Introduction to Software Testing
Chapter 3.6
Disjunctive Normal Form Criteria

Paul Ammann & Jeff Offutt

http://www.cs.gmu.edu/~offutt/softwaretest/
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 $\land$
    - Examples: $ab, \overline{ab}, \overline{a}b$ for $a \land b, a \land \overline{b}, \overline{a} \land \overline{b}$
  - A *(disjunctive normal form) predicate* is a set of terms connected by “or”
    - “or” is denoted by $+$ instead of $\lor$
    - Examples: $abc + \overline{ab} + a\overline{c}$
    - Terms are also called “implicants”
      - If a term is true, that implies the predicate is true
Implicant Coverage

• Obvious coverage idea: Make each implicant evaluate to “true”.
  – Problem: Only tests “true” cases for the predicate.
  – Solution: Include DNF representations for negation.

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

• Example: \( f = ab + b\overline{c} \quad \overline{f} = \overline{b} + \overline{a}c \)
  • Implicants: \( \{ ab, b\overline{c}, \overline{b}, \overline{a}c \} \)
  • Possible test set: \( \{ \text{TTF, FFT} \} \)
  • Observation: IC is relatively weak
Improving on Implicant Coverage

- **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: $f = ab + abc$
    - Implicant $ab$ is a prime implicant
    - Implicant $abc$ is not a prime implicant (due to proper subterm $ac$)
  - 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, nonredundant 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
- Hence 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

**Unique True Point Coverage (UTPC)**: Given minimal DNF representations of a predicate $f$ and its negation $\overline{f}$, TR contains a unique true point for each implicant in $f$ and $\overline{f}$. 

*Introduction to Software Testing (Ch 3)*

© Ammann & Offutt 5
Unique True Point Example

- Consider again: $f = ab + b\overline{c}$, $\overline{f} = \overline{b} + \overline{a}c$
  - Implicants: $\{ab, bc, \overline{b}, \overline{ac}\}$
  - Each of these implicants is prime
  - None of these implicants is redundant

- Unique true points:
  - $ab$: $\{\text{TTT}\}$
  - $bc$: $\{\text{FTF}\}$
  - $\overline{b}$: $\{\text{FFF, TFF, TFT}\}$
  - $\overline{ac}$: $\{\text{FTT}\}$

- Note that there are three possible (minimal) tests satisfying UTPC

- UTPC is fairly powerful
  - Exponential in general, but reasonable cost for many common functions
  - No subsumption relation wrt any of the ACC or ICC Criteria
Near False Points

• 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 $f$
  – Hence we should expect relationship to ACC criteria

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 $c$.

• Note that definition only mentions $f$, and not $\overline{f}$.
• Clearly, CUTPNFP subsumes RACC
CUTPNFP Example

• Consider $f = ab + cd$
  – For implicant $ab$, we have 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
  – For implicant $cd$, we have 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, FTTT, TFFF, FTFF, FFTF, FFFT\}
  – First two tests are unique true points; others are near false points
• Rough number of tests required: $\#$ implicants $\times \#$ literals
DNF Fault Classes

- ENF: Expression Negation Fault
  \[ f = ab+c \quad f' = \overline{ab}+c \]
- TNF: Term Negation Fault
  \[ f = ab+c \quad f' = \overline{ab}+c \]
- TOF: Term Omission Fault
  \[ f = ab+c \quad f' = ab \]
- LNF: Literal Negation Fault
  \[ f = ab+c \quad f' = a\overline{b}+c \]
- LRF: Literal Reference Fault
  \[ f = ab + bcd \quad f' = ad + bcd \]
- LOF: Literal Omission Fault
  \[ f = ab + c \quad f' = a + c \]
- LIF: Literal Insertion Fault
  \[ f = ab + c \quad f' = ab + bc \]
- ORF+: Operator Reference Fault
  \[ f = ab + c \quad f' = abc \]
- ORF*: Operator Reference Fault
  \[ f = ab + c \quad f' = a + b + c \]

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

Literal Insertion Fault

- LIF

Term Omission Fault

- TOF

Operator Reference Fault

- ORF+

Literal Reference Fault

- LRF

Literal Negation Fault

- LNF

Term Negation Fault

- TNF

Expression Negation Fault

- ENF

Literal Omission Fault

- LOF

Operator Reference Fault

- ORF*
Understanding The Detection Relationships

- Consider the TOF (Term Omission Fault) class
  - UTPC requires a unique true point for every implicant (term)
  - Hence UTPC detects all TOF faults
  - From the diagram, UTPC also detects:
    - All LNF faults (Unique true point for implicant now false)
    - All TNF faults (All true points for implicant are now false points)
    - All ORF+ faults (Unique true points for joined terms now false)
    - All ENF faults (Any single test detects this…)

- Although CUTPNFP does not subsume UTPC, CUTPNFP detects all fault classes that UTPC detects (Converse is false)

- Consider what this says about the notions of subsumption vs. fault detection

- Literature has many more powerful (and more expensive) DNF criteria
  - In particular, possible to detect entire fault hierarchy (MUMCUT)
Karnaugh Maps for Testing Logic Expressions

• Fair Warning
  – We *use*, rather than *present*, Karnaugh Maps
  – Newcomer to Karnaugh Maps probably needs 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 \( c \)
K-Map: Negation of a predicate

• Consider the predicate: \( f = ab + bc \)
• Draw the Karnaugh Map for the negation
  – Identify groups
  – Write down negation: \( \overline{f} = \overline{b} + \overline{a} \overline{c} \)
K-Map: Prime and redundant implicants

- Consider the predicate: \( f = abc + abd + \overline{abcd} + \overline{abcd} + \overline{acd} \)

- Draw the Karnaugh Map

- Implicants that are not prime: \( abd, \overline{abcd}, \overline{abcd}, \overline{acd} \)

- Redundant implicant: \( abd \)

- Prime implicants
  - Three: \( \overline{ad}, bcd, abc \)
  - The last is redundant
  - Minimal DNF representation
    - \( f = \overline{ad} + bcd \)
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 \( cd \)
  – FFFT, FTTT, TFTT

• Unique true points for \( \overline{f} \)
  \( \overline{f} = \overline{a}c + \overline{b}c + \overline{a}d + \overline{b}d \)
  – FTFT, TFFT, FTTF, TFTF

• Possible UTPC test set
  – \( f \): \{TTFT, FTTT\}
  – \( \overline{f} \): \{FTFT, TFFT, FTTF, TFTF\}
K-Map: Unique True Point/Near False Point Pairs

- Consider the 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
    - FFFT, FFFT
  - For \( d \), choose UTP, NFP pair
    - FFFT, FFTF

- Possible CUTPNFP test set
  - \{TTFF, TTFT, FFFT \} //UTPs
  - \{FTFF, TFFT, FFTT, FFTF\} //NFPs