Introduction to Software Testing
Chapter 8.2
Syntactic Logic Coverage Criteria
(Disjunctive Normal Form)

Paul Ammann & Jeff Offutt

http://www.cs.gmu.edu/~offutt/softwaretest/

updated April, 2017

Disjunctive Normal Form

- Common Representation for Boolean Functions
  - Slightly Different Notation for Operators
  - Slightly Different Terminology
- Basics:
  - A *literal* is a clause or the negation (overstrike) of a clause
    - Examples: a, $\overline{a}$
  - A *term* is a set of literals connected by logical "and"
    - "and" is denoted by adjacency instead of $\wedge$
    - Examples: ab, $a\overline{b}$, $\overline{a}\overline{b}$ for $a \wedge b$, $a \wedge \neg b$, $\neg a \wedge \neg b$
  - A *(disjunctive normal form) predicate* is a set of terms connected by "or"
    - "or" is denoted by **+** instead of $\vee$
    - Examples: $abc + \overline{a}b + a\overline{c}$
    - Terms are also called "implicants"
      - If a term is true, that implies the predicate is true

Implicant Coverage (8.2.1)

- Obvious coverage idea : Make each implicant evaluate to "true"
  - Problem : Only tests "true" cases for the predicate
  - Solution : Include DNF representations for negation

<u>Implicant Coverage (IC)</u> : Given DNF representations of a predicate f and its negation $\bar{f}$, for each implicant in f and $\bar{f}$, TR contains the requirement that the implicant evaluate to true.

- Example: $f = ab + b\bar{c}$ $\quad \bar{f} = \bar{b} + \bar{a}c$
  - Implicants: $\{ ab, b\bar{c}, \bar{b}, \bar{a}c \}$
  - Possible test set: {TTF, FFT}
- Observation: IC is relatively weak

Improving on Implicant Coverage (8.2.2)

- Additional Definitions :
  - A *proper subterm* is a term with one or more clauses removed
    - Example: *abc* has 6 proper subterms: *a, b, c, ab, ac, bc*
  - A *prime implicant* is an implicant such that no proper subterm is also an implicant
    - Example: $f = ab + ab\overline{c}$
    - Implicant $ab\overline{c}$ is not a prime implicant (due to proper subterm *a*)
  - A *redundant implicant* is an implicant that can be removed without changing the value of the predicate
    - Example: $f = ab + ac + b\overline{c}$
    - *ab* is redundant
    - Predicate can be written: $ac + b\overline{c}$

Unique True Points

- A *minimal DNF representation* is one with only prime, non-redundant implicants
- A *unique true point* with respect to a given implicant is an assignment of truth values so that
  – The given implicant is true, and
  – All other implicants are false
- A unique true point test focuses on just one implicant
- A minimal representation guarantees the existence of at least one unique true point for each implicant

<u>Multiple Unique True Point Coverage (MUTP)</u> : Given minimal DNF representations of a predicate *f*, *for each implicant i, choose unique true points (UTPs) such that clauses not in i take on values T and F.*

Unique True Point Example

- Consider again : $f = ab + b\bar{c}$
  - Implicants : $\{ ab, b\bar{c} \}$
  - Each implicant is prime
  - No implicant is redundant
- Unique true points :
  - ab: {TTT}
  - $b\bar{c}$: {TFT}
  - MUTP requires both of these
- But MUTP is still infeasible for both implicants
  - Not enough UTPs for clauses to take on all truth values
  - Later, we will have an example where MUTP is feasible

Near False Points (8.2.3)

- A *near false point* 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 left as is), *i* (and hence *f*) evaluates to true
- Relation to *determination*: at a near false point, *c* determines *f*

<u>Unique True Point and Near False Point Pair Coverage (CUTPNFP)</u> : 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*.

- Note that definition only mentions *f*, and not $\bar{f}$
- Clearly, CUTPNFP subsumes RACC

CUTPNFP Example

- Consider $f = ab + cd$
  - Implicant *ab* has 3 unique true points : {TTFF, TTFT, TTTF}
    - For clause *a*, we can pair unique true point TTFF with near false point FTFF
    - For clause b, we can pair unique true point TTFF with near false point TFFF
  - Implicant *cd* has 3 unique true points : {FFTT, FTTT, TFTT}
    - For clause *c*, we can pair unique true point FFTT with near false point FFFT
    - For clause *d*, we can pair unique true point FFTT with near false point FFTF
- CUTPNFP set : {TTFF, FFTT, TFFF, FTFF, FFTF, FFFT}
  - First two tests are unique true points; others are near false points
- Rough number of tests required: # implicants * # literals

The MNFP Criterion (8.2.3)

The next two criteria provide enough scaffolding to make guarantees about fault detection (see later slides)

<u>Multiple Near False Point Coverage (MNFP)</u> : Given a minimal DNF representation of a predicate *f*, for each literal *c* in each implicant *i*, TR choose near false points (NFPs) such that clauses not in i take on values T and F.

—

MNFP Example

- Consider again : $f = ab + b\bar{c}$
  - Implicants : $\{ ab, b\bar{c} \}$
- Unique true points :
  - *ab:*
    - NFP for a where c: {FTT,FTF}
    - NFPs for b where c = T, F: {TFT, TFF}
  - $b\bar{c}$:
    - NFPs for $\bar{\text{b}}$ where a = T, F: {TTT, FTT}
    - NFP for c where a : {TFF,FFF}

The MUMCUT Criterion (8.2.3)

Together, these three criteria provide enough scaffolding to make guarantees about fault detection (see later slides)

<u>MUMCUT</u> : Given a minimal DNF representation of a predicate *f*, apply MUTP, CUTPNFP, and MNFP.

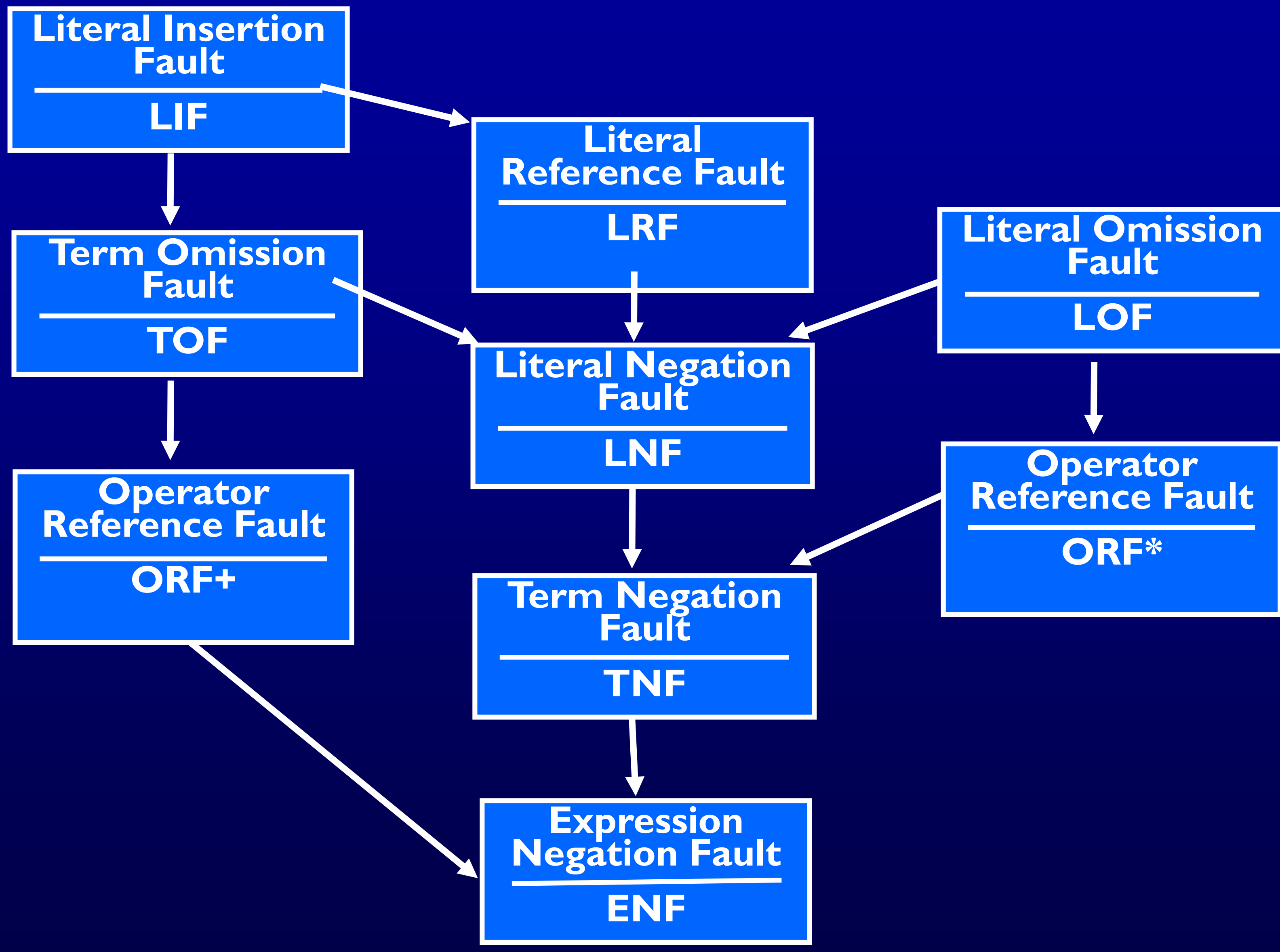
DNF Fault Classes

- 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 $f = ab+c$ $f' = ab+\bar{c}$
- 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 $f = ab + c$ $f' = a + b + c$

Key idea is that fault classes are related with respect to testing :

Test sets guaranteed to detect certain faults are also guaranteed to detect additional faults

Fault Detection Relationships



Karnaugh Maps for Testing Logic Expressions (8.2.4)

- Fair Warning
  - We *use*, rather than *teach*, Karnaugh Maps
  - Newcomers to Karnaugh Maps probably need a tutorial
    - Suggestion: Google "Karnaugh Map Tutorial"
- Our goal: Apply Karnaugh Maps to concepts used to test logic expressions
  - Identify when a clause determines a predicate
  - Identify the negation of a predicate
  - Identify prime implicants and redundant implicants
  - Identify unique true points
  - Identify unique true point / near false point pairs
- No new material here on *testing*
  - Just fast shortcuts for concepts already presented

K-Map: A Clause Determines a Predicate

- Consider the predicate : $f = b + \bar{a}\bar{c} + ac$
- Suppose we want to identify when *b* determines *f*
- The dashed line highlights where *b* changes value
  - If two cells joined by the dashed line have different values for *f*, then *b* determines *f* for those two cells
  - *b* determines *f*: $\bar{a}c + a\bar{c}$ (but NOT at *ac* or $\bar{a}\bar{c}$)
- Repeat for clauses *a* and *c*

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

K-Map: Negation of a predicate

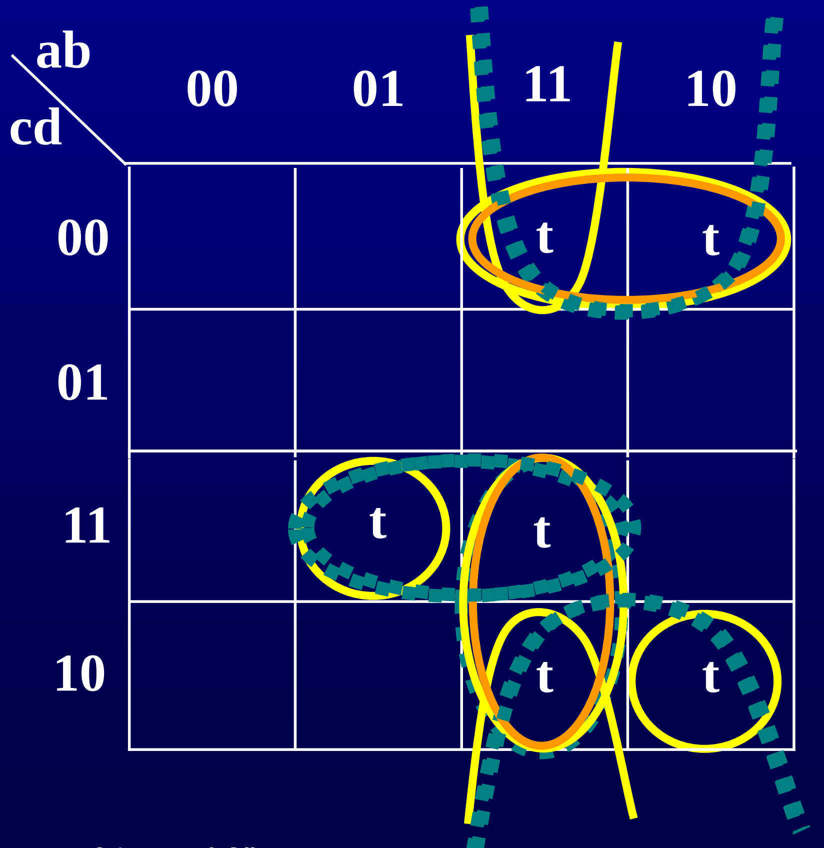
- Consider the predicate: $f = ab + bc$
- Draw the Karnaugh Map for the negation
  - Identify groups
  - Write down negation: $\overline{f} = \overline{b} + \overline{a}\,\overline{c}$

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

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

K-Map: Prime and Redundant Implicants

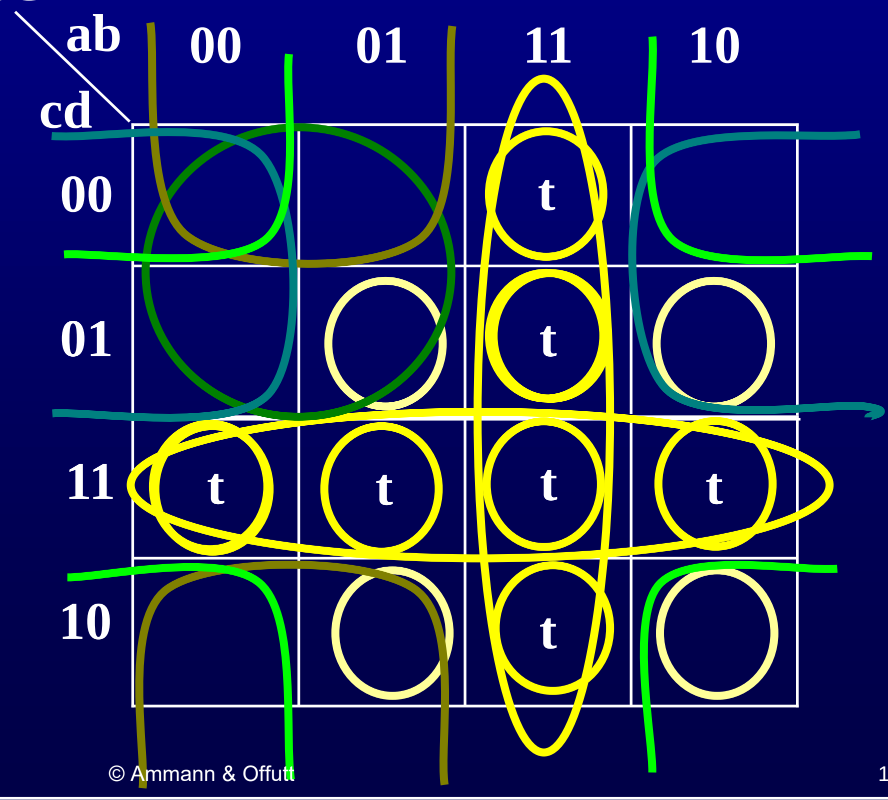
- Consider the predicate: $f = abc + ab\bar{d} + \bar{a}bcd + a\bar{b}c\bar{d} + a\bar{c}\bar{d}$
- Draw the Karnaugh Map
- Implicants that are not prime: $ab\bar{d}$, $\bar{a}bcd$, $a\bar{b}c\bar{d}$, $a\bar{c}\bar{d}$
- Redundant implicant: $ab\bar{d}$
- Prime implicants
  - Three: $a\bar{d}$, bcd, abc
  - The last is redundant
  - Minimal DNF representation
    - $f = a\bar{d} + bcd$



K-Map: Unique True Points

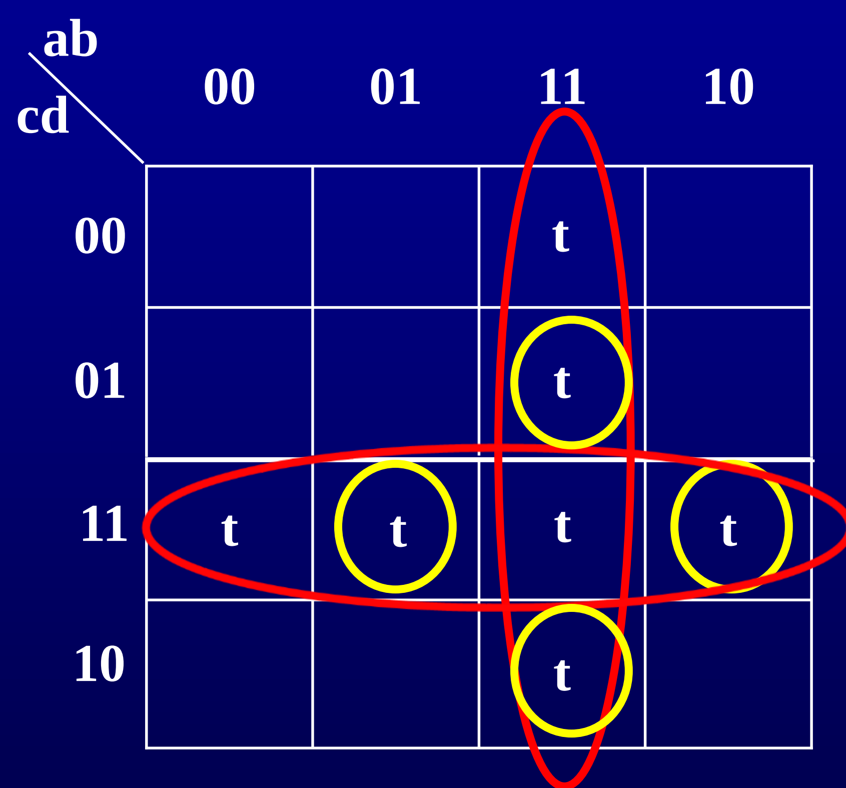
- Consider the predicate: $f = ab + cd$
- Three unique true points for *ab*
  - TTFF, TTFT, TTTF
  - TTTT is a true point, but not a unique true point
- Three unique true points for *cd*
  - FFTT, FTTT, TFTT
- Unique true points for $\overline{f}$

  $\overline{f} = \bar{a}\bar{c} + \bar{b}\bar{c} + \bar{a}\bar{d} + \bar{b}\bar{d}$
  - FTFT,TFFT, FTTF, TFTF



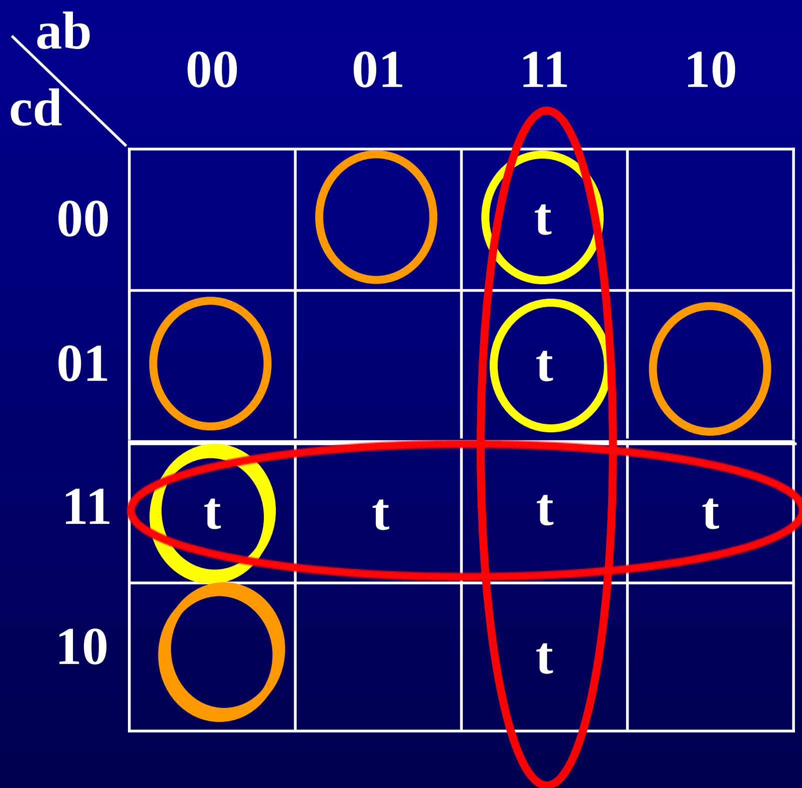
MUTP: Multiple Unique True Points

- For each implicant find unique true points (UTPs) so that
  - Literals not in implicant take on values T and F
- Consider the DNF predicate:
  - $f = ab + cd$
- For implicant *ab*
  - Choose TTFT, TTTF
- For implicant cd
  - Choose FTTT, TFTT
- MUTP test set
  - {TTFT, TTTF, FTTT, TFTT}



CUTPNFP: Corresponding Unique True Point Near False Point Pairs

- Consider the DNF predicate: $f = ab + cd$
- For implicant *ab*
  - For *a*, choose UTP, NFP pair
    - TTFF, FTFF
  - For *b*, choose UTP, NFP pair
    - TTFT, TFFT
- For implicant cd
  - For *c*, choose UTP, NFP pair
    - FFTT, FFFT
  - For *d*, choose UTP, NFP pair
    - FFTT, FFTF
- Possible CUTPNFP test set
  - {TTFF, TTFT, FFTT //UTPs
    FTFF, TFFT, FFFT, FFTF} //NFPs



MNFP : Multiple Near False Points

- Find NFP tests for each literal such that all literals not in the term attain F and T
- Consider the DNF predicate:
  - $f = ab + cd$
- For implicant *ab*
  - Choose FTFT, FTTF for a
  - Choose TFFT, TFTF for b
- For implicant cd
  - Choose FTFT, TFFT for c
  - Choose FTTF, TFTF for d
- MNFP test set
  - {TFTF, TFFT, FTTF, TFTF}
- Example is small, but generally MNFP is large

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

Minimal-MUMCUT Criterion Kaminski et al (ICST 2009)

- Minimal-MUMCUT uses low level criterion feasibility analysis
  - Adds CUTPNFP and MNFP only when necessary
- Minimal-MUMCUT guarantees detecting LIF, LRF, LOF
  - And thus all 9 faults in the hierarchy