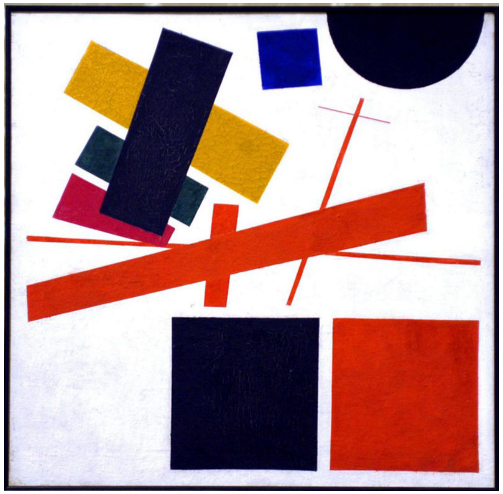
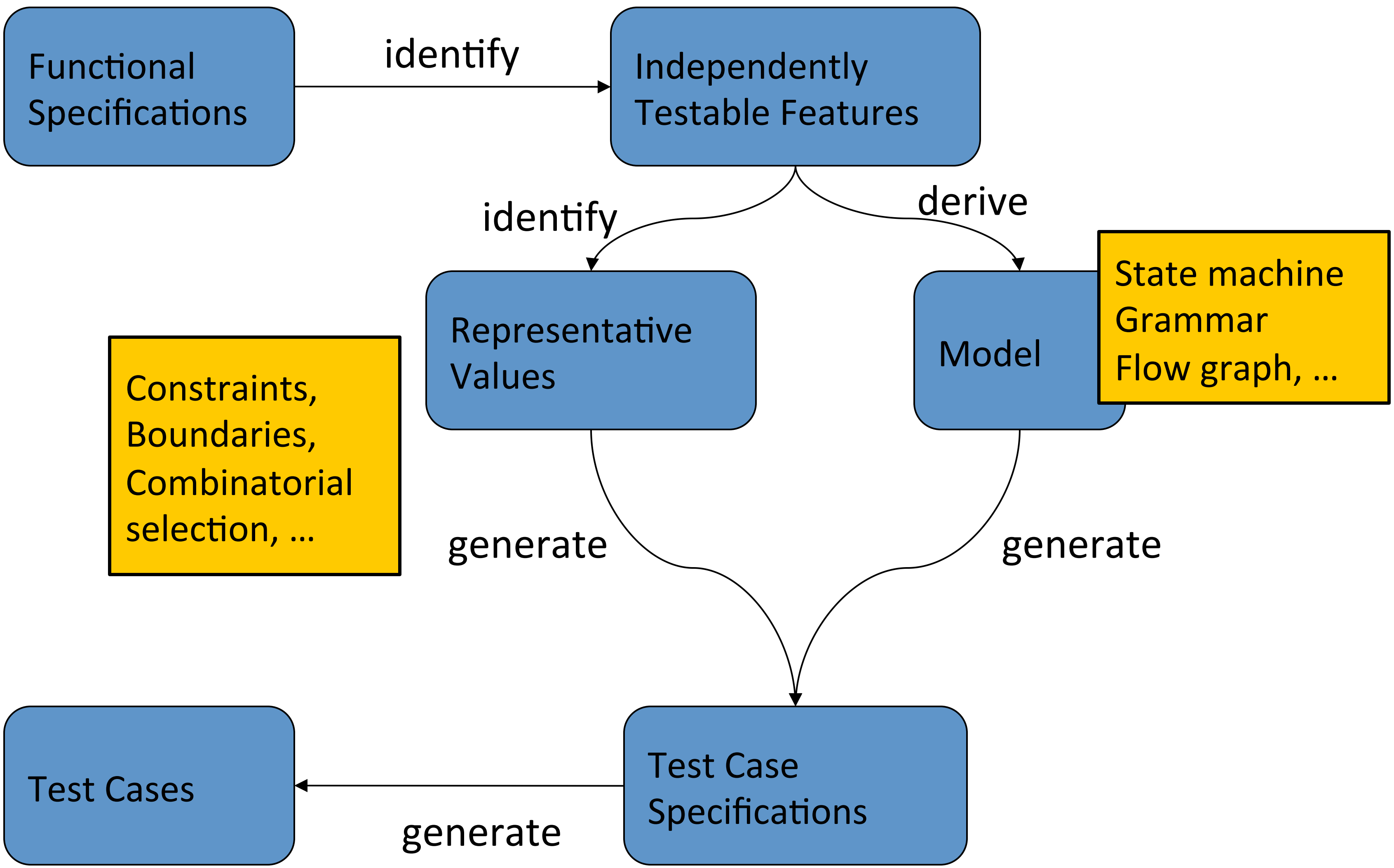
CPEN 422
Software Testing and Analysis



Combinatorial Testing

Systematic *Functional* Testing

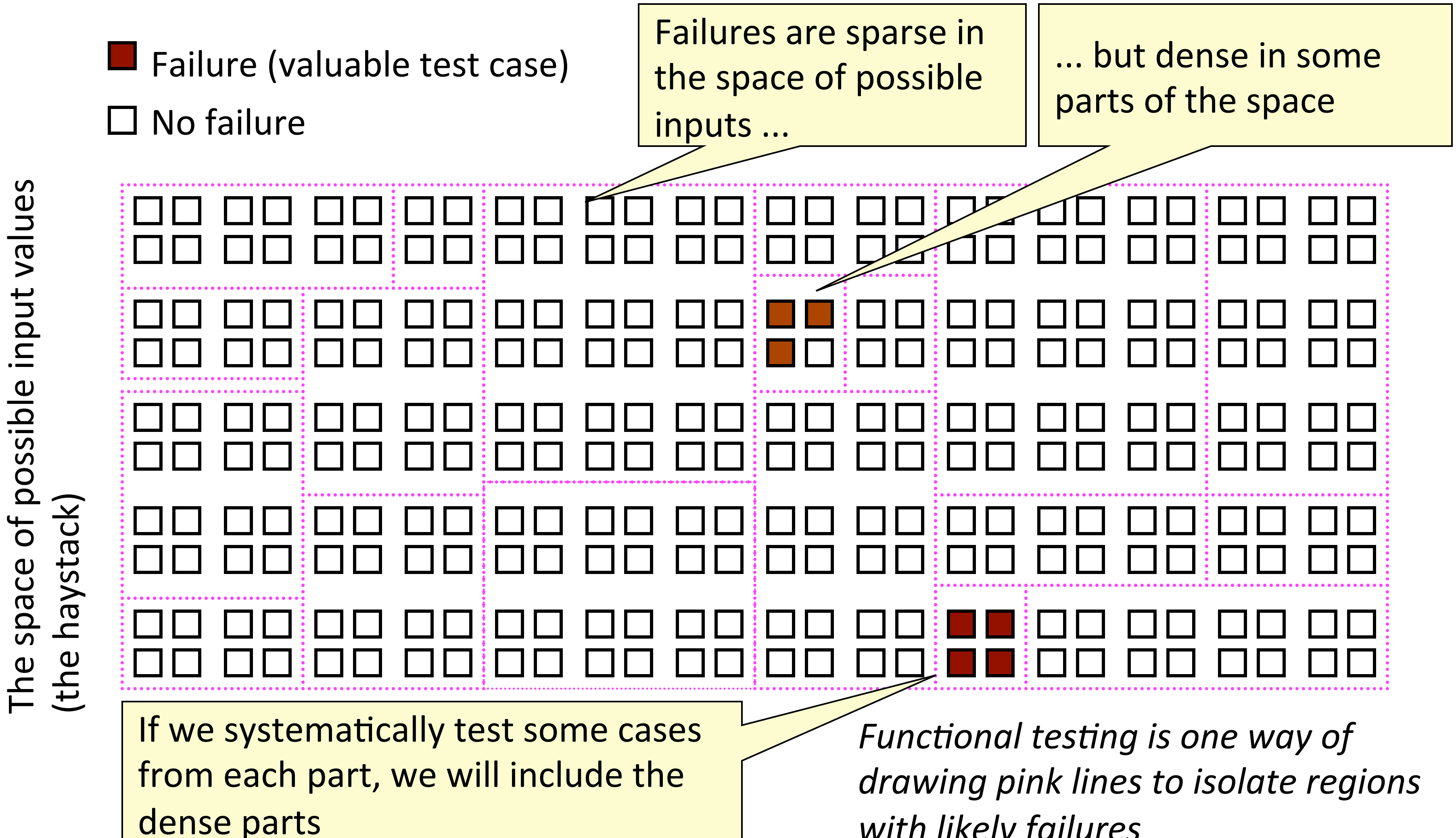


Huge input space

- How to choose inputs in testing?

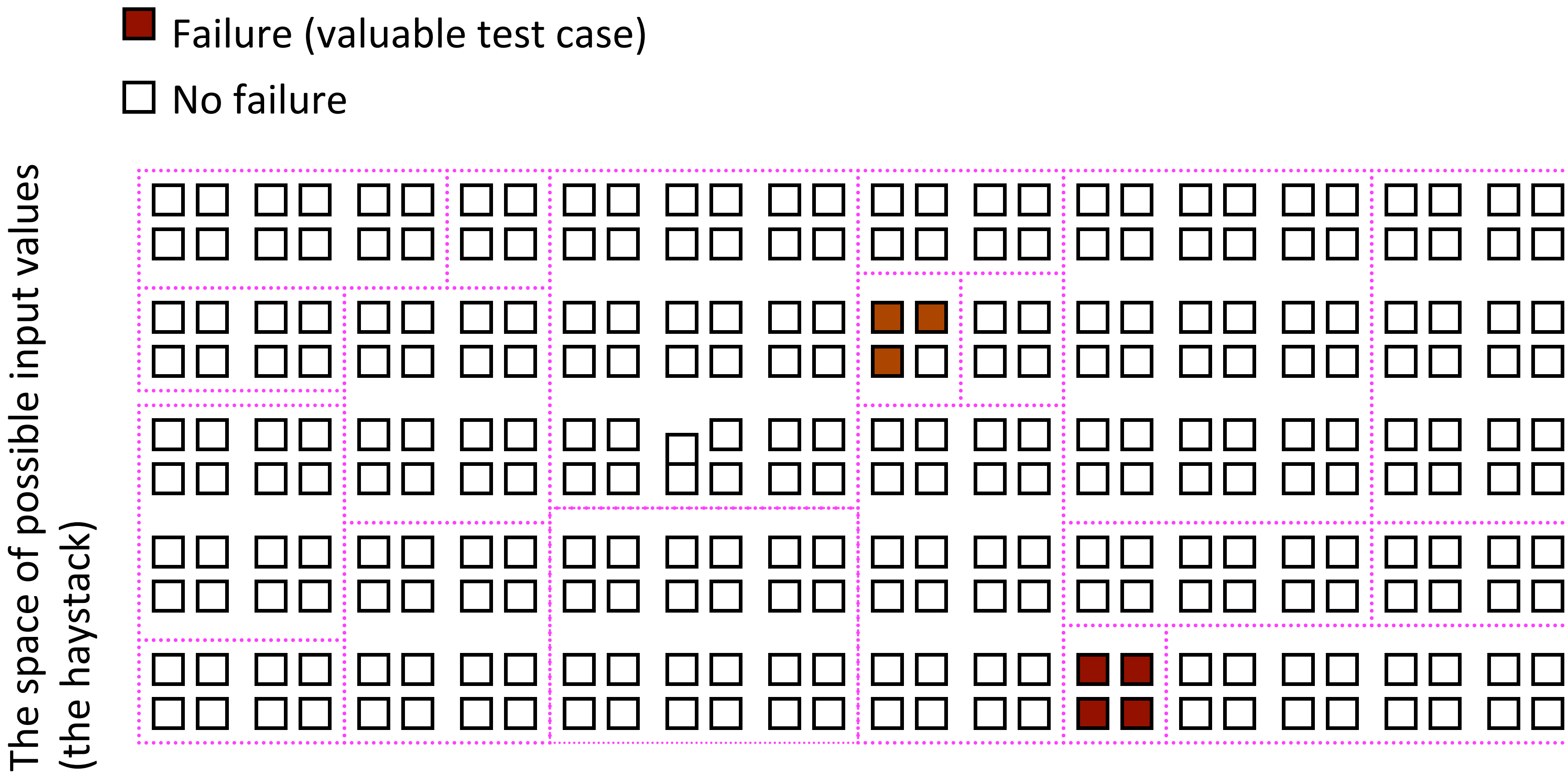
- What is equivalence partitioning?

Systematic Equivalence Partitioning



Functional testing is one way of drawing pink lines to isolate regions with likely failures

Combinations of class values?



Ch. 11: Combinatorial Testing

- What to do about the combinatorial explosion?

Partitioning helps to reduce the problem: we get the classes.
But, for each class multiple values? And then the combinations?

Example

Three parameters:

- **OS**: Linux, Windows, MacOS (3 values)
- **DB Server**: MySQL, Oracle (2 values)
- **Web Server**: IIS, Apache (2 values)

How many combinations to test?

3 * 2 * 2 = 12

What if we have a program with 10 parameters each with 8 possible values?

8^{10}

Input Handling / Combinational Testing

www.google.ca/advanced_search

Google

Advanced Search

Find pages with... — **To do this in the search box.**

all these words:		Type the important words: `tri-colour rat terrier`
this exact word or phrase:		Put exact words in quotes: `"rat terrier"`
any of these words:		Type OR between all the words you want: `miniature OR standard`
none of these words:		Put a minus sign just before words that you don't want: `-rodent, -"Jack Russell"`
numbers ranging from:	to	Put two full stops between the numbers and add a unit of measurement: `10..35 kg, £300..£500, 2010..2011`

Then narrow your results by...

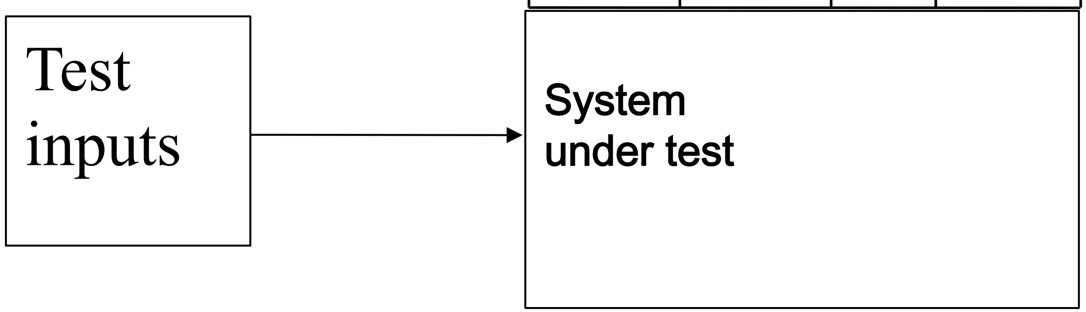
language:	any language	Find pages in the language that you select.
region:	any region	Find pages published in a particular region.
last update:	anytime	Find pages updated within the time that you specify.
site or domain:		Search one site (like `wikipedia.org`) or limit your results to a domain like `.edu`, `.org` or `.gov`
terms appearing:	anywhere in the page	Search for terms in the whole page, page title or web address, or links to the page you're looking for.
SafeSearch:	Show most relevant results	Tell SafeSearch whether to filter sexually explicit content.
reading level:	no reading level displayed	Find pages at one reading level or just view the level info.
file type:	any format	Find pages in the format that you prefer.
usage rights:	not filtered by licence	Find pages that you are free to use yourself.

Advanced Search

Two scopes of combinatorial testing

Test Configurations

Test case	OS	CPU	Protocol
1	Windows	Intel	IPv4
2	Windows	AMD	IPv6
3	Linux	Intel	IPv6
4	Linux	AMD	IPv4



1. Identify parameters

   Inputs, environment

   Affect behavior

2. Identify input characteristics

   Major properties of each parameter

3. Identify selected *combinations* of inputs

SPECIAL ARTICLE

THE CATEGORY-PARTITION METHOD FOR SPECIFYING AND GENERATING FUNCTIONAL TESTS

A method for creating functional test suites has been developed in whic[h] test engineer analyzes the system specification, writes a series of formal specifications, and then uses a generator tool to produce test description[s] from which test scripts are written. The advantages of this method are t[hat] the tester can easily modify the test specification when necessary, and c[an] control the complexity and number of the tests by annotating the tests specification with constraints.

THOMAS J. OSTRAND and MARC J. BALCER

The goal of functional testing of a software system is to find discrepancies between the actual behavior of the implemented system's functions and the desired behavior as described in the system's functional specification. To achieve this goal requires first, that tests be executed for all of the system's functions, and second, that the tests be designed to maximize the chances of finding errors in the software. Although a particular method or testing group may emphasize one or the other, these two aspects of testing are mutually complementary, and both are necessary for maximally productive testing. It is not enough merely to "cover all the functionality"; the tests must be aimed at the most vulnerable parts of the implementation. For functional testing, this implies testing boundary conditions, special cases, error handlers, and cases where inputs and the system environment interact in potentially dangerous ways. Conversely, it is not enough to write excellent, error-exposing tests for only some of a system's functions, or to write tests aimed only at certain types of errors or certain characteristics of a specification or implementation.

Functional tests can be derived from the softwa[re] specifications, from design information, or from th[e] code itself. All three test sources provide useful in[for]mation, and none of them should be ignored. Code[-]based tests relate to the modular structure, logic, [con]trol flow, and data flow of the software. They hav[e] particular advantage that a program is a formal ob[ject] and it is therefore easy to make precise statement[s] about the adequacy or thoroughness of code-base[d] tests. Design-based tests relate to the programmin[g ab]stractions, data structures, and algorithms used to [con]struct the software. Specification-based tests relat[e di]rectly to what the software is supposed to do, and therefore are probably the most intuitively appeal[ing] type of functional tests.

A standard approach to generating specification[-]based functional tests is first to partition the input [do]main of a function being tested, and then to select data from each class of the partition. The idea is t[hat] elements within an equivalence class are essentia[lly] the same for the purposes of testing. If the testing'[s] main emphasis is to attempt to show the presence [of] errors, then the assumption is that any element of [a] class will expose the error as well as any other on[e. If] the testing's main emphasis is to attempt to give c[onfi]dence in the software's correctness, then the assum[p]tion is that correct results for a single element in a will provide confidence that all elements in the cl[ass] would be processed correctly.

A preliminary version of this paper appeared as "Machine-Aided Production of Software Tests," in the Proceedings of the Fifth Annual Pacific Northwest Software Quality Conference, Portland, Oregon, October 19–20, 1987.

Category-Partition: Summary

11.2

1. [A] Identify testable features
2. [A] Identify inputs per feature
3. [A] Identify categories / characteristics per input
4. [B] Identify choices per category
5. [C] Identify constraints between choices
6. [C] Generate remaining combinations, resulting in test specification table [Table 11.2]
7. Turn into test cases

Pairwise Combinatorial Testing

11.3

Three parameters:

- OS: Linux, Windows, MacOS (3 values)
- DB Server: MySQL, Oracle (2 values)
- Web Server: IIS, Apache, (2 values)

Are full 3-way combinations responsible for failures?

Or are specific pairs causing trouble?

Pairwise combinatorial testing

- Category partition works well when intuitive constraints reduce the number of combinations to a small amount of test cases
  - Without many constraints, the number of combinations may be unmanageable

- Pairwise combination (instead of exhaustive)
  - Generate combinations that efficiently cover all pairs (or triples,…) of classes
  - **Rationale**: most failures are triggered by **single values** or **combinations of a few values**. Covering pairs (triples,…) reduces the number of test cases, but reveals most faults

Partitioned classes: OS, DB, Server

Three parameters:

- **OS**: Linux, Windows, MacOS (3 values)
- **DB Server**: MySQL, Oracle (2 values)
- **Web Server**: IIS, Apache, (2 values)

1. Full combination: 3*2*2 = 12
2. Pair-wise: multiplying the **two largest values**
   3*2 = 6 cases

11.3

Let’s cover just the pairs

OS	DB	Web

11.3

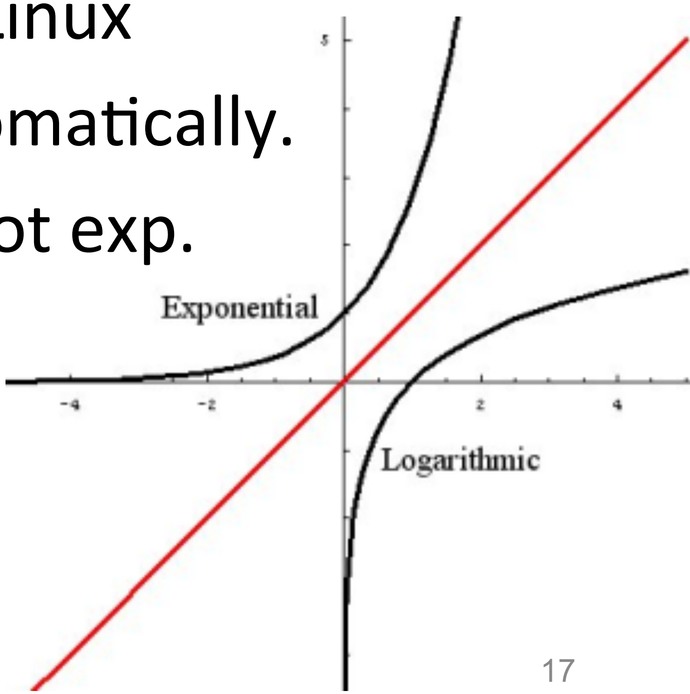
Let’s cover just the pairs

OS	DB	Web
Linux	MySQL	IIS
Linux	Oracle	Apache
Windows	MySQL	Apache
Windows	Oracle	IIS
MacOS	MySQL	Apache
MacOS	Oracle	IIS

6 test cases instead of full 3*2*2=12.

Pairwise Combination Testing

- Step 0: Eliminate invalid combinations
  - IIS only runs on Windows
- Key step: **Cover all *pairs* of combinations**
  - such as MySQL on Windows & Linux
- Generate / test combinations automatically.
- #pairwise grows logarithmically, not exp.



updated

OS	DB	Web
Linux	MySQL	~~IIS~~ (Apache)
Linux	Oracle	Apache
Windows	MySQL	~~Apache~~ (IIS)
Windows	Oracle	IIS
MacOS	MySQL	Apache
MacOS	Oracle	~~IIS~~ (Apache)

Another Example: Display Control

- **How many test cases to cover fully?**
- **How many test cases to cover pairwise?**

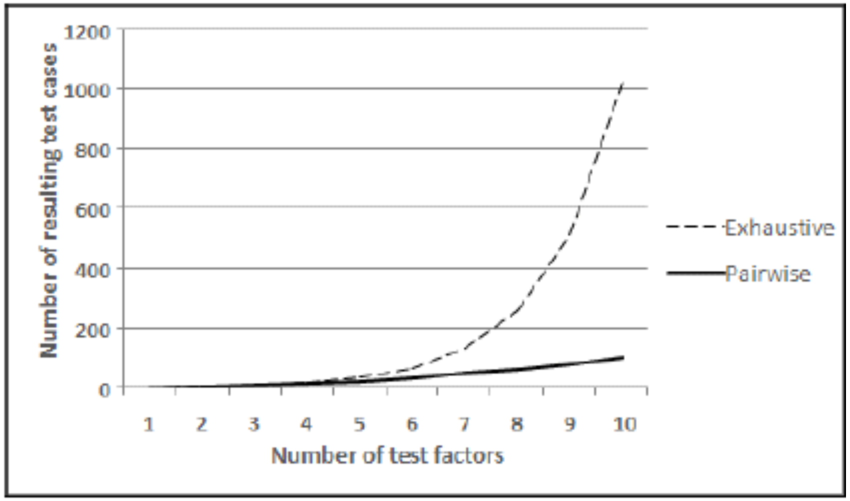
Display Mode	Language	Fonts	Color	Screen size
full-graphics	English	Minimal	Monochrome	Hand-held
text-only	French	Standard	Color-map	Laptop
limited-bandwidth	Spanish	Document-loaded	16-bit	Full-size
	Portuguese		True-color	

1. Full combinations: **432** (3x4x3x4x3) test cases
2. Pairwise: **16** (4 X 4)

Pairwise combinations: 16 test cases (4*4)

Language	Color	Display Mode	Fonts	Screen Size
English	Monochrome	Full-graphics	Minimal	Hand-held
English	Color-map	Text-only	Standard	Full-size
English	16-bit	Limited-bandwidth	-	Full-size
English	True-color	Text-only	Document-loaded	Laptop
~~French~~	~~**Monochrome**~~	~~Limited-bandwidth~~	~~Standard~~	~~**Laptop**~~
French	Color-map	Full-graphics	Document-loaded	Full-size
French	16-bit	Text-only	Minimal	-
French	True-color	-	-	Hand-held
Spanish	Monochrome	-	Document-loaded	Full-size
Spanish	Color-map	Limited-bandwidth	Minimal	Hand-held
Spanish	16-bit	Full-graphics	Standard	Laptop
Spanish	True-color	Text-only	-	Hand-held
Portuguese	-	-	Monochrome	Text-only
Portuguese	Color-map	-	Minimal	Laptop
Portuguese	16-bit	Limited-bandwidth	Document-loaded	Hand-held
Portuguese	True-color	Full-graphics	Minimal	Full-size
Portuguese	True-color	Limited-bandwidth	Standard	Hand-held

Microsoft PICT




```
Type:            Single, Spanned, Striped, Mirror, RAID-5
Size:            10, 100, 1000, 10000, 40000
Format method:   Quick, Slow
File system:     FAT, FAT32, NTFS
Cluster size:    512, 1024, 2048, 4096, 8192, 16384
Compression:     On, Off

# There are limitations on volume size

IF [File system] = "FAT"   THEN [Size] <= 4096;
IF [File system] = "FAT32" THEN [Size] <= 32000;

# And not all file systems support compression

IF [File system] <> "NTFS" or
  ([File system] =  "NTFS" and [Cluster size] > 4096)
THEN [Compression] = "Off";
```

Test domain consisting of ‘input’ and ‘environment’ parameters:

```
# Input parameters

Type:            Single, Spanned, Striped, Mirror, RAID-5
Size:            10, 100, 1000, 10000, 40000
Format method:   Quick, Slow
File system:     FAT, FAT32, NTFS
Cluster size:    512, 1024, 2048, 4096, 8192, 16384
Compression:     On, Off

# Environment parameters

Platform:  x86, x64, ia64
CPUs:      1, 2
RAM:       1GB, 4GB, 64GB

# Environment parameters will form a sub-model

{ PLATFORM, CPUS, RAM } @ 2
```

Hierarchy of test parameters:

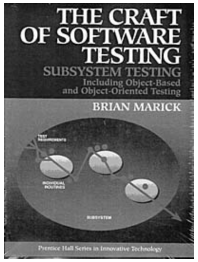
```
- Type
- Size
- Format method
- File system
- Cluster size
- Compression
- <CompoundParameter> (t=2)
   - Platform
   - CPUs
   - RAM
```

Exercise: pair-wise testing

- Suppose we want to demonstrate that a new software application works correctly on machines that use the **Windows**, **Mac**, and **Linux** operating systems, **Intel** or **AMD** processors, and the **IPv4** or **IPv6** protocols.

- How many pair-wise tests are needed! Which?

OS	Processor	Protocol
Linux	Intel	IPv4
Linux	AMD	IPv6
Windows	Intel	IPv6
Windows	AMD	IPv4
Mac	Intel	IPv4
Mac	AMD	IPv6



11.4

Catalog-Based Testing

- Capture the experience of test designers
  - cases to be considered
  - For different types of variables
  - inputs, outputs, status of computation

- Gather experience in systematic collection:
  - speed up test design
  - routinize decisions / focus human effort
  - accelerate training / reduce human error

Catalog based testing process

Step1:

Analyze the initial specification to identify simple elements:

- Pre-conditions
- Post-conditions
- Variables
- Operations
- Definitions

Clues to tell you what's
test worthy about this program

Step 2:

Derive a first set of test case specifications from pre-conditions, post-conditions and definitions

Step 3:

Complete the set of test case specifications using test catalogs

Combinational Testing: Summary

- Multiple parameters lead to gazillions of combinations
- Category-Partition Testing groups values, and selectively combines them;
- Pairwise-testing varies 2-way combinations in N-way test cases
- Catalog based testing reuses knowledge encoded in catalogs.