BLACK BOX TESTING (IV)

Syntax-driven testing

textbook, Section 9.1

Syntax-driven Testing

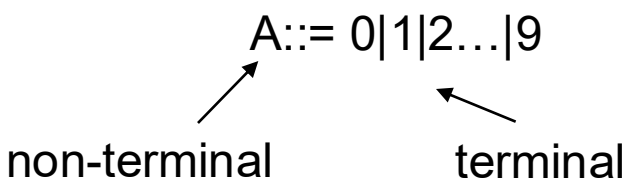
Applicable to systems whose specifications are described by some context-free grammars, say
- **compilers**
- **syntactic classifiers…**

Specifications expressed in a standard BNF (Backus-Naur Form) notation (production rules)

Terminal and non-terminal symbols (shown in < >)

Grammars

Syntax



derivation – replacement of non-terminals with one of its productions

Derivation continues until all non-terminals have been rewritten and only terminal symbols remain

Recognizers
parsing, lex yacc

generators
a string of terminals starting from the start symbol

Syntax-driven Testing: Example (BNF)

```
<expression>::= <expression>+<term>|
                <expression>-<term>|
                <term>
<term>::= <term> * <factor>|
          <term>/<factor>|
          <factor>
<factor>::=<identifier>| (<expression>)
<identifier>::=a|b|c|d|e...|z
```

BNF - Java

Types

<type> ::= *<primitive type>* | *<reference type>*

<primitive type> ::= *<numeric type>* | **`boolean`**

<numeric type> ::= *<integral type>* | *<floating-point type>*

<integral type> ::= **`byte`** | **`short`** | **`int`** | **`long`** | **`char`**

<floating-point type> ::= **`float`** | **`double`**

<reference type> ::= *<class or interface type>* | *<array type>*

<class or interface type> ::= *<class type>* | *<interface type>*

<class type> ::= *<type name>*

<interface type> ::= *<type name>*

<array type> ::= *<type>* **`[ ]`**

BNF - Python

```
<real number>    ::= <sign><natural number> |
                     <sign><natural number>'.'<digit sequence> |
                     <sign>'.'<digit><digit sequence> |
                     <sign><real number>'e'<natural number>
<sign>           ::= '' | '+' | '-'
<natural number> ::= '0' | <nonzero digit><digit sequence>
<nonzero digit>  ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<digit sequence> ::= '' | <digit><digit sequence>
<digit>          ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

Coverage criteria

Terminal symbol coverage (TSC)

test contains each terminal symbol *t* in the grammar *G*

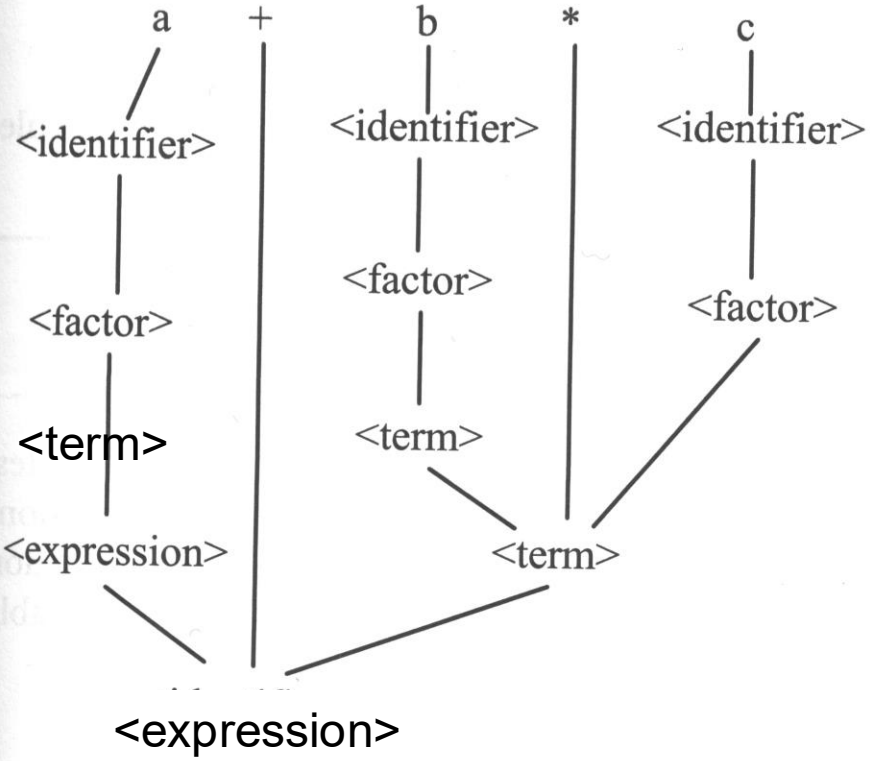
Production coverage (PDC)

test contains each production *p* in the grammar *G*

Derivation coverage (DC)

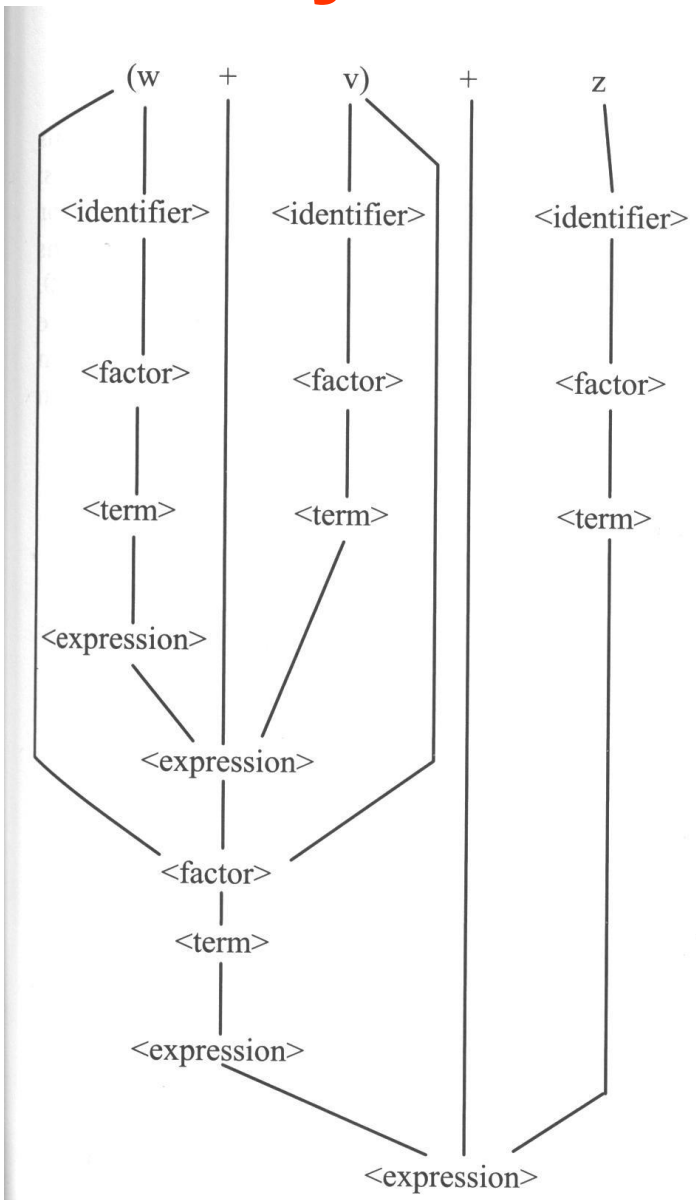
test contains every possible string derived from the grammar *G*

Syntax-driven Testing: production coverage



<expression>:: = <expression> + <term>|
<expression> — <term>|
<term>
<term>: = <term> * <factor>|
<term> / <factor>|
<factor>
<factor>:: = <identifier>| (<expression>)
<identifier>:: = a | b | c | d | e ... | z

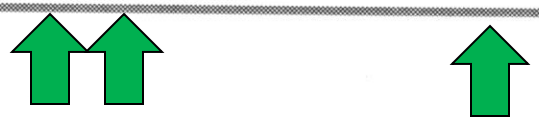
Syntax-driven Testing



<expression>:: = <expression> + <term>|
<expression> —<term>|
<term>
<term>:: = <term> * <factor>|
<term> / <factor>|
<factor>
<factor>:: = <identifier>| (<expression>)
<identifier>: = a | b | c | d | e ... | z

Syntax-driven Testing: Terminal symbol coverage

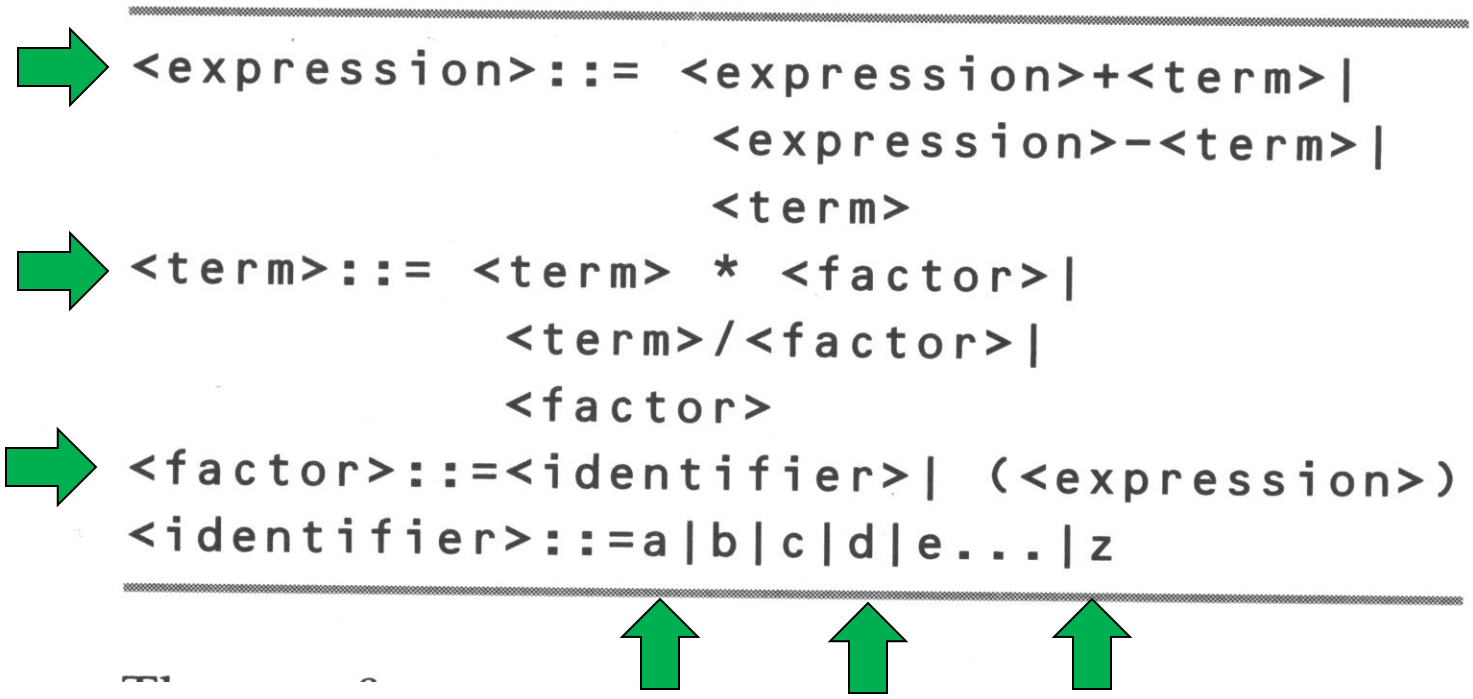
```
<expression>::= <expression>+<term>|
                <expression>-<term>|
                <term>
<term>::= <term> * <factor>|
          <term>/<factor>|
          <factor>
<factor>::=<identifier>| (<expression>)
<identifier>::=a|b|c|d|e...|z
```



Terminal symbol coverage (TSC)

test contains each terminal symbol *t* in the grammar *G.*
Number of terminal symbols (26)

Syntax-driven Testing: production coverage



Production coverage (PDC)

test contains each production *p* in the grammar *G (*26+ 2 +3 +3)

Syntax-driven Testing: Derivation coverage

```
<expression>::= <expression>+<term>|
                <expression>-<term>|
                <term>
<term>::= <term> * <factor>|
          <term>/<factor>|
          <factor>
<factor>::=<identifier>| (<expression>)
<identifier>::=a|b|c|d|e...|z
```

Derivation coverage (DC)

test contains every possible string derived from the grammar *G*. Infinite e.g., <term>::=<term>*<factor>

Strings' errors

- Even small specification may result in many strings
- Strings' errors can be classified as
  - Syntax error
  - Delimiter errors
  - Field-value errors

Delimiter errors

- Delimiters are chars placed between two field
- Excellent source of test cases
- Test for
  - Missing delimiter
  - Wrong delimiter
  - Not a delimiter
  - Too many delimiters
  - Paired delimiters

Field-value errors

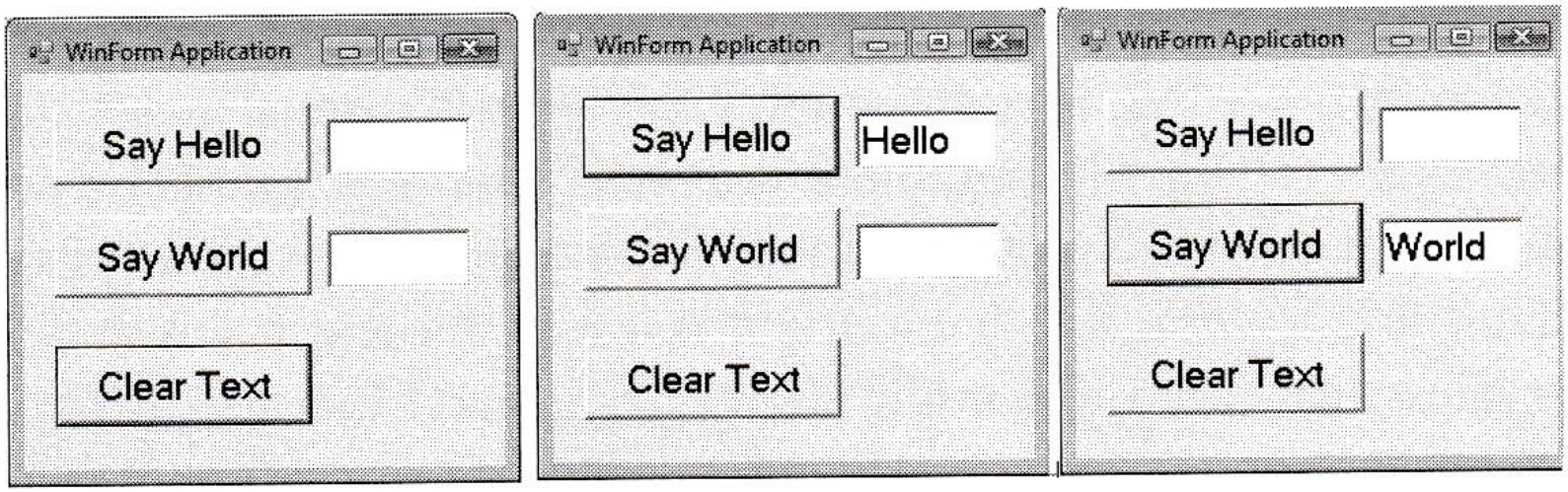
- **Fields have meaning and must be tested:**
  - **Boundary values**
  - **Excluded values**
  - **Troublesome values**

Applications of String Design Testing (SDT)

- All user interfaces
- Operating command processing
- All communications interfaces and protocols
- Internal interface and protocol (call sequences)

Coverage and usage testing with the use of finite state machines

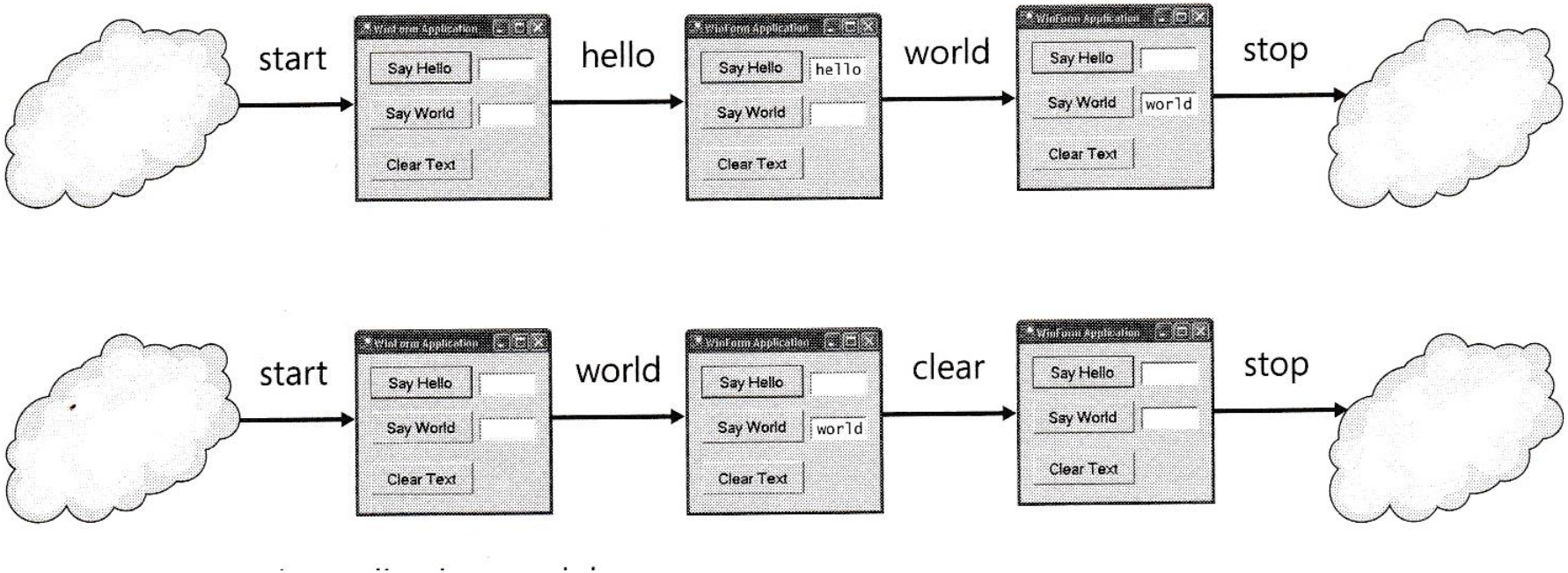
State-based testing



Application with 3 buttons:
-Prints “Hello”
-Prints “World”
-Clears the fields

When the application starts both text boxes are clear regardless of their state when the application terminated

Example test cases

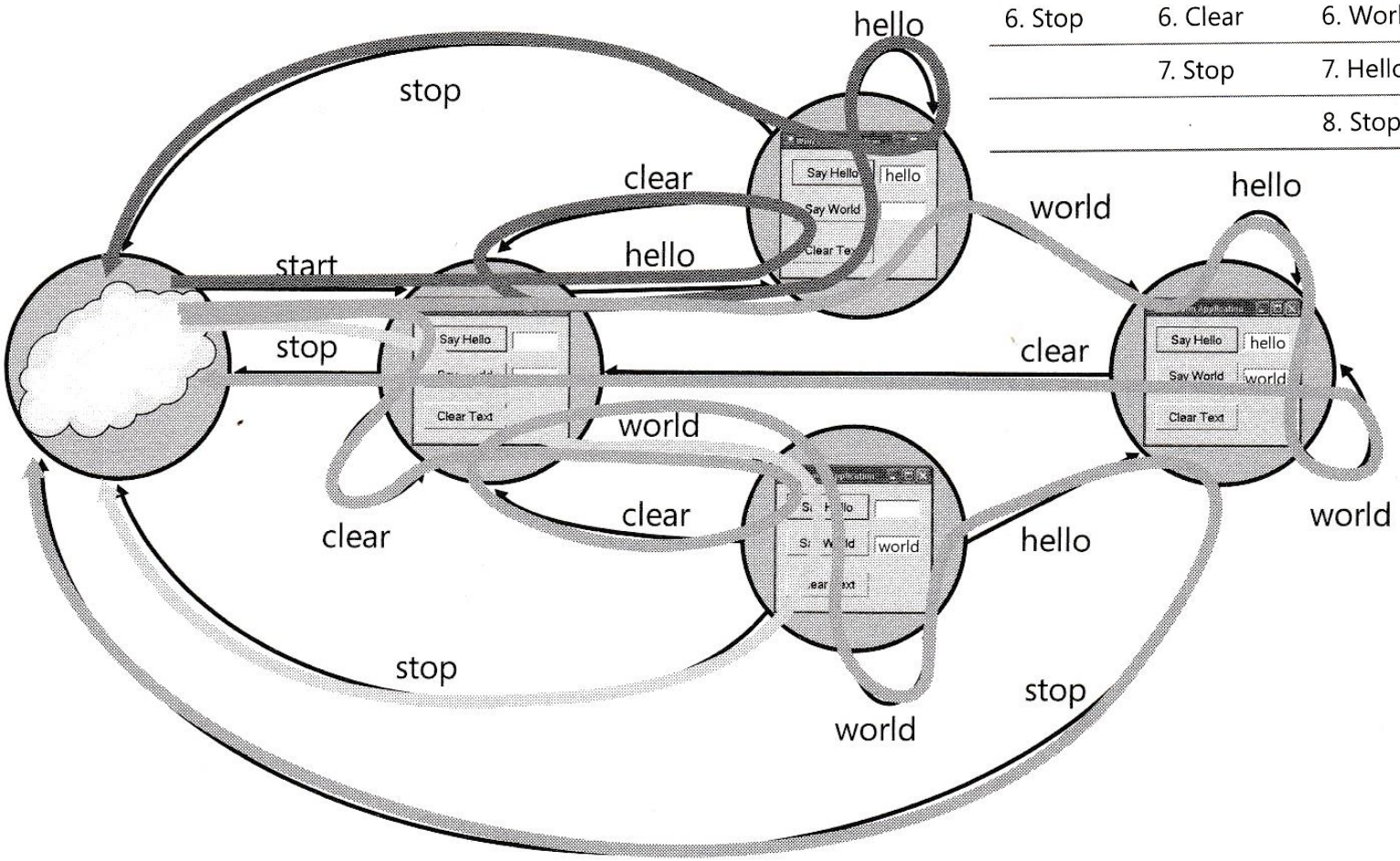


Example - test cases

Test 1	Test 2	Test 3	Test 4
1. Start	1. Start	1. Start	1. Start
2. Hello	2. Hello	2. Clear	2. World
3. Clear	3. World	3. World	3. Stop
4. Hello	4. Hello	4. Clear	
5. Hello	5. World	5. World	
6. Stop	6. Clear	6. World	
	7. Stop	7. Hello	
		8. Stop	

Example - test cases

Test 1	Test 2	Test 3	Test 4
1. Start	1. Start	1. Start	1. Start
2. Hello	2. Hello	2. Clear	2. World
3. Clear	3. World	3. World	3. Stop
4. Hello	4. Hello	4. Clear	
5. Hello	5. World	5. World	
6. Stop	6. Clear	6. World	
	7. Stop	7. Hello	
		8. Stop	



Finite State Machines

Fundamental model comprising the following concepts

- **States**
- **State transitions (transitions)**
- **Inputs**
- **Outputs**

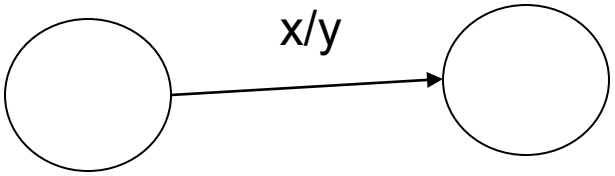
Graphical representation

Each state – node of a graph
Each transition – directed link between the states
Inputs and outputs are associated with state transitions [Mealy model]

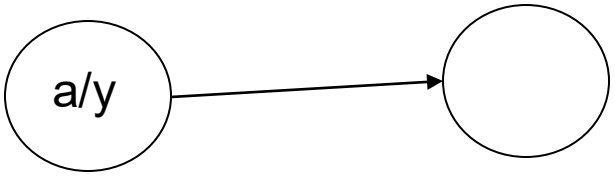
Finite State Machines
Moore and Mealy models

Inputs and outputs are associated with state transitions [Mealy model]

Outputs are associated with states [Moore model]

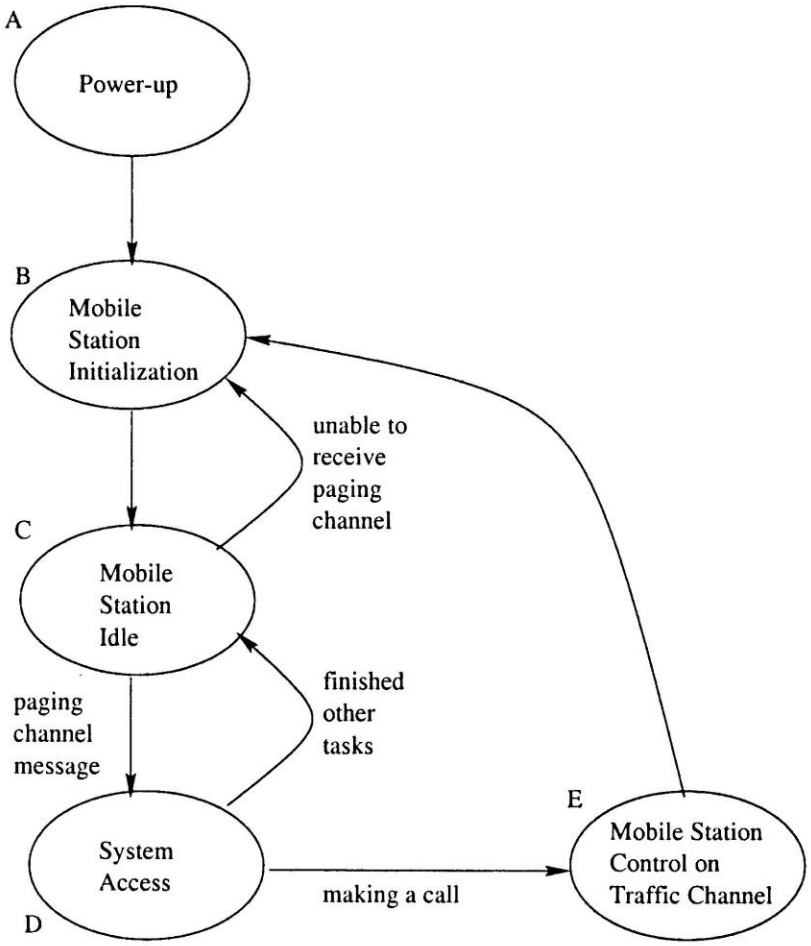


MEALY– link-weighted



MOORE– node-weighted

Example of FSM



FSM:
Categories of applications

Menu-driven software

Real-time systems, control systems

Web-based applications

Advantages and limitations

- easy to interpret and intuitive
- scalability (large number of states)

Finite State Machine

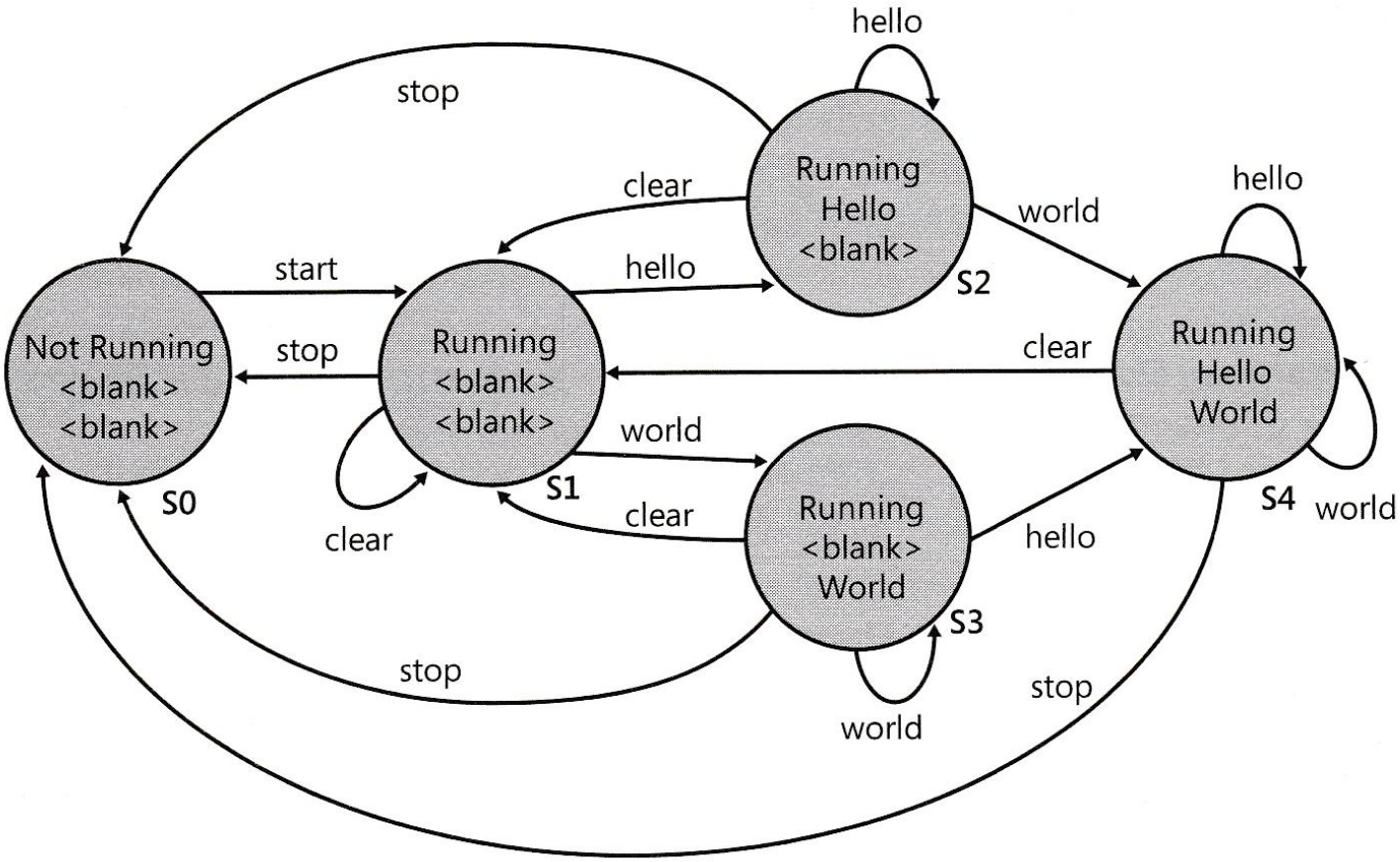
Both text boxes are clear (S1).

The first text box contains "Hello" and the second text box is clear (S2).

The first text box is clear and the second text box contains "World" (S3).

The first text box contains "Hello", and the second contains "World" (S4).

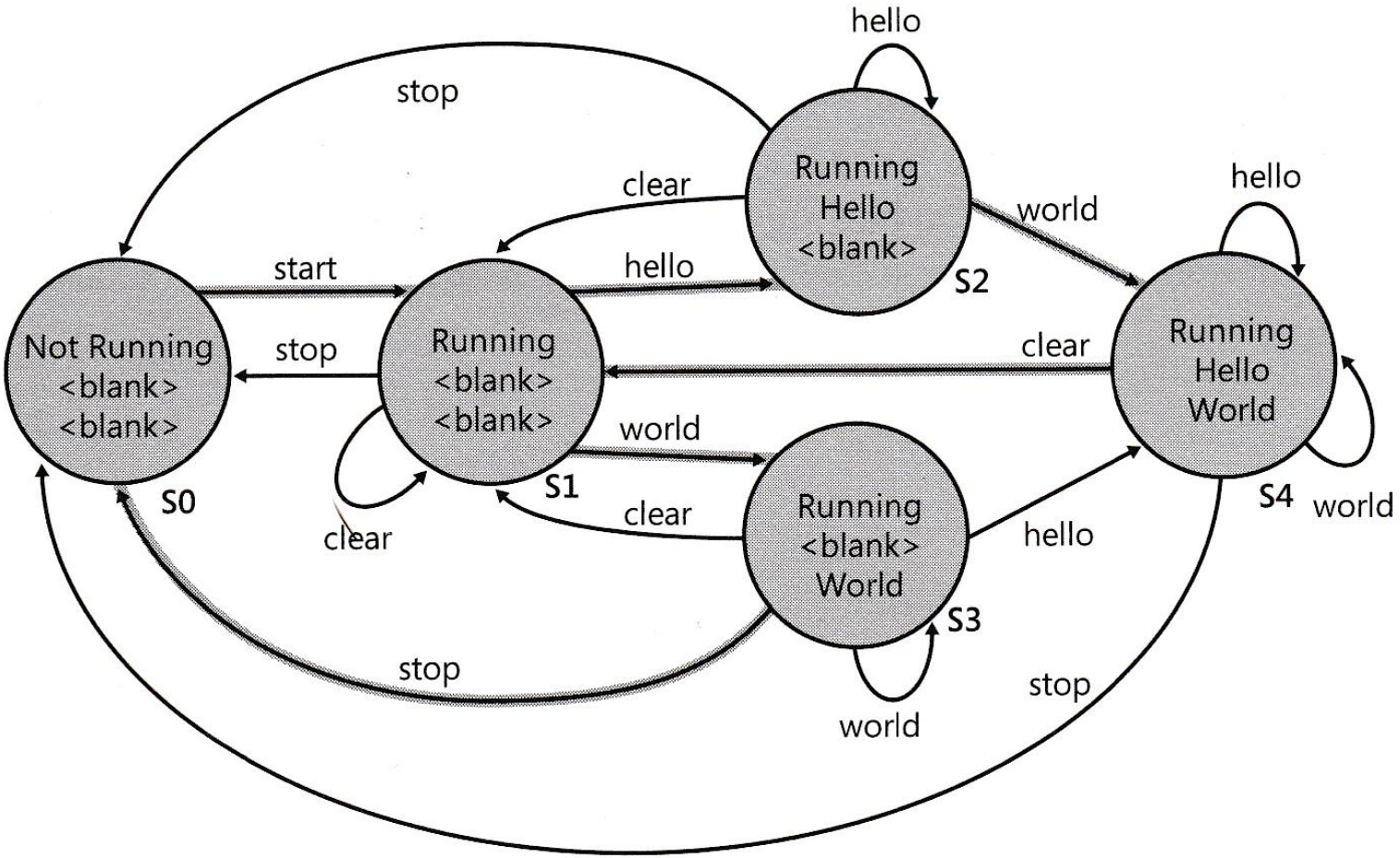
Finite State Machine



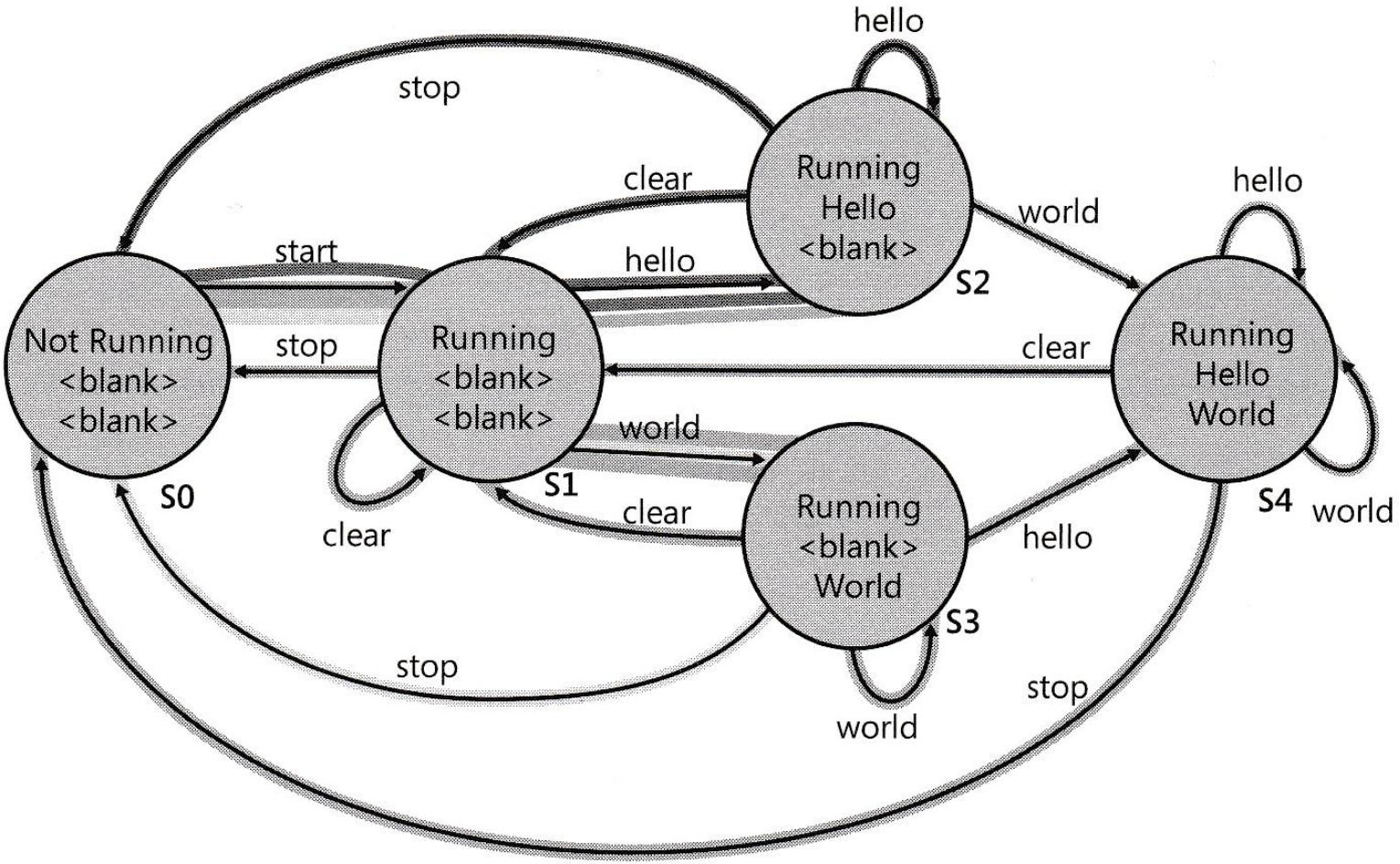
Testing – random walk
(smart monkey test)

Test step	Action	Text box 1	Text box 2
0	Start Application	Empty	Empty
1	Say World	Empty	World
2	Clear	Empty	Empty
3	Say Hello	Hello	Empty
4	Say Hello	Hello	Empty
5	Clear	Empty	Empty
6	Say World	Empty	World
7	Say Hello	Hello	World
8	Say World	Hello	World
9	Clear	Empty	Empty
10	Stop Application	Empty	Empty

Testing – traversing all states



Testing – traversing all transitions



Problems with FSM (1)

State problems

Missing state: valid current state and input; next state is missing

Incorrect state: ill-defined behavior

Extra state: related to unreachable state

Transition problems

Missing transition: valid current state and input; next state missing

Incorrect transition: transition to unexpected state/output

Extra transition: multiple transitions for the current state, unexpected output

Problems with FSM (2)

Input problems

Even invalid inputs are expected to be handled correctly (outputting error message, exception handling)

IN FSM we are concerned with STATE and TRANSITION problems

Testing for states and transitions: selection of test cases

Make sure that each state can be reached by some test cases

Make sure that each transition is covered by some test cases

Test case: a series of input values that enables some transitions

Web-based applications and testing

Document and **information** focus rather than computing

Navigation and computing

No single entry point (no top menu); any web page can be a starting point

Huge number of pages

Multilayer web architecture

client-Web browser
web server
middleware
database back-end

Web testing and FSM

Each web page == state.

Any page can be the initial state and each page could be the final state

Transitions == web navigation; follow hypertext links embedded in HTLM documents and web content

Input and output ==clicking on embedded link; loading the requested page

Modeling Web navigation

Majority of web navigations follow embedded hypertext links (instead of bookmarks and typed URLs)

Markov chains

Huge number of states – selective testing (complete coverage not feasible)

Highly used parts of the FSM

Probabilities of transitions and Markov property

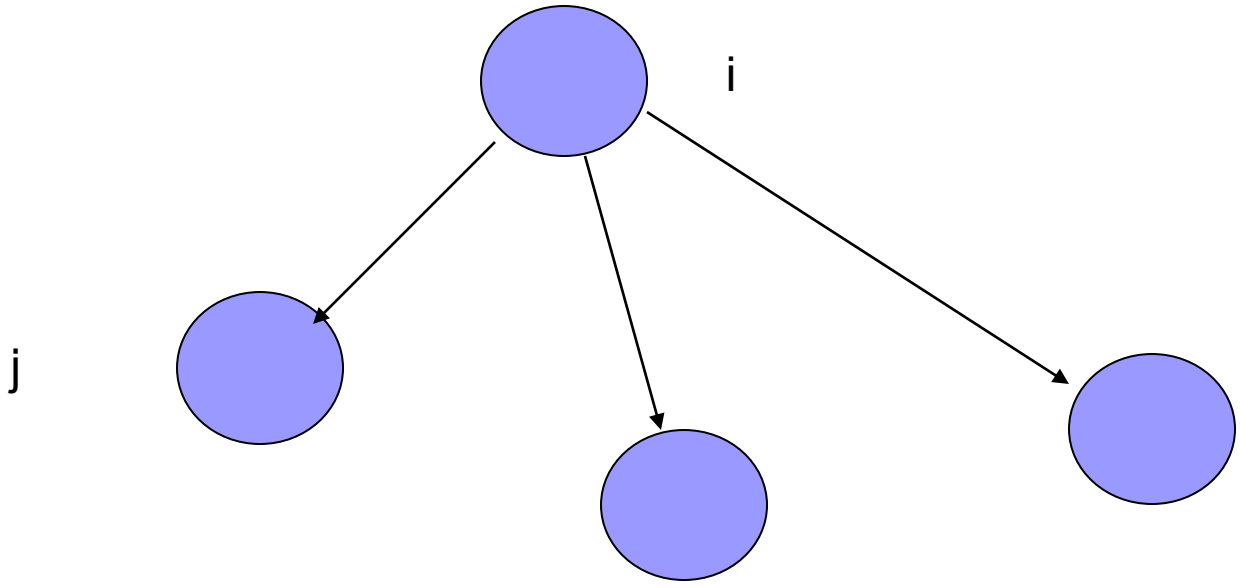
Current state $x_n = i$
Next state $x_{n+1} = j$
Previous state $x_{n-1} = l$, $x_{n-2} = k$, etc

$$p_{ij} = \text{Prob}(x_{n+1} = j \mid x_n = i, x_{n-1} = l, \ldots) = \text{Prob}(x_{n+1} = j \mid x_n = i)$$

Markov (memoryless) property

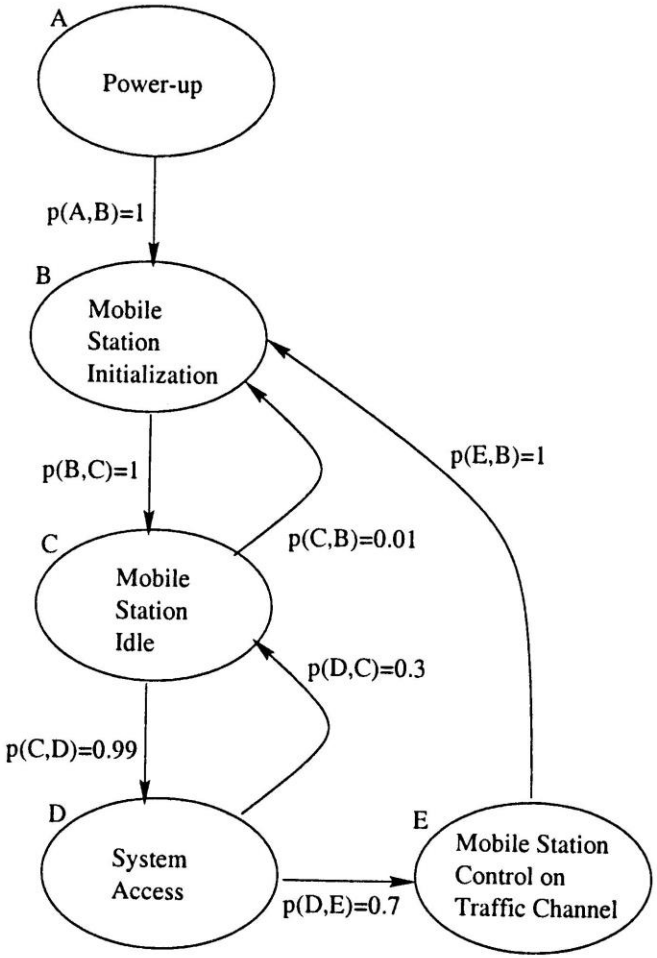
Markov property

$p_{ij} = \text{Prob}(x_{n+1} = j \mid x_n = i, x_{n-1} = l, \ldots) = \text{Prob}(x_{n+1} = j \mid x_n = i)$



$$\sum_j p_{ij} = 1$$

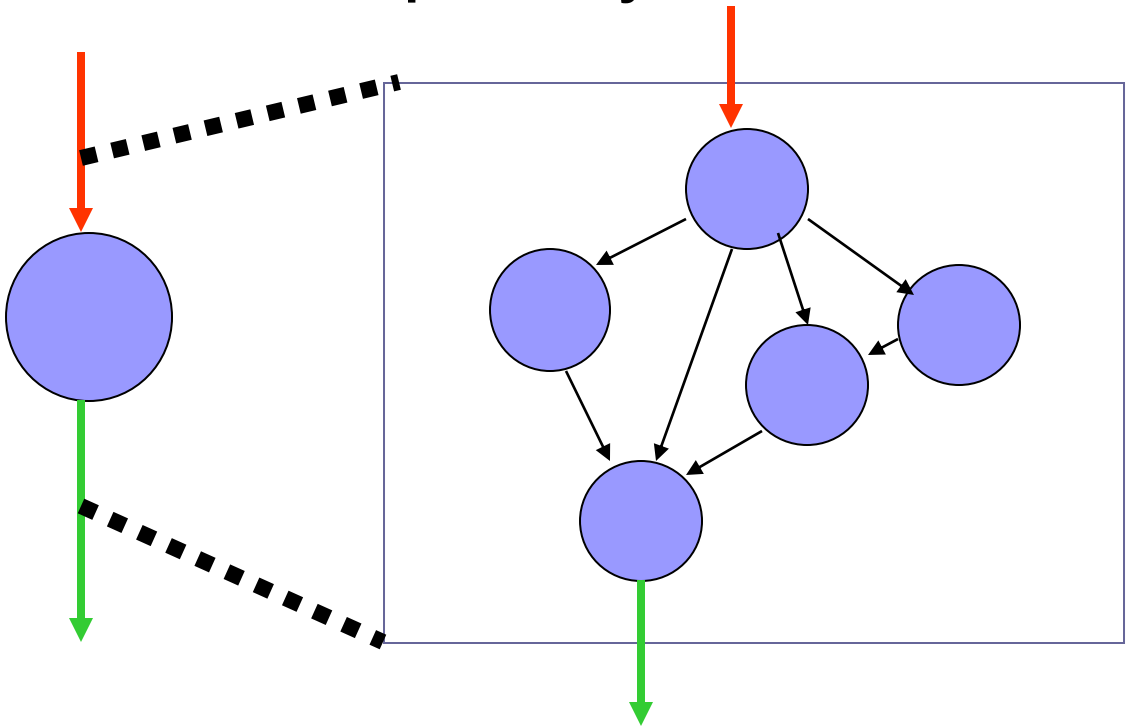
Markov operational profile



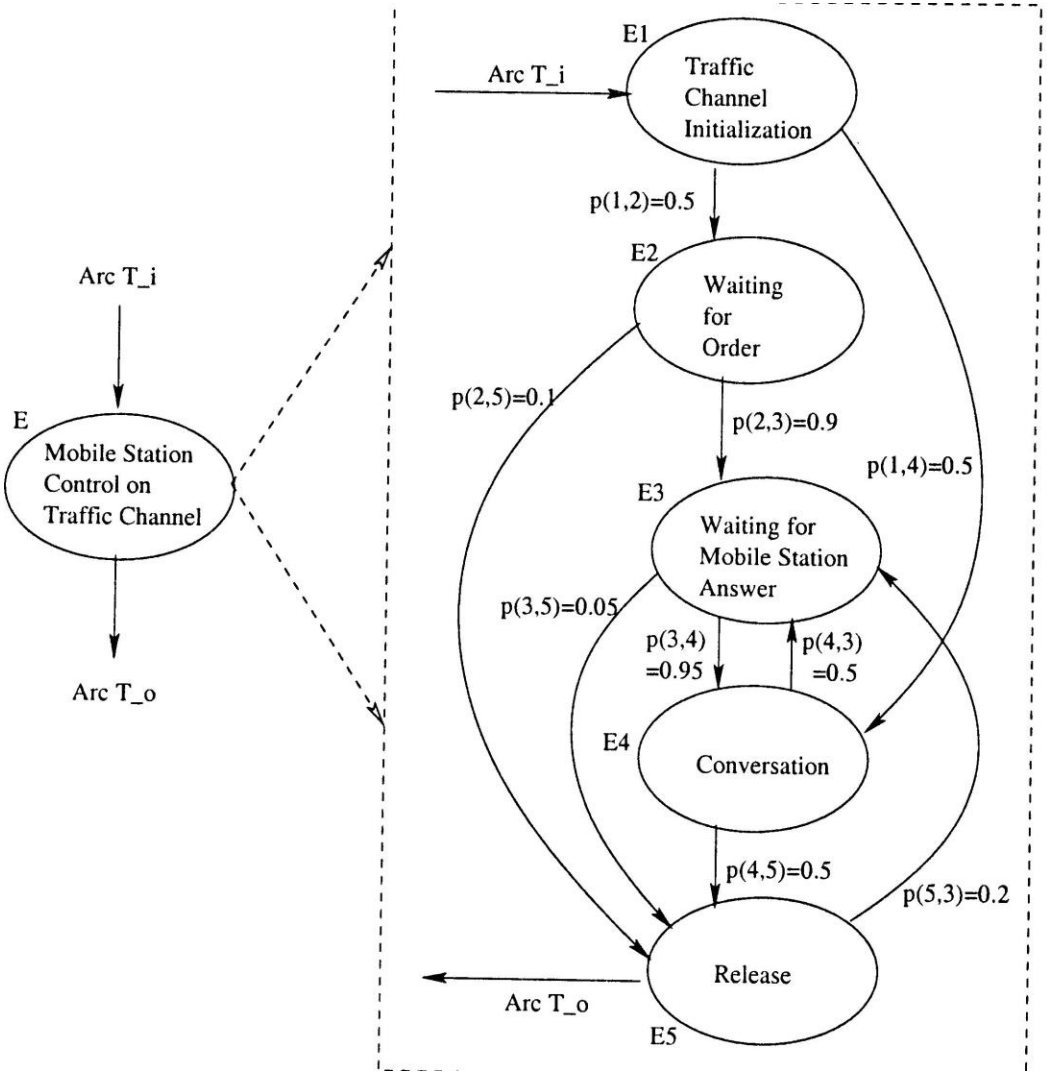
Unified Markov Model (UMM)

Large systems - huge number of states

Building hierarchies of Markov chains – depending upon the levels of detail captured by the model



Unified Markov Model (UMM)



Unified Markov Model (UMM) and usage-based testing

Generation of test cases driven by probabilities of usage

Test all above a certain threshold

Three types of threshold

Overall probability threshold

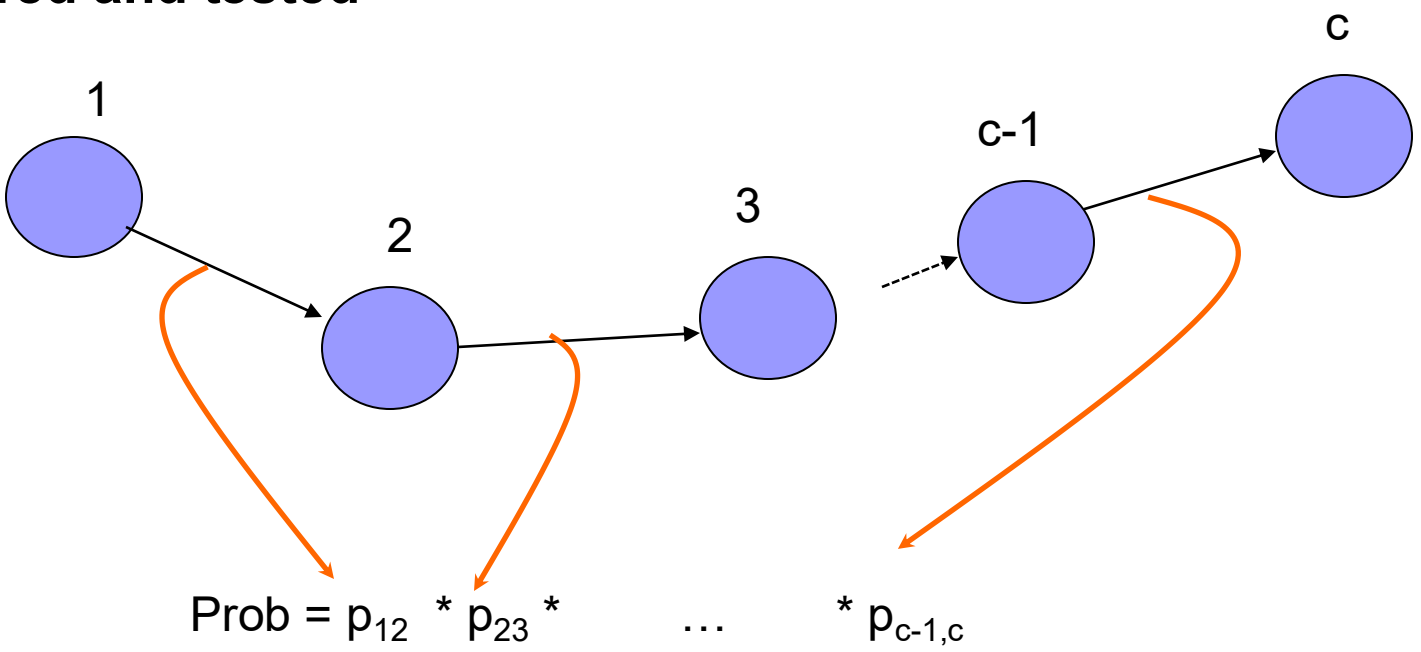
Stationary probability threshold

Transition probability threshold

Unified Markov Model (UMM) and usage-based testing

Overall probability threshold

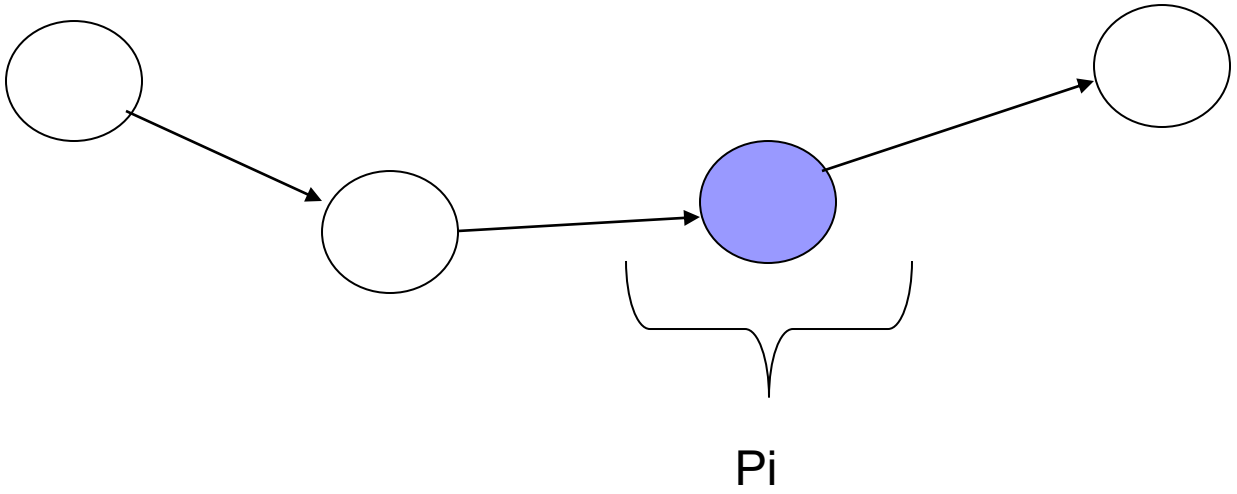
End-to –end operation: commonly used sequences are covered and tested



Unified Markov Model (UMM) and usage-based testing

Stationary probability threshold

Frequently visited states
Stationarity – computing aspects

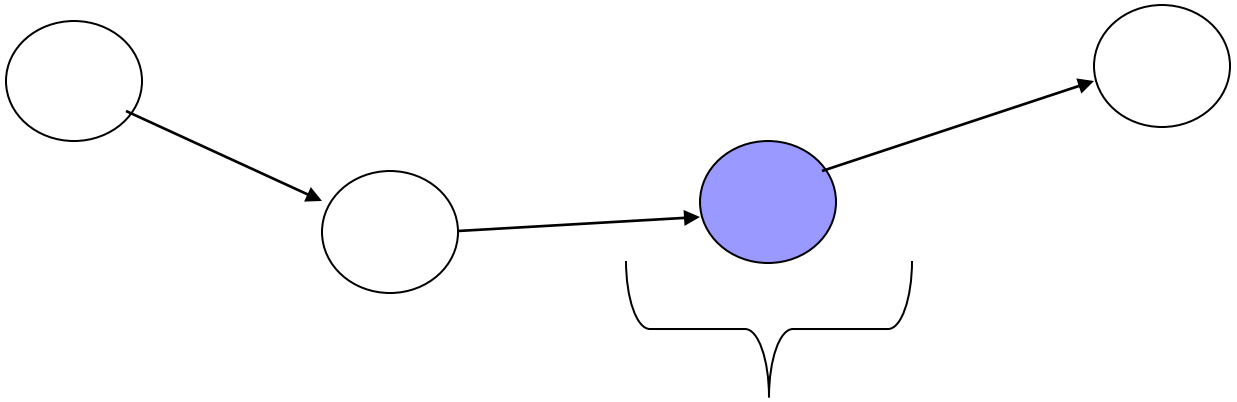


Unified Markov Model (UMM) and usage-based testing

Stationary probabilities

System of linear equations w.r.t. probabilities

$$p_j = \sum_{i=1}^{n} p_{ij} p_i, \quad j = 1, 2, ..., n$$

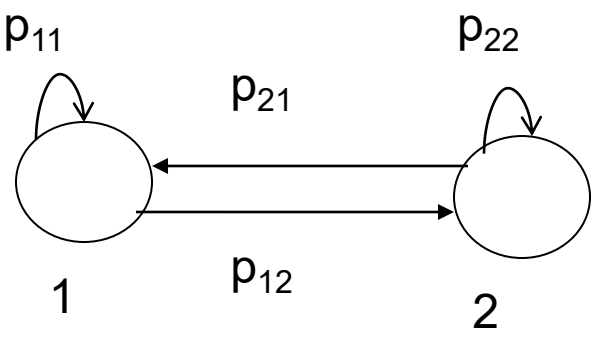


Pi

Unified Markov Model (UMM) and usage-based testing-example

$$p_j = \sum_{i=1}^{n} p_{ij} p_i, \quad j = 1, 2$$

System of linear equations w.r.t. probabilities



$$p_1 = p_{11} p_1 + p_{21} p_2$$

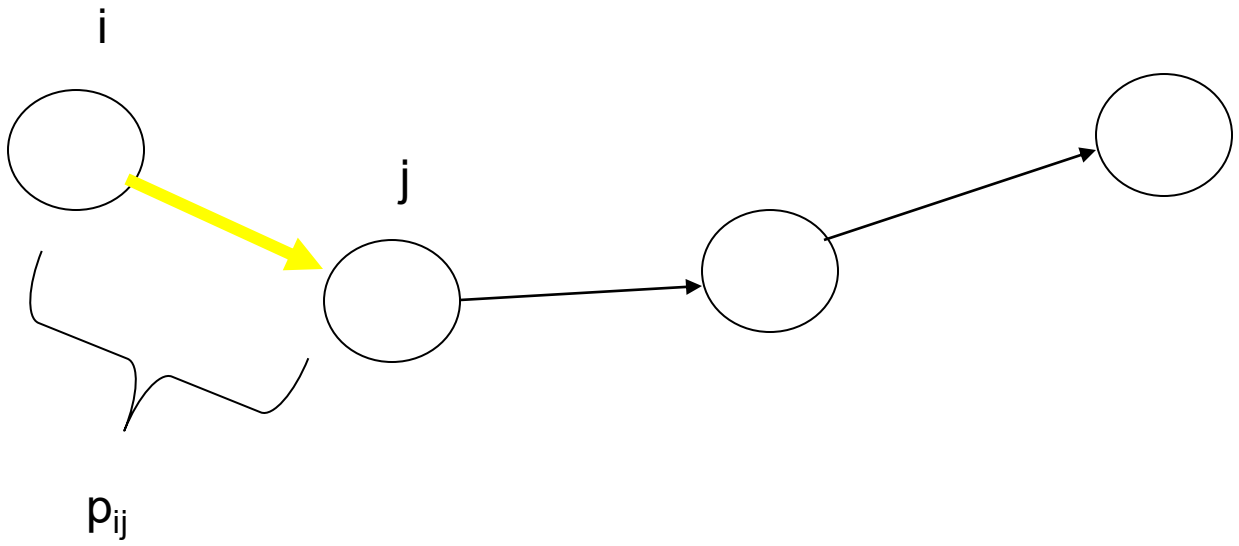
$$p_2 = p_{12} p_1 + p_{22} p_2$$

$$p_1 + p_2 = 1$$

Unified Markov Model (UMM) and usage-based testing

transition probability threshold

Ensure commonly used operation pairs



Combinatorial testing

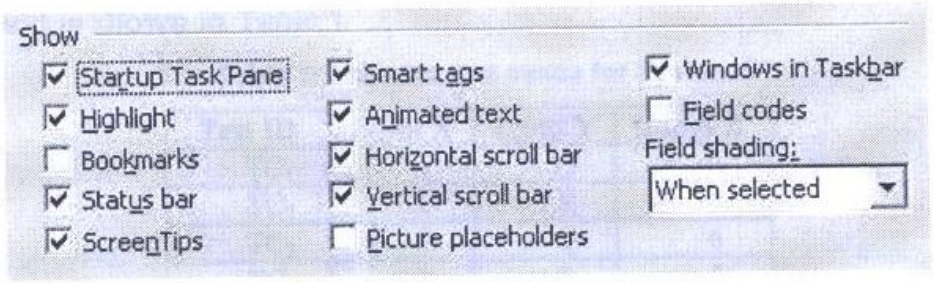
Combinatorial testing problems

We are faced with combinatorial testing whenever we have a product that processes multiple variables that may interact.

User interfaces
Operating systems
Peripherals
Databases
….

Configuration testing

<u>Example</u> How many tests?? $2^{12}*3$ (for the Field shading menu)= 12, 288



Portion of Options Dialog in MS Word

Combinatorial testing problems

A Web site must operate correctly with different browsers—Internet Explorer 5.0, 5.5, and 6.0, Netscape 6.0, 6.1, and 7.0, Mozilla 1.1, and Opera 7; using different plug-ins—RealPlayer, MediaPlayer, or none; running on different client operating systems—Windows 95, 98, ME, NT, 2000, and XP; receiving pages from different servers—IIS, Apache, and WebLogic; running on different server operating systems—Windows NT, 2000, and Linux.

Web Combinations

8 browsers
3 plug-ins
6 client operating systems
3 servers
3 server OS

1,296 combinations.

In an object-oriented system, an object of class A can pass a message containing a parameter P to an object of class X. Classes B, C, and D inherit from A so they too can send the message. Classes Q, R, S, and T inherit from P so they too can be passed as the parameter. Classes Y and Z inherit from X so they too can receive the message.

OO Combinations
4 senders
5 parameters
3 receivers

60 combinations.

Combinatorial testing

bugs lurk in corners and congregate at boundaries

B. Balzer

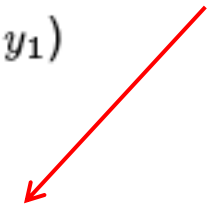
Interaction faults:

Interaction fault occurs when a certain combination of input variables (say, 2 variables) is involved

Software failures rarely occur due to faults that involve more than 5 variables

Example

```
1    begin
2      int x, y, z;
3      input (x, y, z);
4      if(x==x1 and y==y2)
5        output (f(x, y, z));
6      else if(x==x2 and y==y1)
7        output (g(x, y));
8      else
9        output (f(x, y, z)+g(x, y))
10   end
```



Correct output: f(x, y, z)-g(x, y) when x=x1 and y=y1.

interaction between variables x and y.

Combinatorial testing problems: possible solutions

Pairwise testing:

Instead of considering all combinations of the input variables construct a set of test cases that "covers" *all* combinations of the test data for each pair of variables

Use of input-output analysis:

Determine (discover) relationship between inputs and output; not all inputs affect each output

Pairwise testing

General case of *all-combination* testing:

{x_1 x_2… x_n} "n" input variables , "k" different values assumed by the variables

k^n different tests

Pairwise testing (two-way, all-pair):

Instead of considering all combinations of the input variables construct a set of test cases that "covers" *all* combinations of the test data for each pair of variables

two-way, three –way… five-way

See additional reading material on eclass

Pairwise testing- real-world evidence (1)

% faults detected

1-way (single parameter) ~30-50%

2-way (pairwise) ~70-90%

3-way ~95-98%

4-way and above - <2-5%

Example

C(10,2)= 45 C(10,3)=120 C(10,4)=210 C(10,5)=252

Pairwise testing- real-world evidence (2)

Microsoft (windows): configuration and compatibility testing
pairwise testing > 90% faults

National Institute of Standards and Technology (NIST):
3- way testing > 90% reduction in test effort

Automotive & embedded systems
pairwise testing

Pairwise testing- real-world evidence (3)

Metric	Two-Way (Pairwise)	Random
Fault coverage (typical)	70–90% of real bugs	Variable; depends on number of tests
Test suite size reduction	90–99% fewer tests than exhaustive	Uncontrolled
Predictability	High	Low
Practical efficiency	Excellent	Moderate
Common usage	Widely used in QA, embedded, configuration, and OS testing	Used in fuzzing, performance, stress testing

Pairwise testing

Font style – style of font can be set as any combination of bold, italic, strikethrough and underline

Each parameter has two variable states

Pairwise testing

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Bold	c	c	c	c	u	u	u	c	u	u	c	c	u	c	u	u
Italic	u	c	c	c	c	c	c	c	u	u	u	u	u	u	c	u
Underline	u	u	c	c	u	c	c	u	c	c	c	c	u	u	u	u
Strikethrough	u	u	u	c	u	u	c	c	u	c	u	c	c	c	c	u

Pairwise testing

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Bold	c	c	c	c	u	u	u	c	u	u	c	c	u	c	u	u
Italiç	u	c	c	c	c	c	c	c	u	u	u	u	u	u	c	u
Underline	u	u	c	c	u	c	c	u	c	c	c	c	u	u	u	u
Strikethrough	u	u	u	c	u	u	c	c	u	c	u	c	c	c	c	u

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Bold	c	c	c	c	u	u	u	c	u	u	c	c	u	c	u	u
Italic	u	c	c	c	c	c	c	c	u	u	u	u	u	u	c	u
Underline	u	u	c	c	u	c	c	u	c	c	c	c	u	u	u	u
Strikethrough	u	u	u	c	u	u	c	c	u	c	u	c	c	c	c	u

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Bold	c	c	c	c	u	u	u	c	u	u	c	c	u	c	u	u
Italic	u	c	c	c	c	c	c	c	u	u	u	u	u	u	c	u
Underline	u	u	c	c	u	c	c	u	c	c	c	c	u	u	u	u
Strikethrough	u	u	u	c	u	u	c	c	u	c	u	c	c	c	c	u

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Bold	c	c	c	c	u	u	u	c	u	u	c	c	u	c	u	u
Italic	u	c	c	c	c	c	c	c	u	u	u	u	u	u	c	u
Underline	u	u	c	c	u	c	c	u	c	c	c	c	u	u	u	u
Strikethrough	u	u	u	c	u	u	c	c	u	c	u	c	c	c	c	u

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Bold	c	c	c	c	u	u	u	c	u	u	c	c	u	c	u	u
Italic	u	c	c	c	c	c	c	c	u	u	u	u	u	u	c	u
Underline	u	u	c	c	u	c	c	u	c	c	c	c	u	u	u	u
Strikethrough	u	u	u	c	u	u	c	c	u	c	u	c	c	c	c	u

5 tests:

2, 7, 11, 14, 16

Example

Three variables assuming 3 values

A, B, C

X, Y, Z

1, 2,3

All combinations 3^3 =27

pairwise testing- 9 test cases

A	X	1
A	Y	2
A	Z	3
B	X	2
B	Y	3
B	Z	1
C	X	3
C	Y	1
C	Z	2

Orthogonal array (Taguchi)

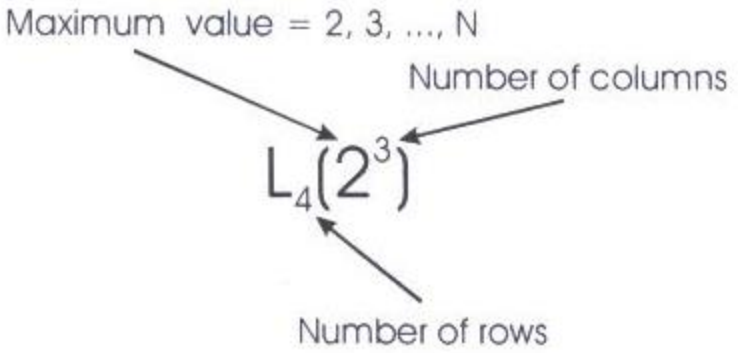
Consider two numbers, 1 and 2
How many pair combinations of “1” and “2” exist?
{1, 1} {1, 2} {2, 1} {2, 2}

Orthogonal array: for any pair of columns– all pairs of “1” and “2” occur there

	1	2	3
1	1	1	1
2	1	2	2
3	2	1	2
4	2	2	1

Orthogonal array - notation

	1	2	3
1	1	1	1
2	1	2	2
3	2	1	2
4	2	2	1



Orthogonal array:

- for any pair of columns– all pairs of "1" and "2" occur there
- if any pair occurs multiple times, all pairs occur the same number of times

Orthogonal array - example

	1	2	3	4
1	1	1	1	1
2	1	2	2	2
3	1	3	3	3
4	2	1	2	3
5	2	2	3	1
6	2	3	1	2
7	3	1	3	2
8	3	2	1	3
9	3	3	2	1

$L_9(3^4)$

Orthogonal array - example

	1	2	3	4	5
1	1	1	1	1	1
2	1	2	3	3	1
3	1	3	2	3	2
4	1	2	2	1	3
5	1	3	1	2	3
6	1	1	3	2	2
7	2	2	2	2	2
8	2	3	1	1	2
9	2	1	3	1	3
10	2	3	3	2	1
11	2	1	2	3	1
12	2	2	1	3	3
13	3	3	3	3	3
14	3	1	2	2	3
15	3	2	1	2	1
16	3	1	1	3	2
17	3	2	3	1	2
18	3	3	2	1	1

$L_{18}(3^5)$

Orthogonal array - example

	1	2	3	4	5	6	7	8
1	1	1	1	1	1	1	1	1
2	1	1	2	2	2	2	2	2
3	1	1	3	3	3	3	3	3
4	1	2	1	1	2	2	3	3
5	1	2	2	2	3	3	1	1
6	1	2	3	3	1	1	2	2
7	1	3	1	2	1	3	2	3
8	1	3	2	3	2	1	3	1
9	1	3	3	1	3	2	1	2
10	2	1	1	3	3	2	2	1
11	2	1	2	1	1	3	3	2
12	2	1	3	2	2	1	1	3
13	2	2	1	2	3	1	3	2
14	2	2	2	3	1	2	1	3
15	2	2	3	1	2	3	2	1
16	2	3	1	3	2	3	1	2
17	2	3	2	1	3	1	2	3
18	2	3	3	2	1	2	3	1

$L_{18}(2^1 3^7)$ Orthogonal Array

Orthogonal array –selection of pairwise subsets for testing

- Identify the variables and the number of values they assume
- Find an orthogonal array :
  - the number of columns = the number of variables
  - values in columns = number of values assumed by the variables
- Map the problem onto the orthogonal array
- Construct test cases

Variable identification

A Web site must operate correctly with different browsers—Internet Explorer 5.0, 5.5, and 6.0, Netscape 6.0, 6.1, and 7.0, Mozilla 1.1, and Opera 7; using different plug-ins—RealPlayer, MediaPlayer, or none; running on different client operating systems—Windows 95, 98, ME, NT, 2000, and XP; receiving pages from different servers—IIS, Apache, and WebLogic; running on different server operating systems—Windows NT, 2000, and Linux.

Web Combinations

8 browsers
3 plug-ins
6 client operating systems
3 servers
3 server OS

1,296 combinations.

Browser: Internet Explorer 5.0, 5.5, 6.0 (3)
Netscape 6.0 6.1
Mozilla 1.1
Opera 7 (8)

Plug-in None, RealPlayer, MediaPlayer

Client operating system: Windows 95, 95, ME,NT, 2000 XP (6)

Server: IIS, Apache, WebLogic (3)

Server operating system: Windows NT, 2000, Linux (3)

Selection of orthogonal array

Ideally the array should be $8^1 6^1 3^3$

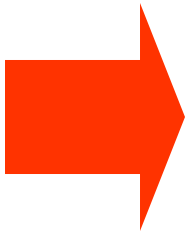
If not available, look for a larger one; the result is $L_{64}(8^2 4^3)$

	1	2	3	4	5
1	1	1	1	1	1
2	1	4	3	4	4
3	1	4	2	4	4
4	1	1	4	1	1
5	1	3	5	3	3
6	1	2	7	2	2
7	1	2	6	2	2
8	1	3	8	3	3
9	3	4	1	3	3
10	3	1	3	2	2
11	3	1	2	2	2
12	3	4	4	3	3
13	3	2	5	1	1
14	3	3	7	4	4
15	3	3	6	4	4
16	3	2	8	1	1
17	2	3	1	2	1
18	2	2	3	3	4
19	2	2	2	3	4
20	2	3	4	2	1
21	2	1	5	4	3
22	2	4	7	1	2
23	2	4	6	1	2
24	2	1	8	4	3
25	4	2	1	4	3
26	4	3	3	1	2
27	4	3	2	1	2
28	4	2	4	4	3
29	4	4	5	2	1
30	4	1	7	3	4
31	4	1	6	3	4
32	4	4	8	2	1
33	5	2	1	4	2
34	5	3	3	1	3
35	5	3	2	1	3
36	5	2	4	4	2
37	5	4	5	2	4
38	5	1	7	3	1
39	5	1	6	3	1
40	5	4	8	2	4
41	7	3	1	2	4
42	7	2	3	3	1
43	7	2	2	3	1
44	7	3	4	2	4
45	7	1	5	4	2
46	7	4	7	1	3
47	7	4	6	1	3
48	7	1	8	4	2
49	6	4	1	3	2
50	6	1	3	2	3
51	6	1	2	2	3
52	6	4	4	3	2
53	6	2	5	1	4
54	6	3	7	4	1
55	6	3	6	4	1
56	6	2	8	1	4
57	8	1	1	1	4
58	8	4	3	4	1
59	8	4	2	4	1
60	8	1	4	1	4
61	8	3	5	3	2
62	8	2	7	2	3
63	8	2	6	2	3
64	8	3	8	3	2

Map the variables (1)

1 ↔ IE 5.0
2 ↔ IE 5.5
3 ↔ IE 6.0
4 ↔ Netscape 6.0
5 ↔ Netscape 6.1
6 ↔ Netscape 7.0
7 ↔ Mozilla 1.1
8 ↔ Opera 7

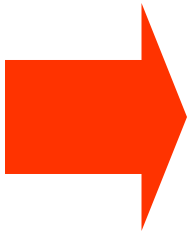
	1	2	3	4	5
1	1	1	1	1	1
2	1	4	3	4	4
3	1	4	2	4	4
4	1	1	4	1	1
5	1	3	5	3	3
6	1	2	7	2	2
7	1	2	6	2	2
8	1	3	8	3	3
9	3	4	1	3	3
10	3	1	3	2	2
11	3	1	2	2	2
12	3	4	4	3	3
13	3	2	5	1	1
14	3	3	7	4	4
15	3	3	6	4	4
16	3	2	8	1	1
17	2	3	1	2	1
18	2	2	3	3	4
19	2	2	2	3	4
20	2	3	4	2	1
21	2	1	5	4	3
22	2	4	7	1	2
23	2	4	6	1	2
24	2	1	8	4	3
25	4	2	1	4	3



	Browser	2	3	4	5
1	IE 5.0	1	1	1	1
2	IE 5.0	4	3	4	4
3	IE 5.0	4	2	4	4
4	IE 5.0	1	4	1	1
5	IE 5.0	3	5	3	3
6	IE 5.0	2	7	2	2
7	IE 5.0	2	6	2	2
8	IE 5.0	3	8	3	3
9	IE 6.0	4	1	3	3
10	IE 6.0	1	3	2	2
11	IE 6.0	1	2	2	2
12	IE 6.0	4	4	3	3
13	IE 6.0	2	5	1	1
14	IE 6.0	3	7	4	4
15	IE 6.0	3	6	4	4
16	IE 6.0	2	8	1	1
17	IE 5.5	3	1	2	1
18	IE 5.5	2	3	3	4
19	IE 5.5	2	2	3	4
20	IE 5.5	3	4	2	1
21	IE 5.5	1	5	4	3
22	IE 5.5	4	7	1	2
23	IE 5.5	4	6	1	2
24	IE 5.5	1	8	4	3
25	Net 6.0	2	1	4	3

Map the variables (2)

1- None
2-RealPlayer
3- MediaPlayer
4- NOT USED



	Browser	Plug-in	3	4	5
1	IE 5.0	None	1	1	1
2	IE 5.0	4	3	4	4
3	IE 5.0	4	2	4	4
4	IE 5.0	None	4	1	1
5	IE 5.0	MediaPlayer	5	3	3
6	IE 5.0	RealPlayer	7	2	2
7	IE 5.0	RealPlayer	6	2	2
8	IE 5.0	MediaPlayer	8	3	3
9	IE 6.0	4	1	3	3
10	IE 6.0	None	3	2	2
11	IE 6.0	None	2	2	2
12	IE 6.0	4	4	3	3
13	IE 6.0	RealPlayer	5	1	1
14	IE 6.0	MediaPlayer	7	4	4
15	IE 6.0	MediaPlayer	6	4	4
16	IE 6.0	RealPlayer	8	1	1
17	IE 5.5	MediaPlayer	1	2	1
18	IE 5.5	RealPlayer	3	3	4
19	IE 5.5	RealPlayer	2	3	4
20	IE 5.5	MediaPlayer	4	2	1
21	IE 5.5	None	5	4	3
22	IE 5.5	4	7	1	2
23	IE 5.5	4	6	1	2
24	IE 5.5	None	8	4	3
25	Net 6.0	RealPlayer	1	4	3

Keep mapping the variables (3)

Client operating system
1: Windows 95
2: Windows 98
3: Windows ME
4: Windows NT
5: Windows 2000
6: Windows XP
7: NOT USED
8: NOT USED

Servers
1; IIS
2: Apache
3: WebLogic
4: NOT USED

Server operating system
1: Windows NT
2: Windows 2000
3: Linux
4: NOT USED

Mapping

	Browser	Plug-in	Client OS	Server	Server OS
1	IE 5.0	None	Win 95	IIS	Win NT
2	IE 5.0	4	Win ME	4	4
3	IE 5.0	4	Win 98	4	4
4	IE 5.0	None	Win NT	IIS	Win NT
5	IE 5.0	MediaPlayer	Win 2000	WebLogic	Linux
6	IE 5.0	RealPlayer	7	Apache	Win 2000
7	IE 5.0	RealPlayer	Win XP	Apache	Win 2000
8	IE 5.0	MediaPlayer	8	WebLogic	Linux
9	IE 6.0	4	Win 95	WebLogic	Linux
10	IE 6.0	None	Win ME	Apache	Win 2000
11	IE 6.0	None	Win 98	Apache	Win 2000
12	IE 6.0	4	Win NT	WebLogic	Linux
13	IE 6.0	RealPlayer	Win 2000	IIS	Win NT
14	IE 6.0	MediaPlayer	7	4	4
15	IE 6.0	MediaPlayer	Win XP	4	4
16	IE 6.0	RealPlayer	8	IIS	Win NT
17	IE 5.5	MediaPlayer	Win 95	Apache	Win NT
18	IE 5.5	RealPlayer	Win ME	WebLogic	4
19	IE 5.5	RealPlayer	Win 98	WebLogic	4
20	IE 5.5	MediaPlayer	Win NT	Apache	Win NT
21	IE 5.5	None	Win 2000	4	Linux
22	IE 5.5	4	7	IIS	Win 2000
23	IE 5.5	4	Win XP	IIS	Win 2000
24	IE 5.5	None	8	4	Linux
25	Net 6.0	RealPlayer	Win 95	4	Linux
26	Net 6.0	MediaPlayer	Win ME	IIS	Win 2000
27	Net 6.0	MediaPlayer	Win 98	IIS	Win 2000
28	Net 6.0	RealPlayer	Win NT	4	Linux
29	Net 6.0	4	Win 2000	Apache	Win NT
30	Net 6.0	None	7	WebLogic	4
31	Net 6.0	None	Win XP	WebLogic	4
32	Net 6.0	4	8	Apache	Win NT
33	Net 6.1	RealPlayer	Win 95	4	Win 2000
34	Net 6.1	MediaPlayer	Win ME	IIS	Linux
35	Net 6.1	MediaPlayer	Win 98	IIS	Linux
36	Net 6.1	RealPlayer	Win NT	4	Win 2000
37	Net 6.1	4	Win 2000	Apache	4
38	Net 6.1	None	7	WebLogic	Win NT
39	Net 6.1	None	Win XP	WebLogic	1 Win NT
40	Net 6.1	4	8	Apache	4
41	Moz 1.1	MediaPlayer	Win 95	Apache	4
42	Moz 1.1	RealPlayer	Win ME	WebLogic	Win NT
43	Moz 1.1	RealPlayer	Win 98	WebLogic	Win NT
44	Moz 1.1	MediaPlayer	Win NT	Apache	4
45	Moz 1.1	None	Win 2000	4	Win 2000
46	Moz 1.1	4	7	IIS	Linux
47	Moz 1.1	4	Win XP	IIS	Linux
48	Moz 1.1	None	8	4	Win 2000
49	Net 7.0	4	Win 95	WebLogic	Win 2000
50	Net 7.0	None	Win ME	Apache	Linux
51	Net 7.0	None	Win 98	Apache	Linux
52	Net 7.0	4	Win NT	WebLogic	Win 2000
53	Net 7.0	RealPlayer	Win 2000	IIS	4
54	Net 7.0	MediaPlayer	7	4	Win NT
55	Net 7.0	MediaPlayer	Win XP	4	Win NT
56	Net 7.0	RealPlayer	8	IIS	4
57	Opera 7	None	Win 95	IIS	4
58	Opera 7	4	Win ME	4	Win NT
59	Opera 7	4	Win 98	4	Win NT
60	Opera 7	None	Win NT	IIS	4
61	Opera 7	MediaPlayer	Win 2000	WebLogic	Win 2000
62	Opera 7	RealPlayer	7	Apache	Linux
63	Opera 7	RealPlayer	Win XP	Apache	Linux
64	Opera 7	MediaPlayer	8	WebLogic	Win 2000

Pairwise testing Allpairspy 2.5

https://pypi.org/project/allpairspy/

```
from allpairspy import AllPairs

parameters = [
    ["Brand X", "Brand Y"],
    ["98", "NT", "2000", "XP"],
    ["Internal", "Modem"],
    ["Salaried", "Hourly", "Part-Time", "Contr."],
    [6, 10, 15, 30, 60],
]

print("PAIRWISE:")
for i, pairs in enumerate(AllPairs(parameters)):
    print("{:2d}: {}".format(i, pairs))
```

Pairwise testing Allpairspy 2.5

```
PAIRWISE:
 0: ['Brand X', '98', 'Internal', 'Salaried', 6]
 1: ['Brand Y', 'NT', 'Modem', 'Hourly', 6]
 2: ['Brand Y', '2000', 'Internal', 'Part-Time', 10]
 3: ['Brand X', 'XP', 'Modem', 'Contr.', 10]
 4: ['Brand X', '2000', 'Modem', 'Part-Time', 15]
 5: ['Brand Y', 'XP', 'Internal', 'Hourly', 15]
 6: ['Brand Y', '98', 'Modem', 'Salaried', 30]
 7: ['Brand X', 'NT', 'Internal', 'Contr.', 30]
 8: ['Brand X', '98', 'Internal', 'Hourly', 60]
 9: ['Brand Y', '2000', 'Modem', 'Contr.', 60]
10: ['Brand Y', 'NT', 'Modem', 'Salaried', 60]
11: ['Brand Y', 'XP', 'Modem', 'Part-Time', 60]
12: ['Brand Y', '2000', 'Modem', 'Hourly', 30]
13: ['Brand Y', '98', 'Modem', 'Contr.', 15]
14: ['Brand Y', 'XP', 'Modem', 'Salaried', 15]
15: ['Brand Y', 'NT', 'Modem', 'Part-Time', 15]
16: ['Brand Y', 'XP', 'Modem', 'Part-Time', 30]
17: ['Brand Y', '98', 'Modem', 'Part-Time', 6]
18: ['Brand Y', '2000', 'Modem', 'Salaried', 6]
19: ['Brand Y', '98', 'Modem', 'Salaried', 10]
20: ['Brand Y', 'XP', 'Modem', 'Contr.', 6]
21: ['Brand Y', 'NT', 'Modem', 'Hourly', 10]
```

Pairwise testing Allpairspy 2.5

```
    n = len(row)

    if n > 1:
        # Brand Y does not support Windows 98
        if "98" == row[1] and "Brand Y" == row[0]:
            return False

        # Brand X does not work with XP
        if "XP" == row[1] and "Brand X" == row[0]:
            return False

    if n > 4:
        # Contractors are billed in 30 min increments
        if "Contr." == row[3] and row[4] < 30:
            return False

    return True

parameters = [
    ["Brand X", "Brand Y"],
    ["98", "NT", "2000", "XP"],
    ["Internal", "Modem"],
    ["Salaried", "Hourly", "Part-Time", "Contr."],
    [6, 10, 15, 30, 60]
]

print("PAIRWISE:")
for i, pairs in enumerate(AllPairs(parameters, filter_func=is_valid_combination)):
    print("{:2d}: {}".format(i, pairs))
```

Pairwise testing Allpairspy 2.5

```
    n = len(row)

    if n > 1:
        # Brand Y does not support Windows 98
        if "98" == row[1] and "Brand Y" == row[0]:
            return False

        # Brand X does not work with XP
        if "XP" == row[1] and "Brand X" == row[0]:
            return False

    if n > 4:
        # Contractors are billed in 30 min increments
        if "Contr." == row[3] and row[4] < 30:
            return False

    return True

parameters = [
    ["Brand X", "Brand Y"],
    ["98", "NT", "2000", "XP"],
    ["Internal", "Modem"],
    ["Salaried", "Hourly", "Part-Time", "Contr."],
    [6, 10, 15, 30, 60]
]

print("PAIRWISE:")
for i, pairs in enumerate(AllPairs(parameters, filter_func=is_valid_combination)):
    print("{:2d}: {}".format(i, pairs))
```

Pairwise testing Allpairspy 2.5

```
PAIRWISE:
 0: ['Brand X', '98', 'Internal', 'Salaried', 6]
 1: ['Brand Y', 'NT', 'Modem', 'Hourly', 6]
 2: ['Brand Y', '2000', 'Internal', 'Part-Time', 10]
 3: ['Brand X', '2000', 'Modem', 'Contr.', 30]
 4: ['Brand X', 'NT', 'Internal', 'Contr.', 60]
 5: ['Brand Y', 'XP', 'Modem', 'Salaried', 60]
 6: ['Brand X', '98', 'Modem', 'Part-Time', 15]
 7: ['Brand Y', 'XP', 'Internal', 'Hourly', 15]
 8: ['Brand Y', 'NT', 'Internal', 'Part-Time', 30]
 9: ['Brand X', '2000', 'Modem', 'Hourly', 10]
10: ['Brand Y', 'XP', 'Modem', 'Contr.', 30]
11: ['Brand Y', '2000', 'Modem', 'Salaried', 15]
12: ['Brand Y', 'NT', 'Modem', 'Salaried', 10]
13: ['Brand Y', 'XP', 'Modem', 'Part-Time', 6]
14: ['Brand Y', '2000', 'Modem', 'Contr.', 60]
```

Functional testing (Howden)

In functional testing, a program P is viewed as a function transforming input vector X_i into an output vector Y_i such that $Y_i=P(X_i)$

Examples:

Y_1= sqrt(X_1)

Y_2= c-compiler (X2) produces object code from C program X_2

Y_3 = sort (X_3) sorting X_3 = {A, N}

Y_4 = telephone switch (X_4)

X_4={off hook, on hook, phone number, voice}
Y_4 ={idle, dial, ring, fast busy tone, slow busy tone, voice}

Combinatorial testing problems: possible solutions

Pairwise testing:

Instead of considering all combinations of the input variables construct a set of test cases that "covers" *all* combinations of the test data for each pair of variables

Use of input-output analysis:

Determine (discover) relationship between inputs and output; not all inputs affect each output

Functional testing (Howden)

Key concepts:

Precise definition of domain of input and output variables

Selection of values from the data domain of each variable having important properties

Consideration of combination of special values from different input domains to design test cases

Consideration of input values such that the program under test produces special values from the domains of the output variables

Test vector

Test vector (test data) – instance of the input to the program

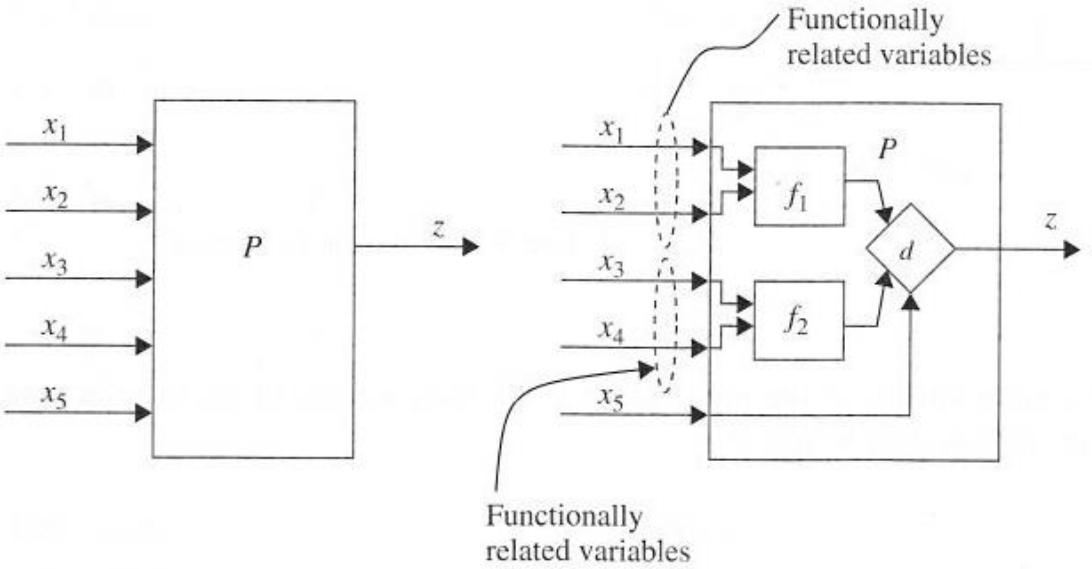
n-variables assuming k_1, k_2, …k_n special values

$k_1 * k_2 * \ldots * k_n$ possible combinations of test data

Reduction of number of input combinations:

No need of combining values of all input variables to design a test vector if the variables are *functionally* related

Example



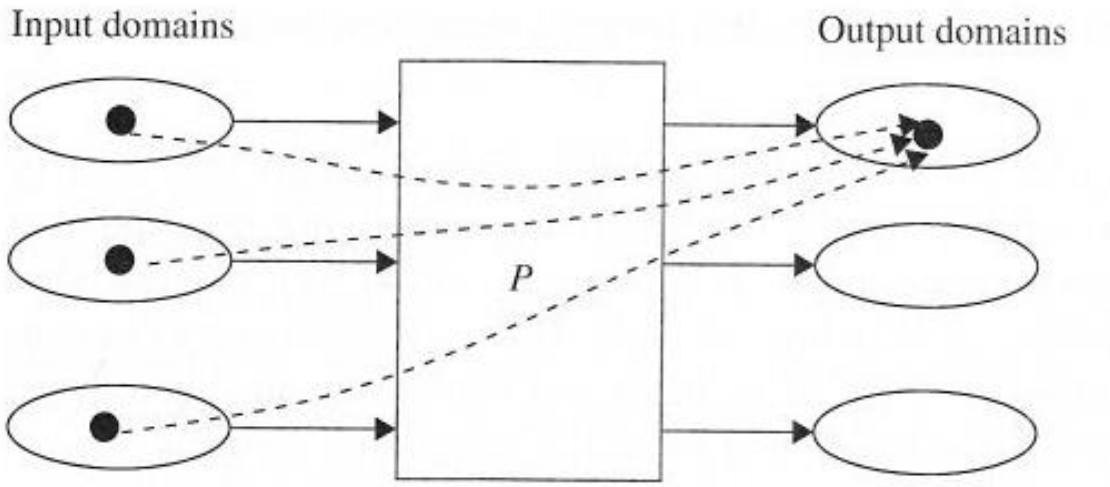
$x_1, x_2, \ldots, x_4$ assume 3 special values, x_5 – Boolean variable
All combinations $3^4*2 = 162$.

structure

f_1 uses x_1 and $x_2 \rightarrow 3^2=9$
f_2 uses x_3 and $x_4 \rightarrow 3^2=9$
decision box $\rightarrow 2$

Total = (9+9)*2 = 36 combinations

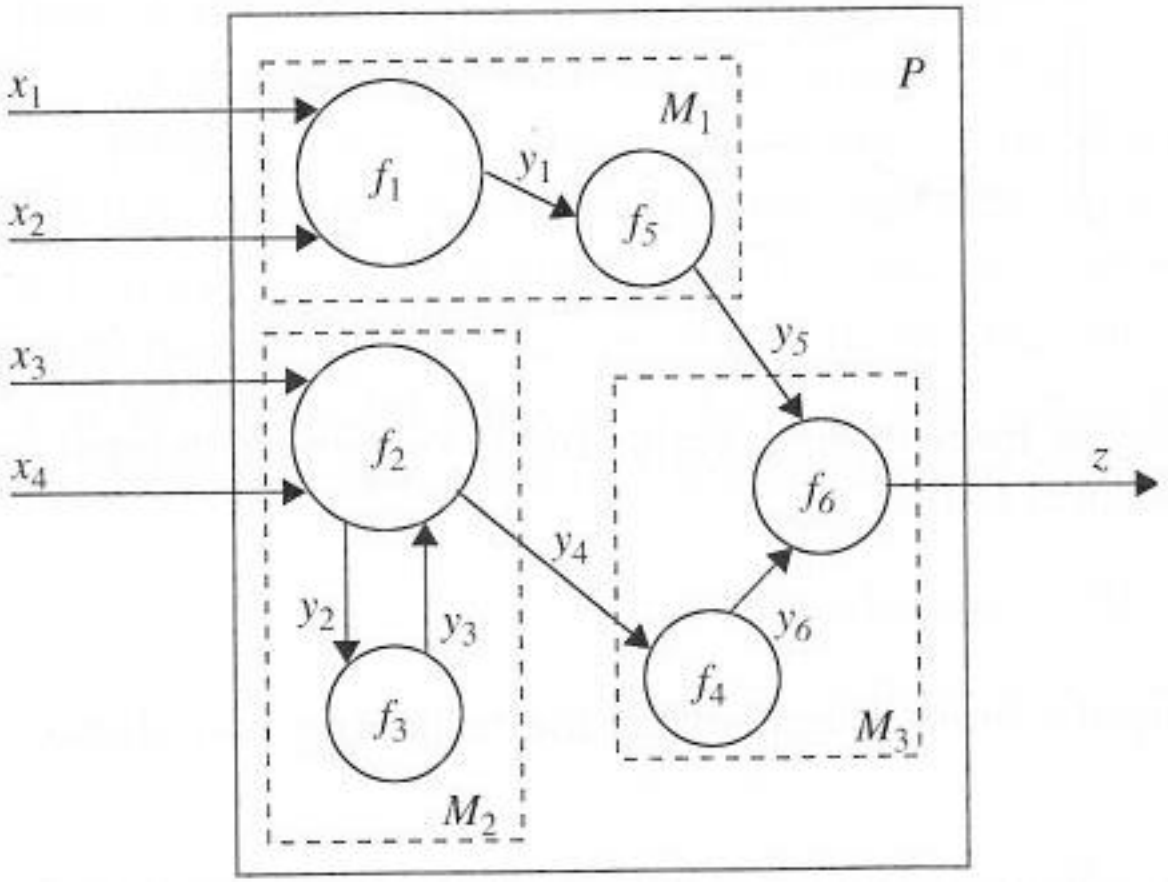
Generating test data- analysis of input domains



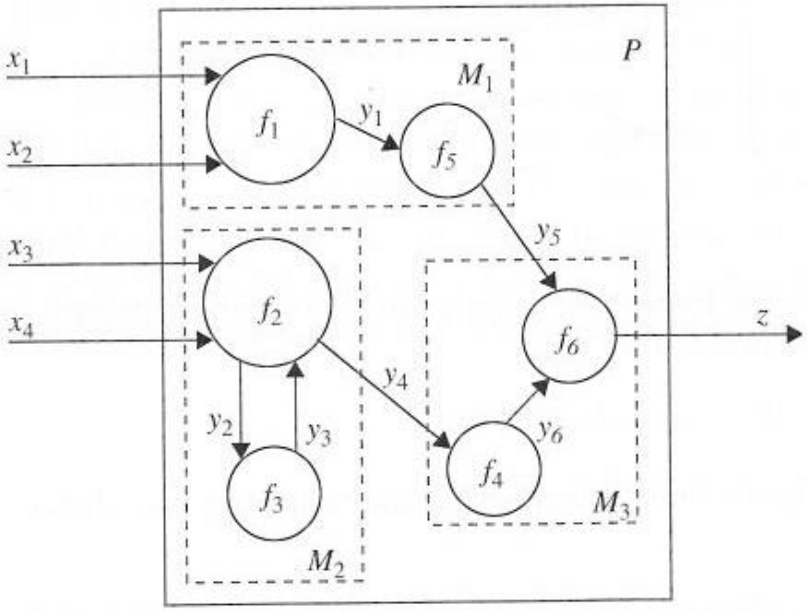
The number of test cases obtained from the analysis of the input domains will be high

Generation of the expected output for a test vector is simple (specification)

Functional testing: a general view and levels of decomposition (1)



Functional testing: a general view and levels of decomposition (2)



Entity Name	Input Variables	Output Variables
P	$\{x_1, x_2, x_3, x_4\}$	$\{z\}$
M_1	$\{x_1, x_2\}$	$\{y_5\}$
M_2	$\{x_3, x_4\}$	$\{y_4\}$
M_3	$\{y_4, y_5\}$	$\{z\}$
f_1	$\{x_1, x_2\}$	$\{y_1\}$
f_2	$\{x_3, x_4, y_3\}$	$\{y_2, y_4\}$
f_3	$\{y_2\}$	$\{y_3\}$
f_4	$\{y_4\}$	$\{y_6\}$
f_5	$\{y_1\}$	$\{y_5\}$
f_6	$\{y_5, y_6\}$	$\{z\}$

Input-output analysis

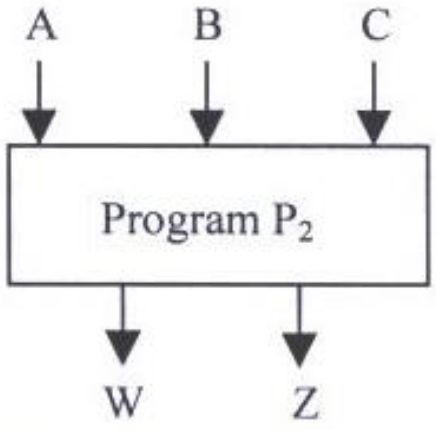
Not all inputs affect every output

Consider output y
Denote by T(y) the set of all combinatorial tests with respect to this output

The set of all combinatorial tests T(**Y**) is taken as

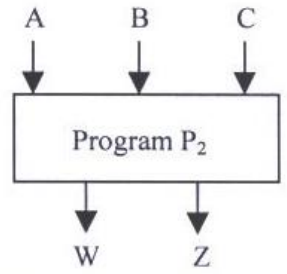
$$T(\mathbf{Y}) = \bigcup_{y \in \mathbf{Y}} T(y)$$

Input-output analysis: example



Input Var.	Test Data Values
A	1.5, 3.6
B	North, South, East, West
C	TDC, BDM

Input-output analysis: example

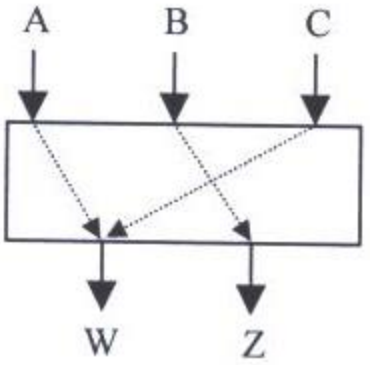


Input Var.	Test Data Values
A	1.5, 3.6
B	North, South, East, West
C	TDC, BDM

Test ID	Input A	Input B	Input C
C_1	1.5	North	TDC
C_2	1.5	North	BDM
C_3	1.5	South	TDC
C_4	1.5	South	BDM
C_5	1.5	East	TDC
C_6	1.5	East	BDM
C_7	1.5	West	TDC
C_8	1.5	West	BDM
C_9	3.6	North	TDC
C_{10}	3.6	North	BDM
C_{11}	3.6	South	TDC
C_{12}	3.6	South	BDM
C_{13}	3.6	East	TDC
C_{14}	3.6	East	BDM
C_{15}	3.6	West	TDC
C_{16}	3.6	West	BDM

Input-output analysis: example

Relationships between inputs and outputs are known



Tests from the Perspective of Output W

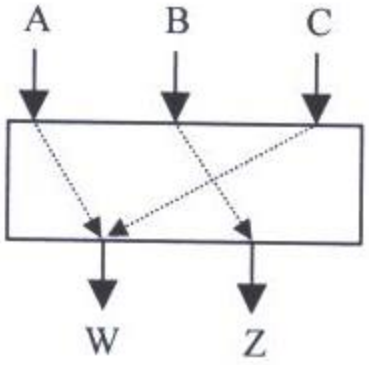
Test ID	Input A	Input B	Input C
IO_1	1.5	North	TDC
IO_2	1.5	North	BDM
IO_3	3.6	North	TDC
IO_4	3.6	North	BDM

Tests from the Perspective of Output Z

Test ID	Input A	Input B	Input C
IO_5	1.5	North	TDC
IO_6	1.5	South	TDC
IO_7	1.5	East	TDC
IO_8	1.5	West	TDC

Input-output analysis: example

Relationships between inputs and outputs are known



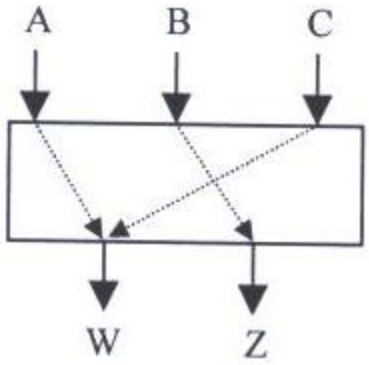
Union of Test Sets for Outputs W and Z

Test ID	Input A	Input B	Input C
IO_1	1.5	North	TDC
IO_2	1.5	North	BDM
IO_3	3.6	North	TDC
IO_4	3.6	North	BDM
IO_6	1.5	South	TDC
IO_7	1.5	East	TDC
IO_8	1.5	West	TDC

IO_5 is a duplicate of IO_1

Input-output analysis: example

Minimal set of tests



Minimal Test Set for Program P_2

Test ID	Input A	Input B	Input C
MIN_1	1.5	North	TDC
MIN_2	1.5	South	BDM
MIN_3	3.6	East	TDC
MIN_4	3.6	West	BDM

Test Patterns

Like design patterns, test patterns provide guidelines and strategies to design tests

Help communicate the intent of the testing approach and share test design techniques in a way it is *understandable* and actionable

Use of templates

Test Patterns – Templates (1)

Attributes:

Name - gives a memorable name

Problem – one-sentence description of the problem the pattern solves

Analysis – describes the problem area

Design – explains how this pattern is executed (the pattern turns from design into test cases)

Test Patterns – Templates (2)

Attributes (cont.):

Oracle- explains the expected results

Examples– lists examples of how this pattern finds faults

Pitfalls or limitations - explains under which circumstances this pattern should be avoided

Related patterns– lists any related patterns (if known)

Test Patterns – Example BVA

Name	Boundary-Value Analysis (BVA)
Problem	Many errors in software occur near the edges of physical and data boundaries. For example, using > instead of >=, or off-by-one indexing errors (zero-based vs. one-based).
Analysis	Choose test cases on or near the boundaries of the input domain of variables, with the rationale that many defects tend to concentrate near the extreme values of inputs. A classic example of boundary-value analysis in security testing is to create long input strings to probe potential buffer overflows. More generally, insecure behavior in boundary cases is often unforeseen by developers, who tend to focus on nominal situations instead.
Design	For each input value, determine the minimum and maximum allowed value (min and max). Design a set of test cases that tests min, max, min – 1, and max + 1 (note that BVA is sometimes defined to include min + 1 and max – 1). Test cases should include the following: ❒ Input(s) to the component ❒ Partition boundaries exercised ❒ Expected outcome of the test case

Test Patterns – Example BVA

Name	Boundary-Value Analysis (BVA)
Oracle	Minimum and maximum values are expected to pass. Values outside of this range are expected to fail gracefully.
Examples	For an input field that expects a number between 1 and 10, test 0, 1, 10, and 11.
Pitfalls or Limitations	Boundaries are not always known. Knowledge of the domain or access to source code might be necessary to achieve useful boundary-value analysis. If the input range contains "special" values—values within the boundaries that are handled differently by the application—BVA might miss issues with these values.
Related Patterns	Equivalence Class Partitioning

Unit testing in Python

Unit testing in python (1)

unittest (actually unittest2), PyUnit; see also Pytest

a series of special assertion methods in the unittest.TestCase class

```
import unittest2

class Test(unittest2.TestCase):

    def test1(self):
        self.assertEqual(4+5,9)

    def test2(self):
                self.assertIn(9,[5,4,3])

if __name__ == '__main__':
    unittest2.main()
```

Unit testing in python (2)

```
class Calculate(object):
    def add(self,x,y):
        return x+y


import unittest2


class Test2Calculate(unittest2.TestCase):
    def setUp(self):
        self.calc=Calculate()

    def test_add_method_returns_correct_result(self):
        self.assertEqual(4,self.calc.add(0,2))

if __name__ == '__main__':
    unittest2.main()
```

```python
import unittest2


class Test(unittest2.TestCase):

    def test0(self):
        self.assertEqual(4+5,9)


    def test1(self):
        message="result of addition"
        self.assertEqual(4+5,6,message)


    def test2(self):
        self.assertAlmostEqual(4+5,9.3, delta=0.1)


    def test3(self):
        message="element not on the list"
        self.assertIn(6,[5,6,10],message)


    def test4(self):
        message="lists are not equal"
        self.assertItemsEqual([1,2,3],[3,2,1],message)


    def test5(self):
        message = "lists are not equal"
        self.assertItemsEqual([1, 2, 3], [3, 7, 1], message)
if __name__ == '__main__':
    unittest2.main()
```

```
.FF..F
======================================================================
FAIL: test1 (__main__.Test)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/Users/witold/Desktop/pythonProject/test/main.py", line 25, in test1
    self.assertEqual(4+5,6,message)
AssertionError: 9 != 6 : result of addition

======================================================================
FAIL: test2 (__main__.Test)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/Users/witold/Desktop/pythonProject/test/main.py", line 28, in test2
    self.assertAlmostEqual(4+5,9.3, delta=0.1)
AssertionError: 9 != 9.3 within 0.1 delta
======================================================================
```

```python
import unittest2


class Test(unittest2.TestCase):

    def test0(self):
        self.assertEqual(4+5,9)


    def test1(self):
        message="result of addition"
        self.assertEqual(4+5,6,message)


    def test2(self):
        self.assertAlmostEqual(4+5,9.3, delta=0.1)


    def test3(self):
        message="element not on the list"
        self.assertIn(6,[5,6,10],message)


    def test4(self):
        message="lists are not equal"
        self.assertItemsEqual([1,2,3],[3,2,1],message)


    def test5(self):
        message = "lists are not equal"
        self.assertItemsEqual([1, 2, 3], [3, 7, 1], message)
if __name__ == '__main__':
    unittest2.main()
```

```
======================================================================
FAIL: test5 (__main__.Test)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/Users/witold/Desktop/pythonProject/test/main.py", line 40, in test5
    self.assertItemsEqual([1, 2, 3], [3, 7, 1], message)
AssertionError: Sequences differ: [1, 2, 3] != [1, 3, 7]

First differing element 1:
2
3

- [1, 2, 3]
?    ^  ^

+ [1, 3, 7]
?    ^  ^
 : lists are not equal

----------------------------------------------------------------------
Ran 6 tests in 0.001s

FAILED (failures=3)
```

Set of assertions

Method	Checks that	New in
assertEqual(a, b)	a == b	
assertNotEqual(a, b)	a != b	
assertTrue(x)	bool(x) is True	
assertFalse(x)	bool(x) is False	
assertIs(a, b)	a is b	2.7
assertIsNot(a, b)	a is not b	2.7
assertIsNone(x)	x is None	2.7
assertIsNotNone(x)	x is not None	2.7
assertIn(a, b)	a in b	2.7
assertNotIn(a, b)	a not in b	2.7
assertIsInstance(a, b)	isinstance(a, b)	2.7
assertNotIsInstance(a, b)	not isinstnace(a, b)	2.7
assertRaises(exc, fun, *args*, *kwds)	fun(*args, **kwds) raises exc	
assertRaisesRegexp(exc, r, fun, *args*, *kwds)	round(a-b, 7) == 0	2.7
assertAlmostEqual(a, b)	round(a-b, 7) == 0	
assertNotAlmostEqual(a, b)	round(a-b, 7) != 0	
assertGreater(a, b)	a > b	2.7
assertGreaterEqual(a, b)	a >= b	
assertLess(a, b)	a < b	2.7
assertLessEqual(a, b)	a <= b	2.7
assertRegexpMatches(s, r)	r.search(s)	2.7
assertNotRegexpMatches(a, b)	not r.search(s)	2.7
assertItemsEqual(a, b)	sorted(a) == sorted(b)Works with unhashable objects	
assertDictContainsSubset(a, b)	All the key/value pairs in a exist in b	2.7

Black Box Testing: Summary

Focus on external behavior (functional testing)

High level of abstraction (large software systems or substantial parts of the system)

Timeline: early (before any code completed) and late (system and acceptance testing)

Defect focus: external functions

Defect fixing: difficult, problems with interfaces and interactions

Tester: professional tester