Intro to Software Testing
Chapter 8.2

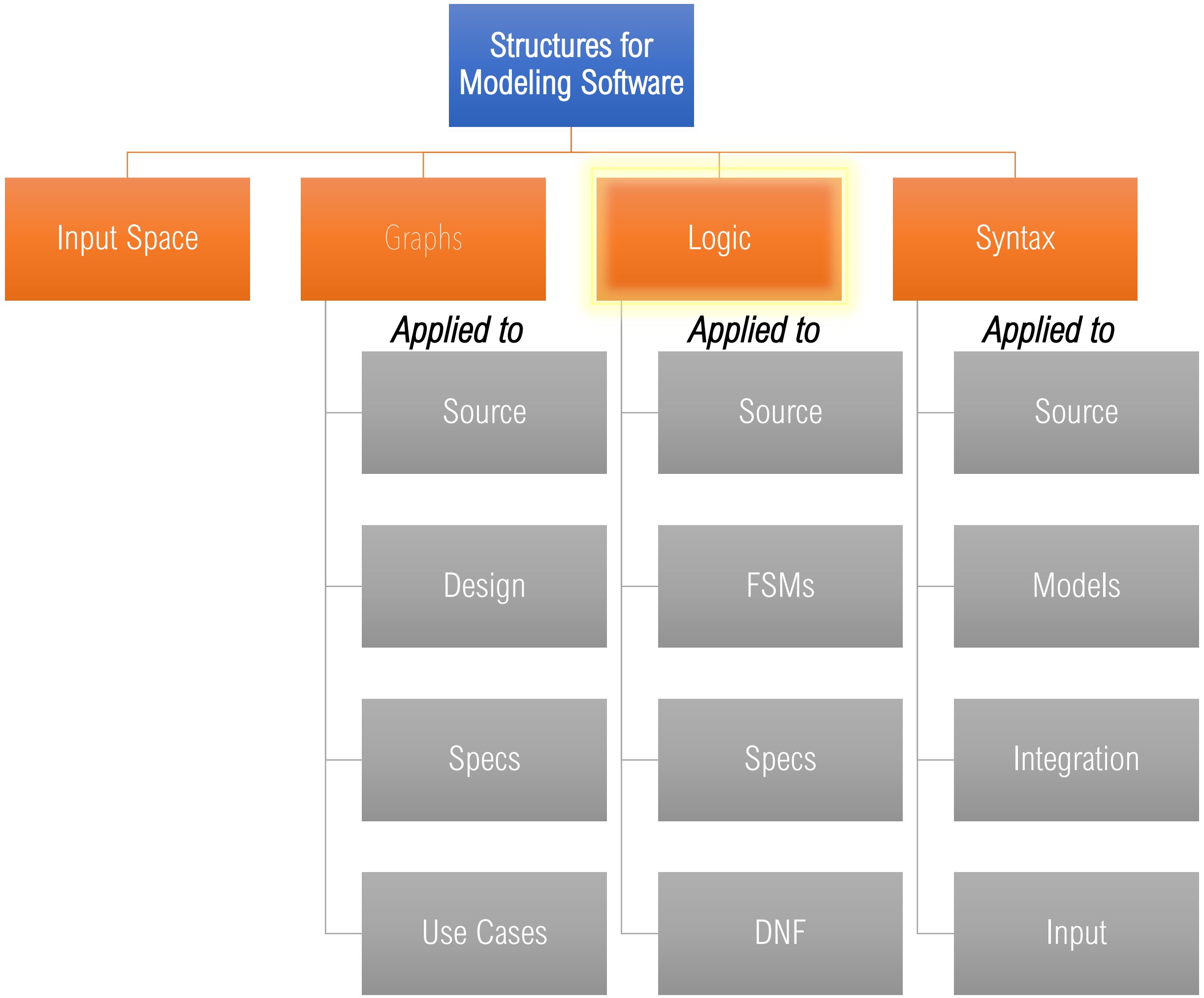
Syntactic Logic Coverage
Disjunctive Normal Form (DNF)

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

https://go.gmu.edu/SWE637

Adapted from slides by Jeff Offutt and Bob Kurtz

Logic Coverage



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: $\boldsymbol{a}, \bar{\boldsymbol{a}}$

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

$\boldsymbol{a} \wedge \boldsymbol{b}$ becomes $\boldsymbol{ab}$

$\neg\boldsymbol{a} \wedge \boldsymbol{b}$ becomes $\bar{\boldsymbol{a}}\boldsymbol{b}$

$\neg\boldsymbol{a} \wedge \neg\boldsymbol{b}$ 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 by +, for example:

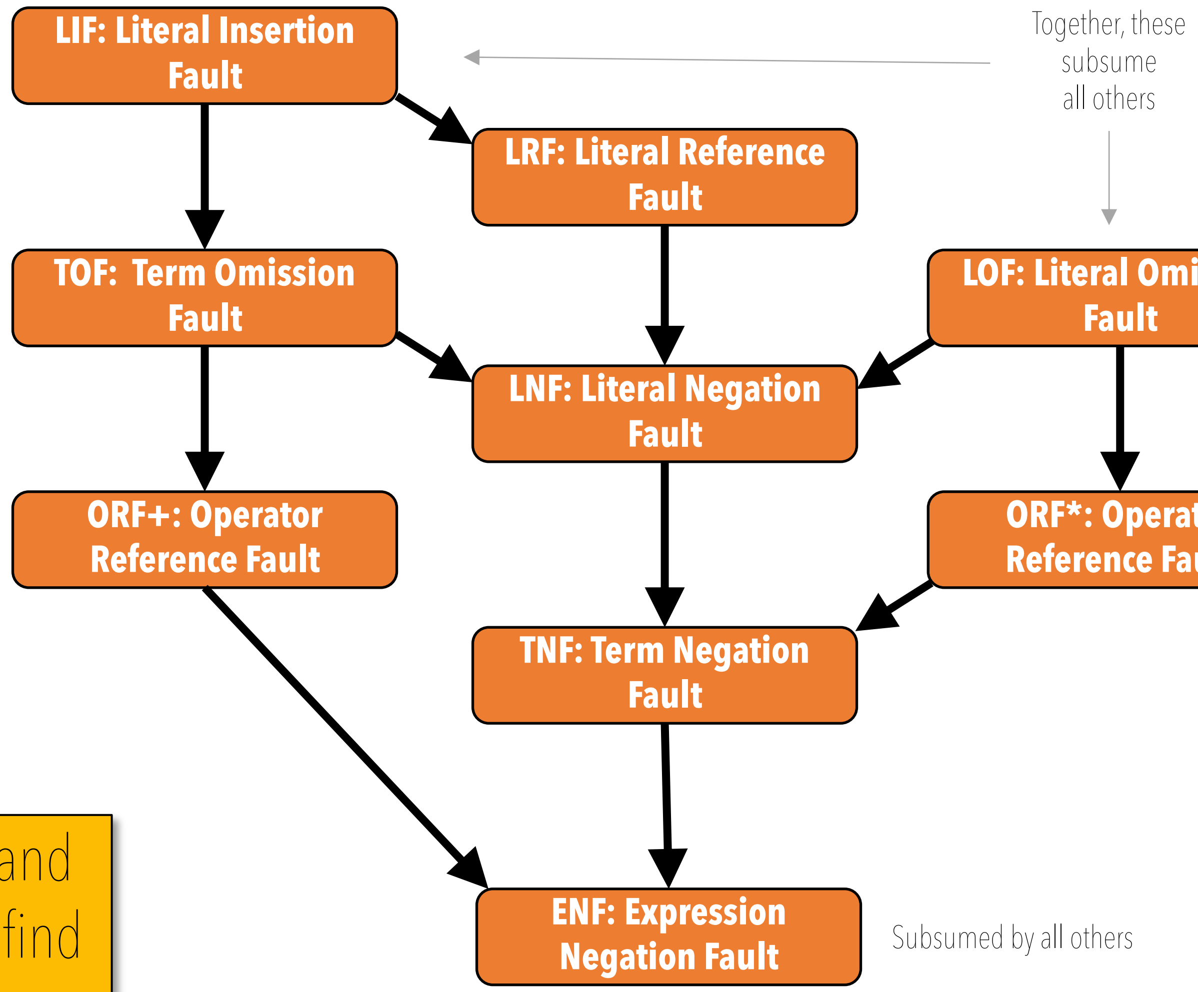
$\boldsymbol{a} \vee \boldsymbol{b}$ becomes $\boldsymbol{a} + \boldsymbol{b}$

DNF Fault Classes

There are 9 types of syntactic faults on DNF predicates; we want criteria that are guaranteed to find 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 fault	$f = ab + c$	$f = a + c$
LIF: literal insertion fault	$f = ab + c$	$f = ab + bc$
ORF+: operator reference fault	$f = ab + c$	$f = abc$
ORF*: operator reference fault	$f = ab + c$	$f = a + b + c$

DNF Fault Class Subsumption



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 $\boldsymbol{f}$, so we include DNF negation of the entire predicate $\boldsymbol{f}$

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 }
- 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

$\boldsymbol{abc}$ 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 $\boldsymbol{f = ab + a\bar{b}c}$, $\boldsymbol{ab}$ 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 $\boldsymbol{f = ab + ac}$, 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 $\boldsymbol{f = ab + ac + b\bar{c}}$, implicant $\boldsymbol{ab}$ is redundant because the predicate can be simplified to $\boldsymbol{ac + b\bar{c}}$ (again, we'll soon see how to determine that)

Simplifying Predicates

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

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

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

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

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

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

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

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

c \ ab	00	01	11	10
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 $\boldsymbol{f = ab + ac + b\bar{c}}$

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

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

ab

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

c \ ab	00	01	11	10
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

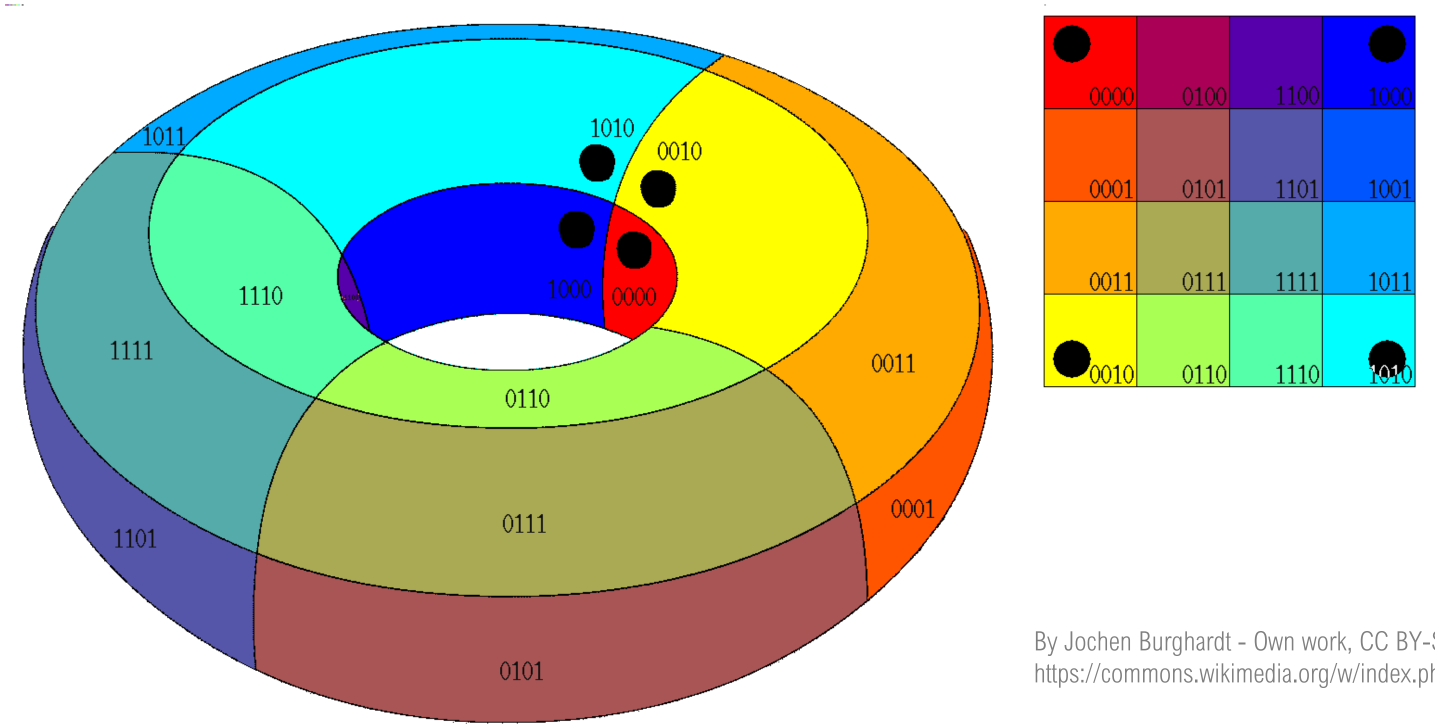
c	00	01	11	10
0		t	t	
1			t	t

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



K-Maps are Toroidal

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

Draw the K-map

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

K-Maps are Toroidal

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

Draw the K-map

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

Draw the K-map

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

Prime Implicants

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

Draw the K-map

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

Not prime implicants:

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

All these have proper subterms that are implicants

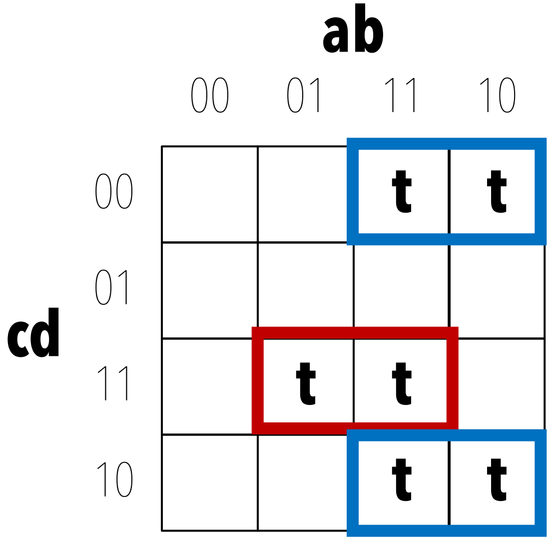
Minimal DNF representation: $\boldsymbol{f} = \boldsymbol{a\bar{d}} + \boldsymbol{bcd}$

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: $\boldsymbol{f = a\bar{d} + bcd}$



Determination

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

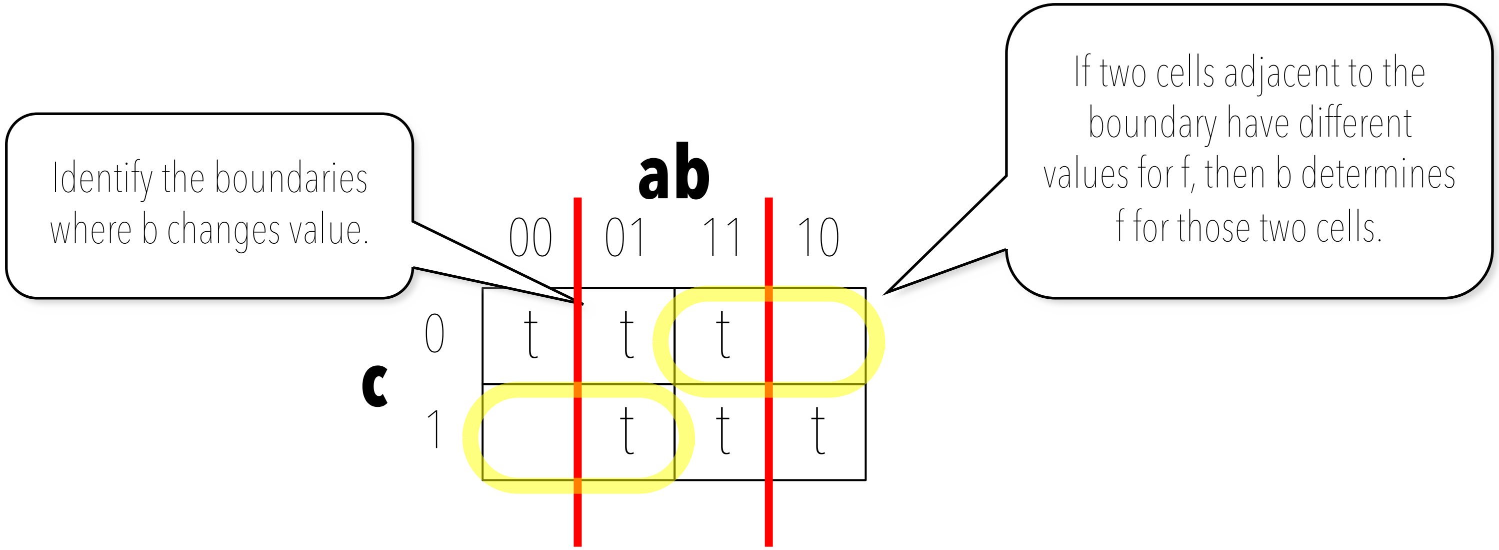
Draw K-map

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

Determination

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

Draw K-map



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

Predicate Negation

Given predicate $\boldsymbol{f} = \boldsymbol{ab} + \boldsymbol{bc}$, suppose we want to negate $\boldsymbol{f}$

Draw the K-map for f.

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

Negate all the cells in the K-map.

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

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

True and False Points

Given $f = ab + cd$

ab

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

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 $f = ab + cd$

cd \ ab	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 Unique True Point 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 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

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**

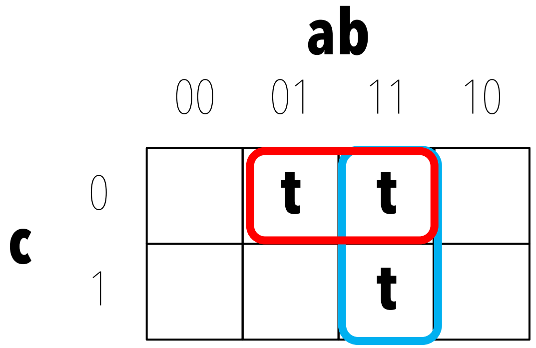
MUTP test set:

{ **TTFT, TTTF, FTTT, TFTT** }

MUTP Infeasibility

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

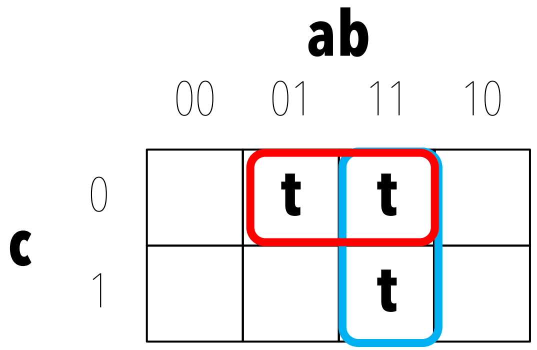
- Implicants are $\{ ab, b\bar{c} \}$
- Both implicants are prime
- Neither implicant is redundant



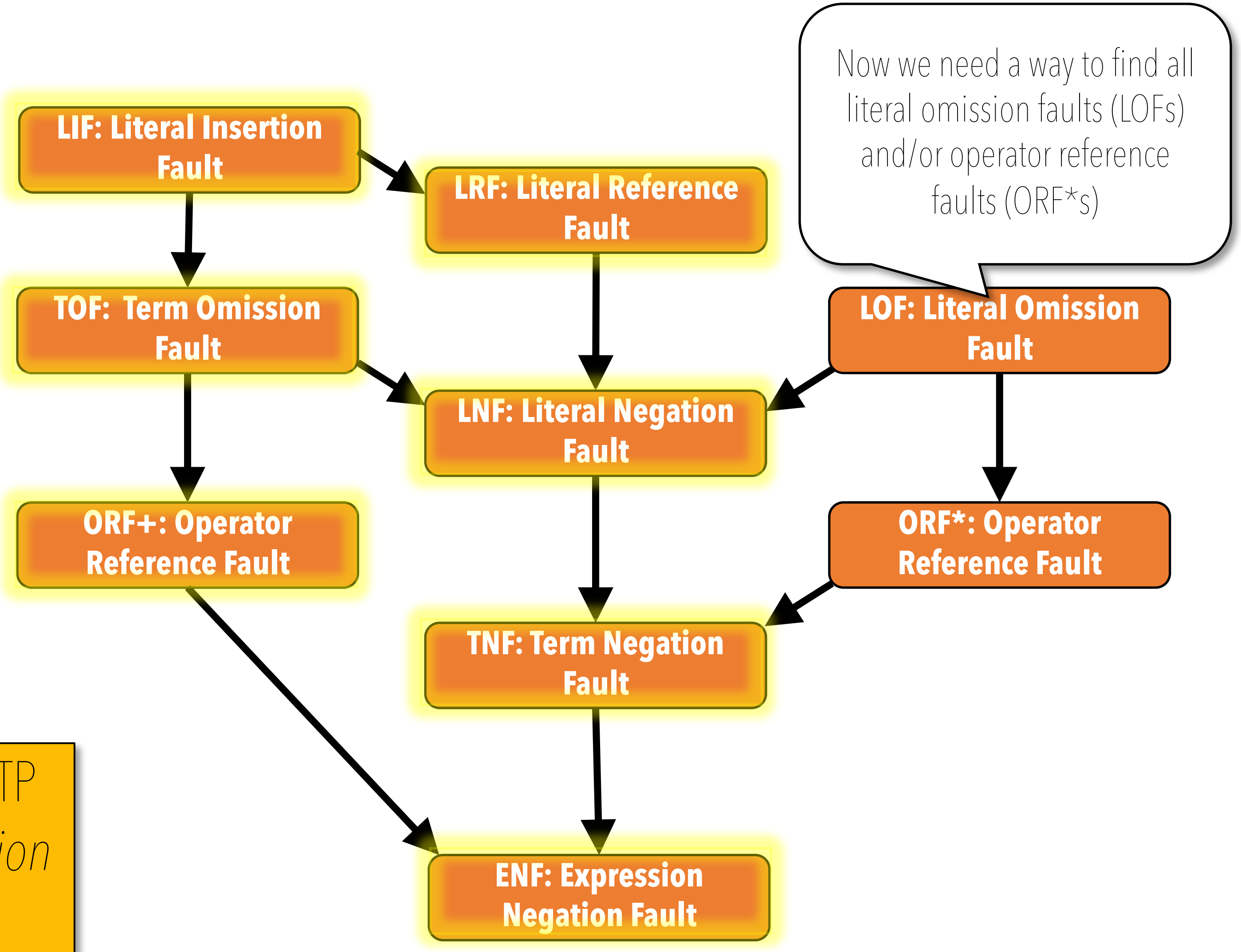
MUTP Infeasibility

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



MUTP Fault Detection



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

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***

DEFINITION

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

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

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.

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

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

For clause ***c*** in ***cd***, choose UTP and NFP

- **FFTT** and **FFFT**, or
- **FTTT** and **FTFT** or
- **TFTT** and **TFFT**

For clause ***d*** in ***cd***, choose UTP and NFP

- **FFTT** and **FFTF**, or
- **FTTT** and **FTTF** 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 $\boldsymbol{f = ab + cd}$

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

ab / **cd**

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

For clause ***a*** in ***ab***, choose UTP and NFP
TTFT and **FTFT**

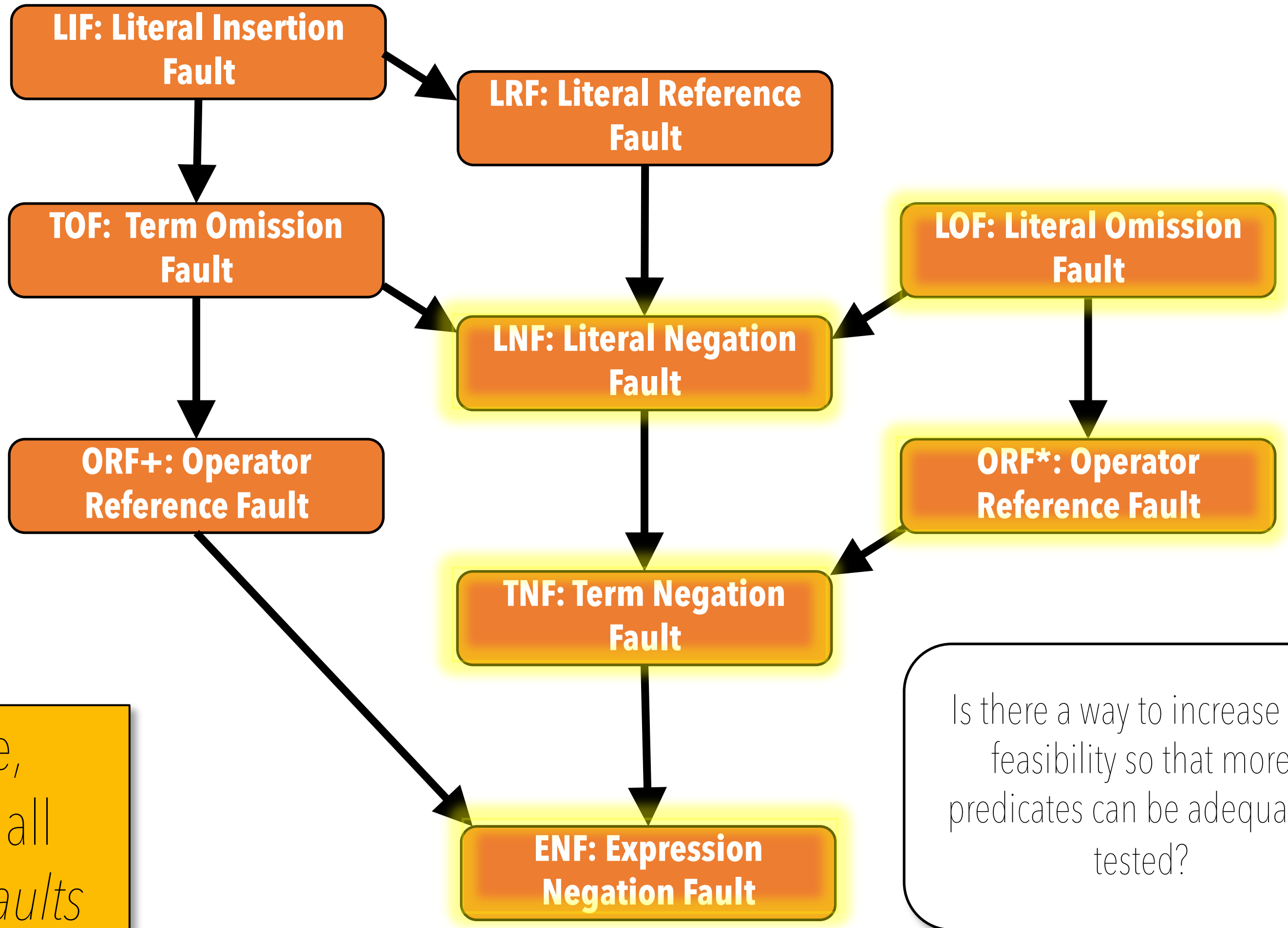
For clause ***b*** in ***ab***, choose UTP and NFP
TTFT and **TFFT**

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, CUTNPFP finds all *literal omission faults* (LOFs)

Is there a way to increase the feasibility so that more predicates can be adequately tested?

Multiple Near False Point Coverage

We saw earlier that MUTP can easily be infeasible in its entirety, and the same is true of CUTPNFP.

DEFINITION

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.

MNFP Example

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.

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

For clause ***a*** in ***ab***, choose NFP **FTFT** and NFP **FTTF**

For ***b*** in ***ab***, choose **TFFT** 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** }

MUMCUT

We can combine the previous three criteria (MUTP, CUTPNFP, and MNFP)

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 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