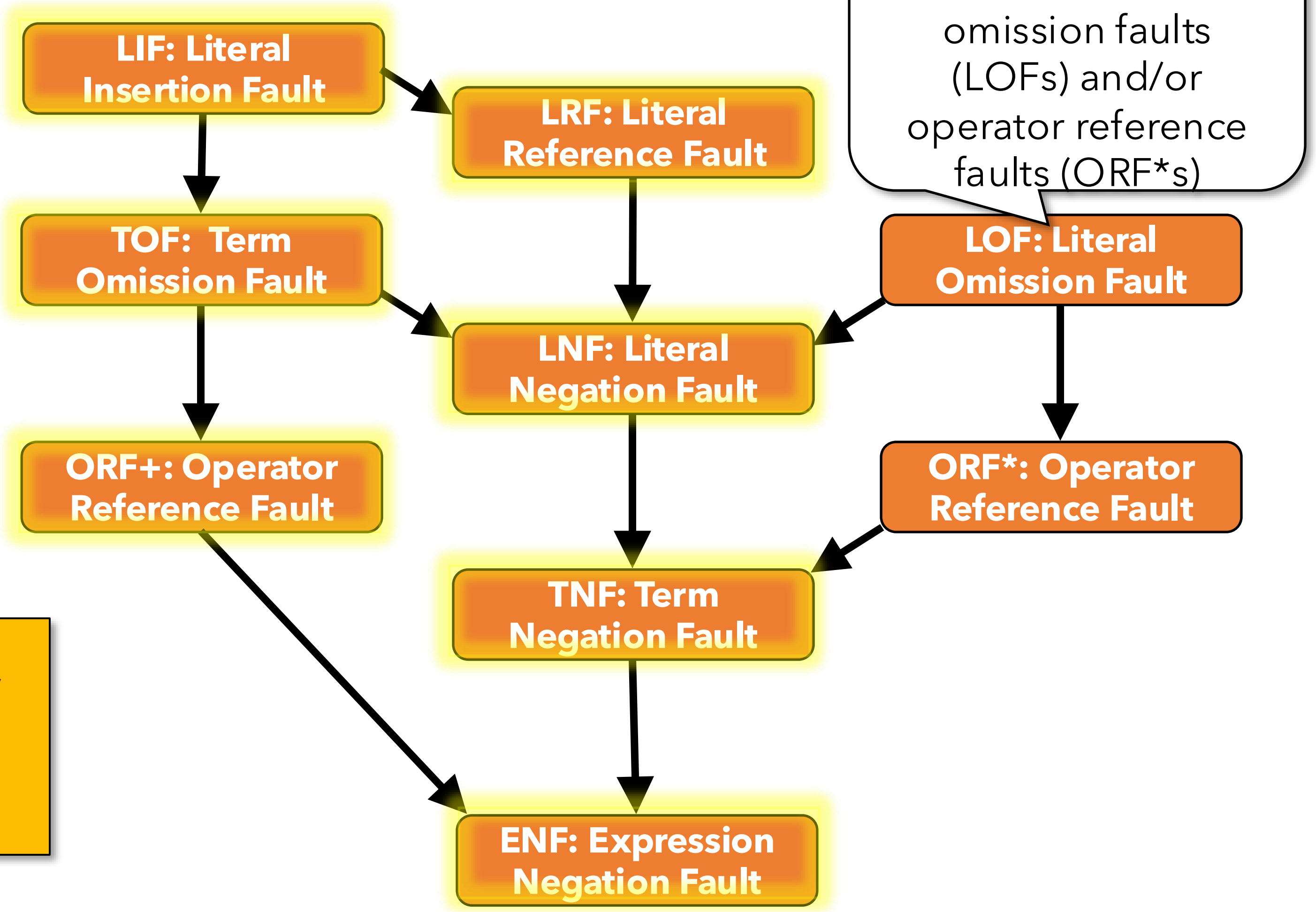
Unique true points required by MUTP

- $\boldsymbol{ab}$: {TTT} causes $\boldsymbol{ab}$ to be true and $\boldsymbol{b\bar{c}}$ 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
- $\boldsymbol{b\bar{c}}$: {FTF} causes $\boldsymbol{ab}$ to be false and $\boldsymbol{b\bar{c}}$ 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



	00	01	11	10
0		t	t	
1			t	

MUTP Fault Detection


LIF: Literal Insertion Fault
LRF: Literal Reference Fault
TOF: Term Omission Fault
LNF: Literal Negation Fault
LOF: Literal Omission Fault
ORF+: Operator Reference Fault
ORF*: Operator Reference Fault
TNF: Term Negation Fault
ENF: Expression Negation Fault

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

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