CS1632, Lecture 9: Pairwise and Combinatorial Testing

Bill Laboon

Let’s Test A Word Processor

- Specifically, its ten possible font effects
  - Italic
  - Bold
  - Underline
  - Strikethrough
  - Superscript
  - Shadow
  - Embossed
  - 3-D
  - Outline
  - Inverse

These can be combined

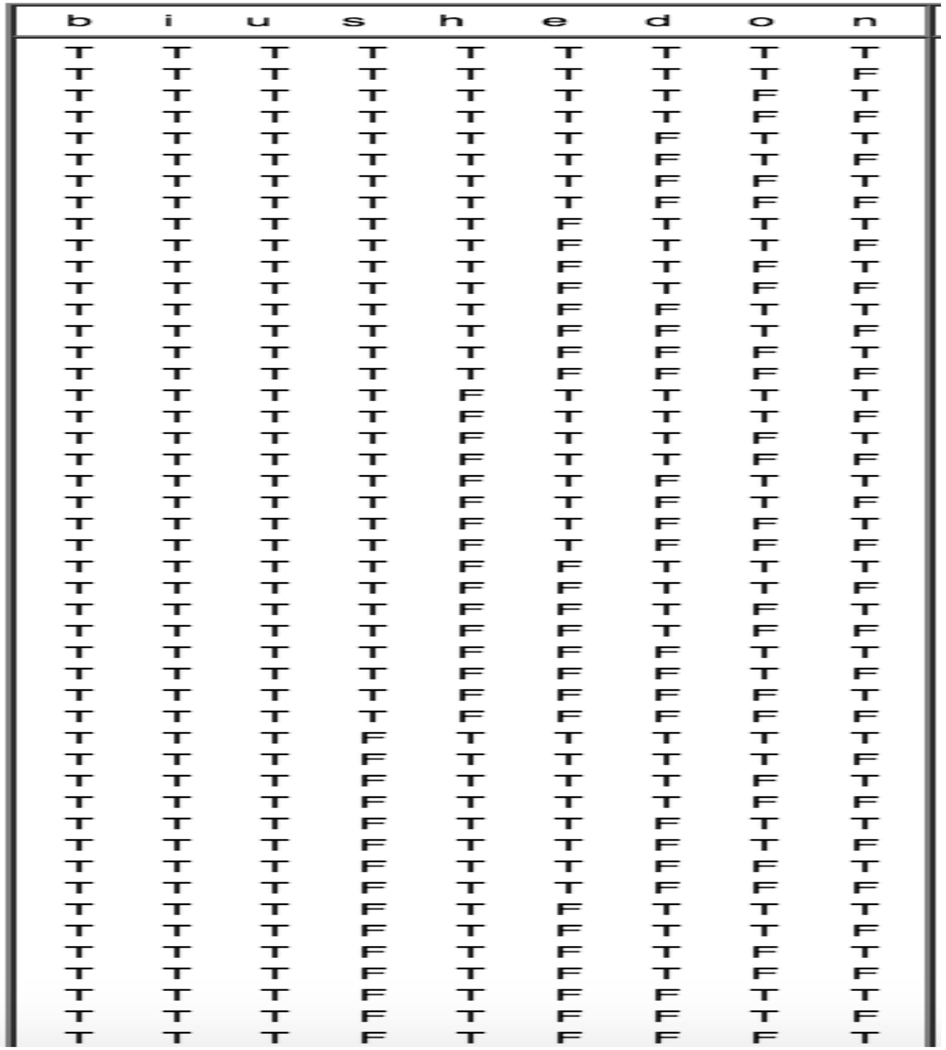
- Plain text
- Superscript
- Bold
- ~~Italic and strikethrough~~
- Bold and underlined
- ~~Bold italic strikethrough shadowed superscript~~

How many tests would you need to test all the possible font combinations?

$$2^{10}$$

1,024 tests!

That’s quite a few tests…



b	i	u	s	h	e	d	o	n
T	T	T	T	T	T	T	T	T
T	T	T	T	T	T	T	T	F
T	T	T	T	T	T	T	F	T
T	T	T	T	T	T	T	F	F
T	T	T	T	T	T	F	T	T
T	T	T	T	T	T	F	T	F
T	T	T	T	T	T	F	F	T
T	T	T	T	T	T	F	F	F
T	T	T	T	T	F	T	T	T
T	T	T	T	T	F	T	T	F
T	T	T	T	T	F	T	F	T
T	T	T	T	T	F	T	F	F
T	T	T	T	T	F	F	T	T
T	T	T	T	T	F	F	T	F
T	T	T	T	T	F	F	F	T
T	T	T	T	T	F	F	F	F
T	T	T	T	F	T	T	T	T
T	T	T	T	F	T	T	T	F
T	T	T	T	F	T	T	F	T
T	T	T	T	F	T	T	F	F
T	T	T	T	F	T	F	T	T
T	T	T	T	F	T	F	T	F
T	T	T	T	F	T	F	F	T
T	T	T	T	F	T	F	F	F
T	T	T	T	F	F	T	T	T
T	T	T	T	F	F	T	T	F
T	T	T	T	F	F	T	F	T
T	T	T	T	F	F	T	F	F
T	T	T	T	F	F	F	T	T
T	T	T	T	F	F	F	T	F
T	T	T	T	F	F	F	F	T
T	T	T	T	F	F	F	F	F
T	T	T	F	T	T	T	T	T
T	T	T	F	T	T	T	T	F
T	T	T	F	T	T	T	F	T
T	T	T	F	T	T	T	F	F
T	T	T	F	T	T	F	T	T
T	T	T	F	T	T	F	T	F
T	T	T	F	T	T	F	F	T
T	T	T	F	T	T	F	F	F
T	T	T	F	T	F	T	T	T
T	T	T	F	T	F	T	T	F
T	T	T	F	T	F	T	F	T
T	T	T	F	T	F	T	F	F
T	T	T	F	T	F	F	T	T
T	T	T	F	T	F	F	T	F
T	T	T	F	T	F	F	F	T

But it’s necessary! What if…

… a problem only occurs with 3-D shadowed bold italic superscript text?

That’s going to be hard to find.

Turns Out Other People Have Thought About This!

The National Institute of Standards and Technology did a study on the topic.

See: "Practical Combinatorial Testing", http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-142.pdf

Turns Out That’s Unlikely!

- Think of each font effect as a Boolean variable (e.g. bold vs not bold, italic vs non-italic)
- Most (50 - 90%, depending on the project) defects come from combinations of one or two interactions (variables).
- In other words, most defects would be found if you just tested, e.g., "bold 3-D" (two interactions) text or just "bold text" (one interactions).

Similar Distribution Found In Many Domains

- Web browser
- Avionics software
- Telecommunications software
- Flight Traffic Control
- Network security software

The Interaction Rule

"Most failures are triggered by one or two parameters, and progressively fewer by three, four, or more parameters, and the maximum interaction degree is small." -Eric Kuhn, NIST

The Interaction Rule

- The maximum number of interactions found to cause a defect was SIX, no matter the value of *n* (# of variables).
- This was after an analysis of dozens of software projects.

So…

- So we can find a large percentage of defects with minimal work by making sure we test all possible pairs of values.

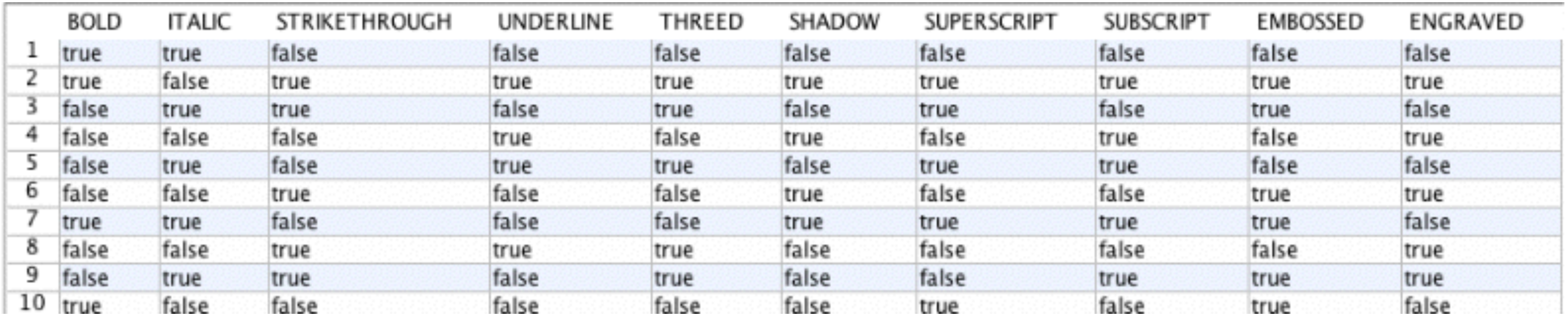
Pairwise Testing

- This is called “pairwise”, or “all-pairs” testing.
- We are testing all possible pairs of interactions, e.g.:
  - Not-Bold / Not-Italic
  - Bold / Not-Italic
  - Not-Bold / Italic
  - Bold / Italic

Remember our exhaustive 10-font-effect testing plan?

- It was 2 ^ n, thus 1,024 (2 ^ 10) tests.
- How many tests would it require to test all pairs of interactions?
  - That is, all possible combinations of:
    - bold/italic,
    - subscript/bold
    - underline/strikethrough
    - 3-D / italic
    - Every possible pairing of two variables

Answer: 10



	BOLD	ITALIC	STRIKETHROUGH	UNDERLINE	THREED	SHADOW	SUPERSCRIPT	SUBSCRIPT	EMBOSSED	ENGRAVED
1	true	true	false	false	false	false	false	false	false	false
2	true	false	true	true	true	true	true	true	true	true
3	false	true	true	false	true	false	true	false	true	false
4	false	false	false	true	false	true	false	true	false	true
5	false	true	false	true	true	false	true	true	false	false
6	false	false	true	false	false	true	false	false	true	true
7	true	true	false	false	false	true	true	true	true	false
8	false	false	true	true	true	false	false	false	false	true
9	false	true	true	false	true	false	false	true	true	true
10	true	false	false	false	false	false	true	false	true	false

Reduce Number of Tests By Two Orders Of Magnitude

Is This Always Good Enough?

Of course not

- But we can “dial up” the number of possible interactions to check for any *t*
- For example, check every three-way combination (t = 3):
  - Bold / Italic / Underline
  - Italic / Underline / Superscript
  - Shadow / Italic / Bold
- Or four-way (t = 4)
  - Bold / Italic / Underline / Superscript
  - Embossed / 3-D / Outline / Strikethrough
  - Shadow / Bold / Inverse / Outline
- Up to *n* (the number of interactions) - This would be the same as exhaustive testing

Combinatorial Testing

- This generalized version of pairwise testing is known as “combinatorial testing”
- Note that pairwise testing is technically just a specific kind of combinatorial testing where $t = 2$

Combinatorial Testing Example

- The maximum number of interactions causing a defect found in the NIST studies was six. So let's test all six-way combinations of our font effects.
- Recall that:
  - # tests required for full pairwise testing was 10
  - # tests required for exhaustive testing was 1,024
  - How many to test all six-way interactions?

Actually a difficult question to answer off the top of your head

- Determining the exact number necessary is an NP-Hard problem.
- But there are some good algorithms out there that approximate it (e.g. IPOG).
- See “IPOG: A General Strategy for T-Way Software Testing” http://csrc.nist.gov/acts/ecbs-cr-final.pdf

… and the answer is…

- The best answer my software could come up with is 178.
- Approximately an order of magnitude less than exhaustive testing!
- But in any piece of software tested by NIST, would have found the same number of defects

Interesting!

- 10 tests catch 90% of defects
- 178 tests catch ~99.9999999% of defects
- 1024 tests catch ~100% of defects

IF THEY ARE DONE RIGHT!

Recap

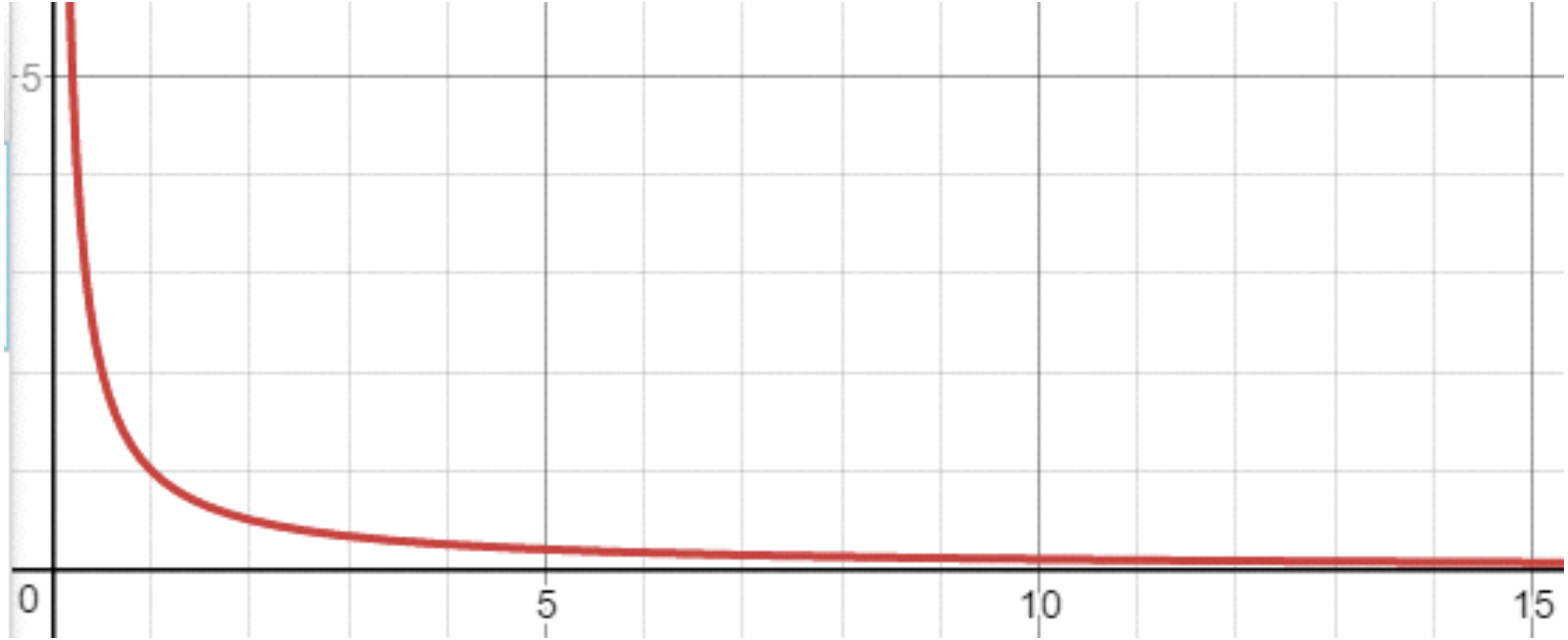
- 10 tests catch 90% of defects
- 178 tests catch ~99.9999999% of defects
- 1024 tests catch ~100% of defects

IF THEY ARE DONE RIGHT!

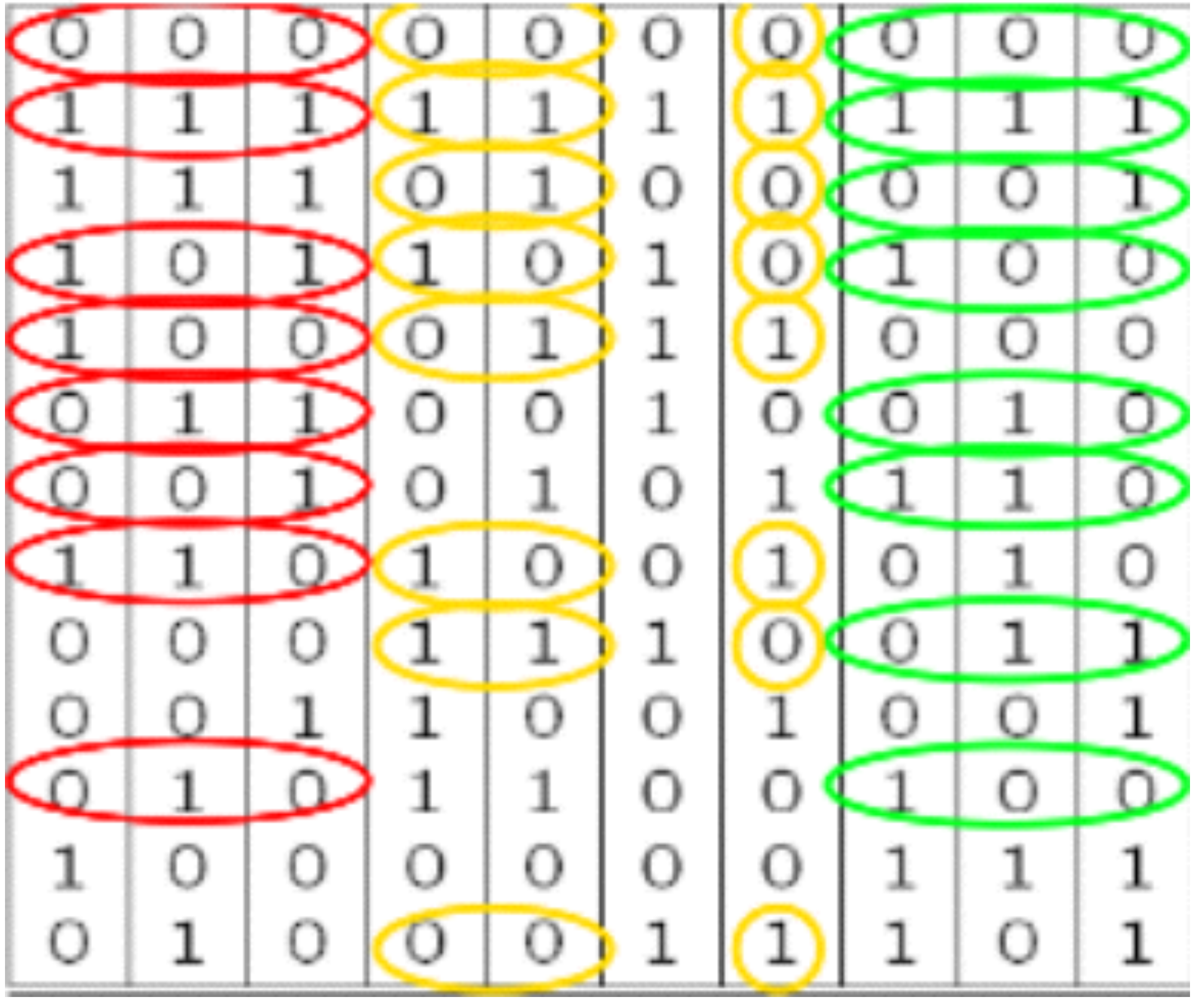
It Gets Harder the Closer You Get

- You can see how much more expensive it becomes to test depending on how arbitrarily close to "100% free of defects" you want to be.
- It is NOT a linear relationship.
- It is asymptotic.

Example: f(x) = 1 / x



Covering Arrays



0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1
1	1	1	0	1	0	0	0	0	1
1	0	1	1	0	1	0	1	0	0
1	0	0	0	1	1	1	0	0	0
0	1	1	0	0	1	0	0	1	0
0	0	1	0	1	0	1	1	1	0
1	1	0	1	0	0	1	0	1	0
0	0	0	1	1	1	0	0	1	1
0	0	1	1	0	0	1	0	0	1
0	1	0	1	1	0	0	1	0	0
1	0	0	0	0	0	0	1	1	1
0	1	0	0	0	1	1	1	0	1

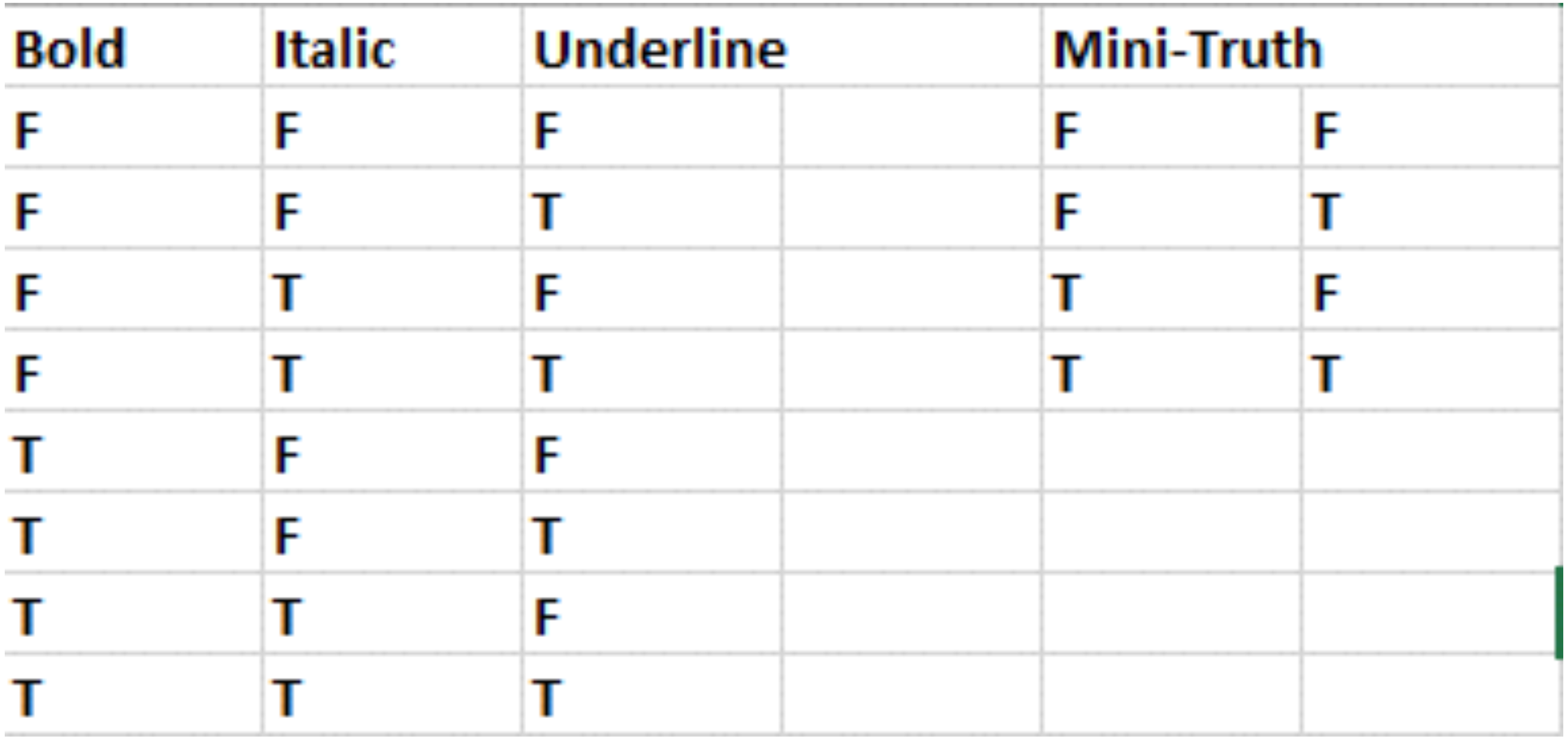
Steps To Make Your Own Covering Array

- Make a truth table with all variables
  - Each line in truth table indicates a test
  - Running all these tests would be an exhaustive test
- Make a list of all t-way interactions for desired t
  - Example: Bold, Italic, Underline. t = 2
    - Bold / Italic
    - Bold /Underline
    - Italic/Underline

Generating Covering Arrays

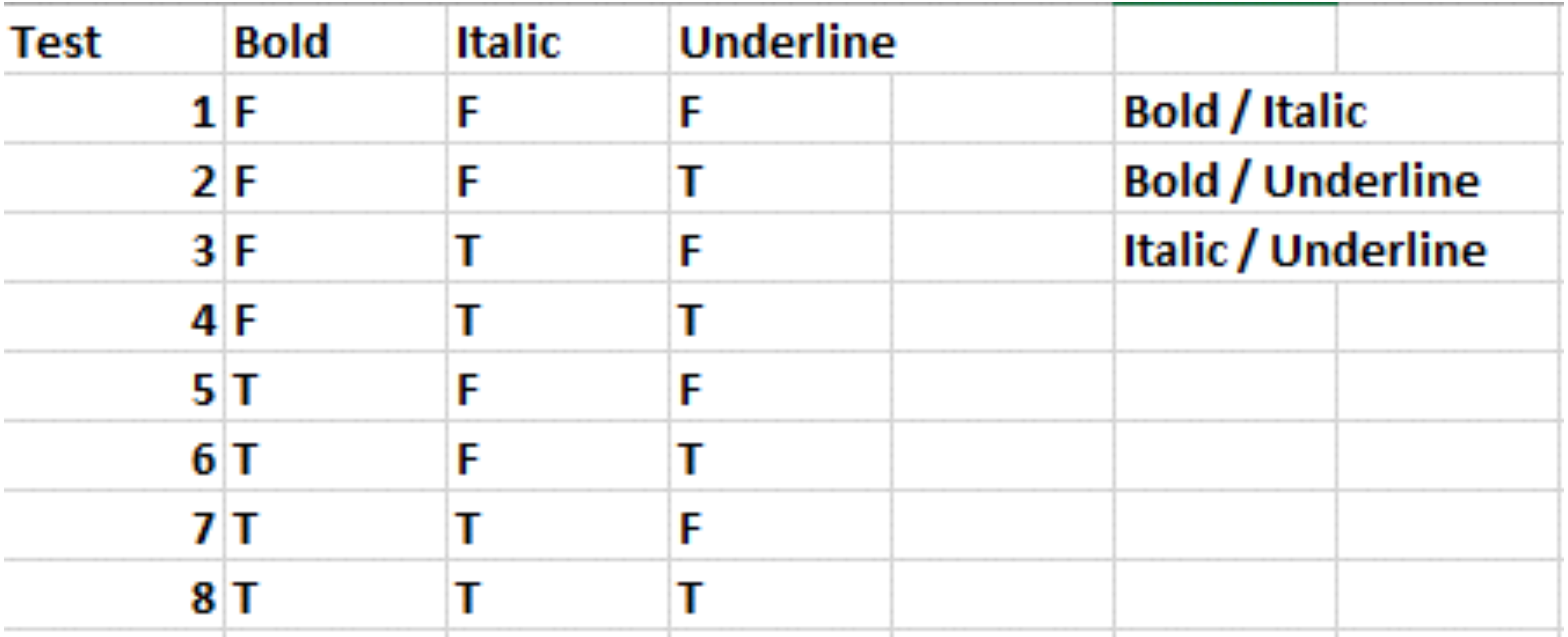
- Look for tests which make a complete truth table for each t-way interaction
- Mark these tests as “Tests To Be Executed”
- Continue adding t-way interactions tests
  - Prefer using tests which are already scheduled to be executed
- When all t-way interaction “mini truth tables” have been completed, put together all tests to be executed

Covering Array Example



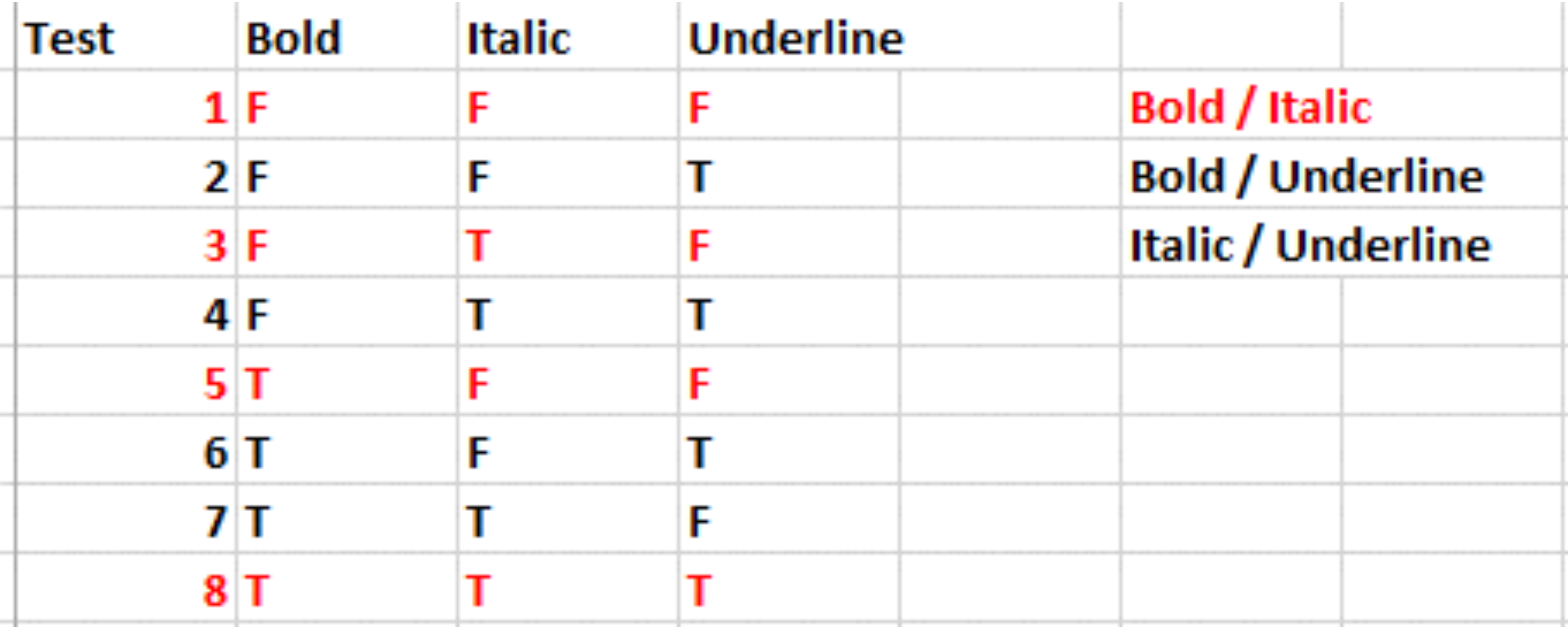
Bold	Italic	Underline		Mini-Truth	
F	F	F		F	F
F	F	T		F	T
F	T	F		T	F
F	T	T		T	T
T	F	F			
T	F	T			
T	T	F			
T	T	T			

Covering Array Example



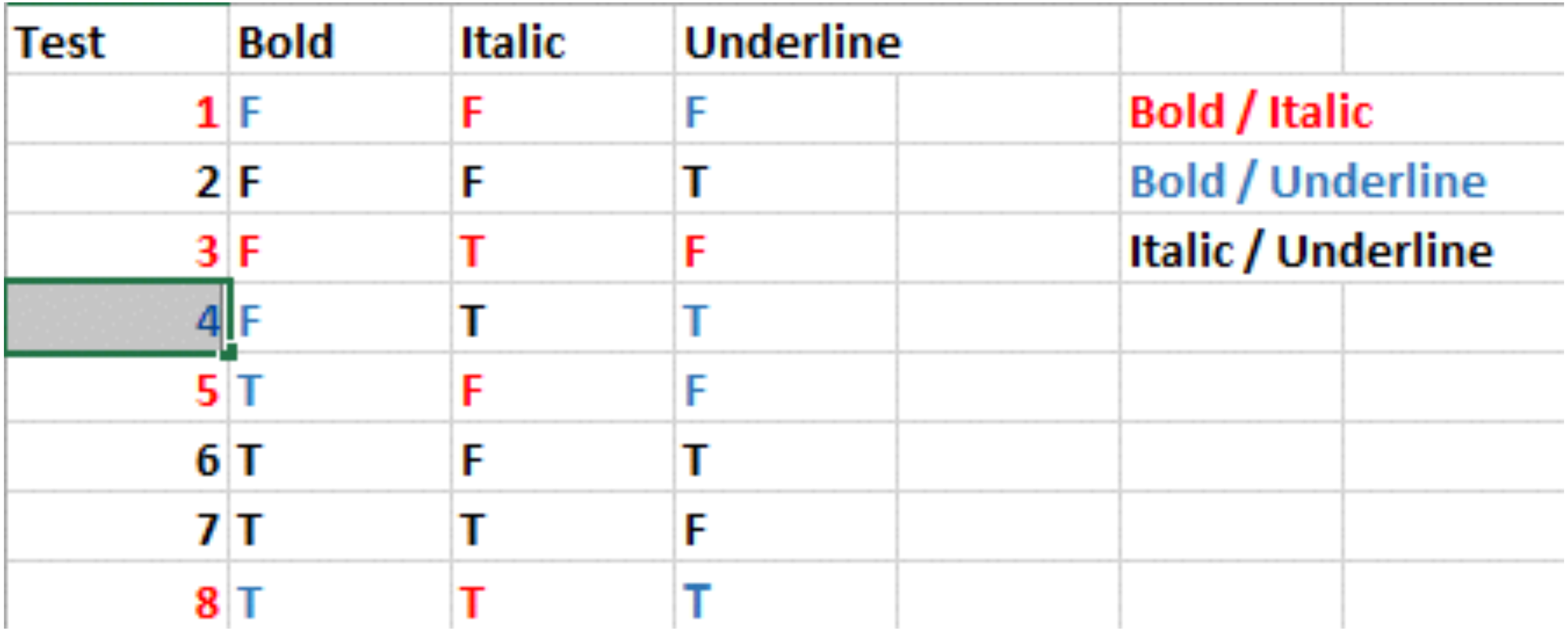
Test	Bold	Italic	Underline			
1	F	F	F		Bold / Italic	
2	F	F	T		Bold / Underline	
3	F	T	F		Italic / Underline	
4	F	T	T			
5	T	F	F			
6	T	F	T			
7	T	T	F			
8	T	T	T			

Covering Array Example – Bold / Italic



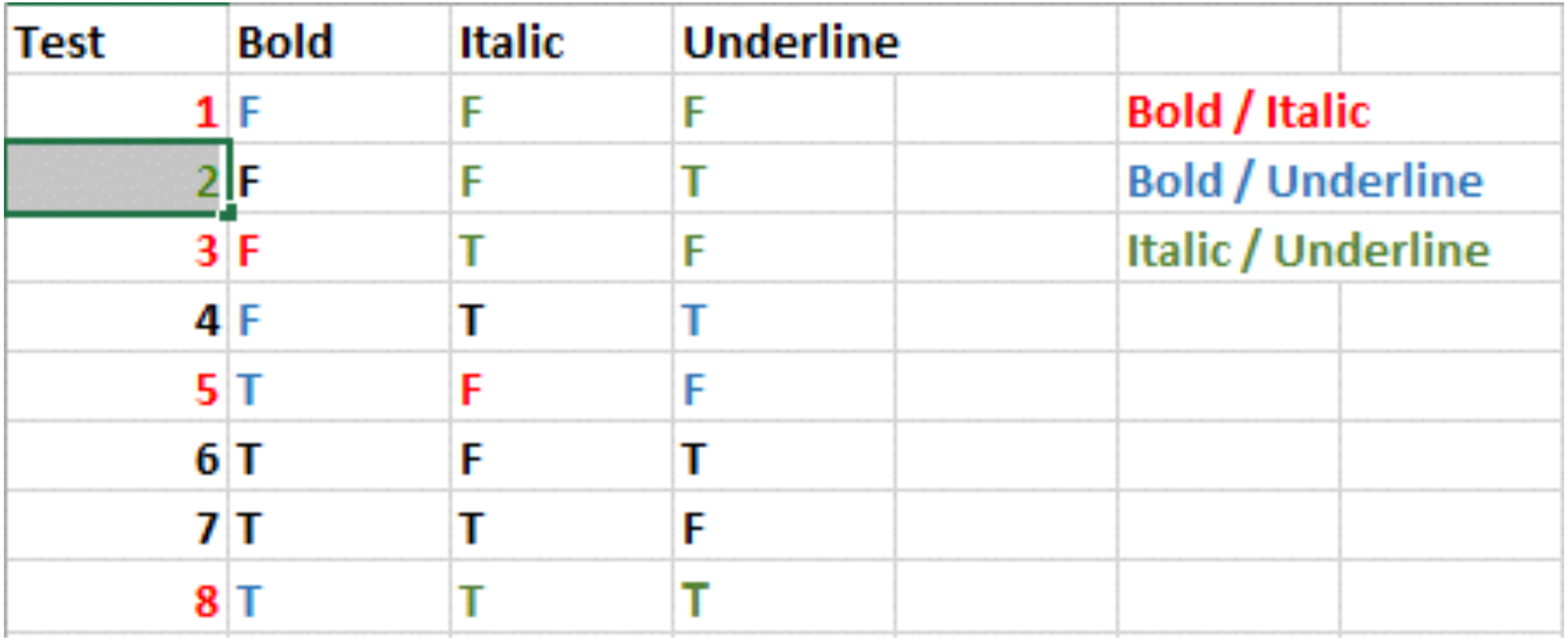
Test	Bold	Italic	Underline		
1	F	F	F		Bold / Italic
2	F	F	T		Bold / Underline
3	F	T	F		Italic / Underline
4	F	T	T		
5	T	F	F		
6	T	F	T		
7	T	T	F		
8	T	T	T		

Covering Array Example – Bold / Underline



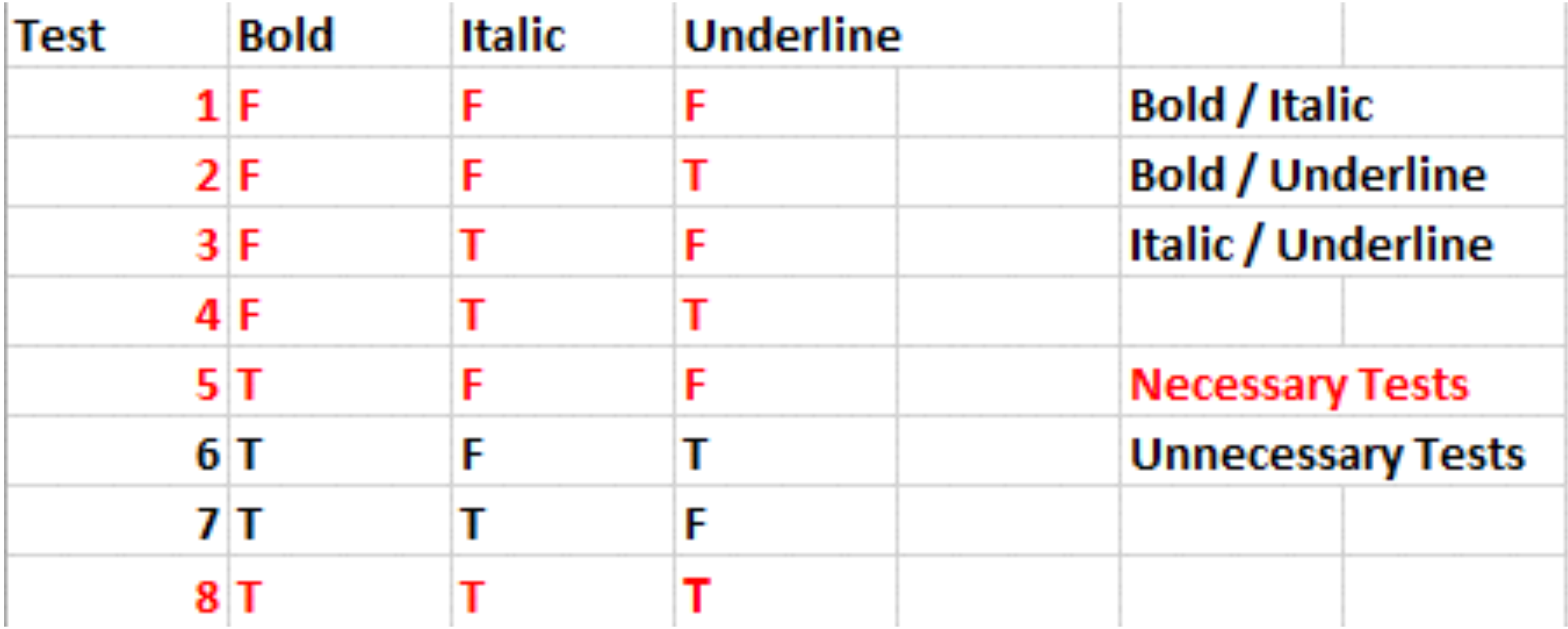
Test	Bold	Italic	Underline		
1	F	F	F		Bold / Italic
2	F	F	T		Bold / Underline
3	F	T	F		Italic / Underline
4	F	T	T		
5	T	F	F		
6	T	F	T		
7	T	T	F		
8	T	T	T		

Covering Array Example – Italic / Underline



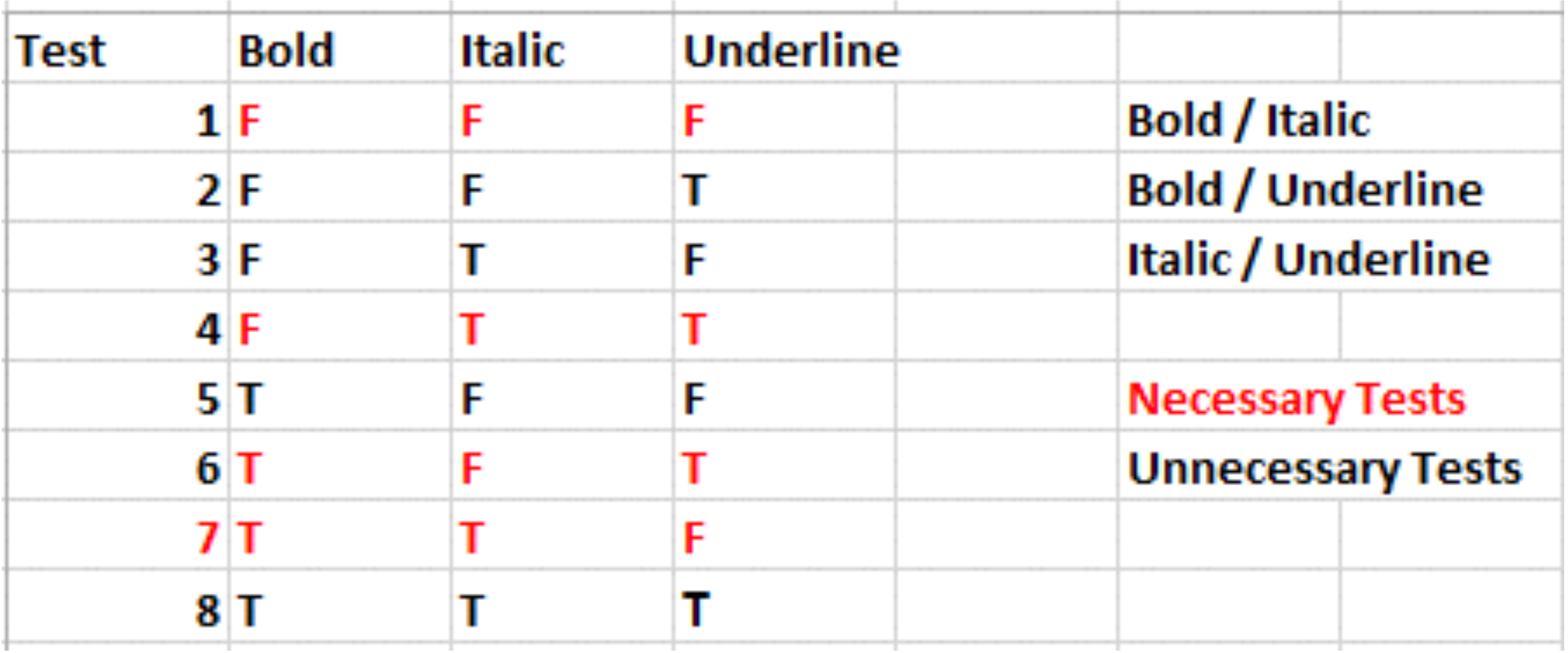
Test	Bold	Italic	Underline		
1	F	F	F		Bold / Italic
2	F	F	T		Bold / Underline
3	F	T	F		Italic / Underline
4	F	T	T		
5	T	F	F		
6	T	F	T		
7	T	T	F		
8	T	T	T		

Run a Subset of Tests



Test	Bold	Italic	Underline		
1	F	F	F		Bold / Italic
2	F	F	T		Bold / Underline
3	F	T	F		Italic / Underline
4	F	T	T		
5	T	F	F		Necessary Tests
6	T	F	T		Unnecessary Tests
7	T	T	F		
8	T	T	T		

Can Minimize Further Using “Intuition” Or Better Algorithms



Test	Bold	Italic	Underline		
1	F	F	F		Bold / Italic
2	F	F	T		Bold / Underline
3	F	T	F		Italic / Underline
4	F	T	T		
5	T	F	F		Necessary Tests
6	T	F	T		Unnecessary Tests
7	T	T	F		
8	T	T	T		

OK, this works for small numbers of variables, but what about big ones?

- Imagine a 34-variable system
  - Exhaustive testing: 17 billion tests
  - All 3-way interactions: 33 tests
  - All 4-way interactions: 85 tests
- Actually gets BETTER the higher the number of variables
- Not just a little better – many orders of magnitude better

Remember at the beginning of the term when I talked about the impossibility of testing every combination of inputs?

This is a possible amelioration.

Won’t It Take a Long Time To Make Covering Arrays For Large Values of *n* or *t*?

YES

- These are not artisanal, hand-crafted arrays, carved by the European masters high in their Swiss valleys
- Let’s use a program to do it
- Example: NIST ACTS