北京交通大学

Software Quality Assurance and Testing Technology

2nd Semester, Spring 2022

Haiming Liu

School of Software Engineering

Beijing Jiaotong University

北京交通大学

Black-box Testing
Random Testing
Equivalence Partitioning
Boundary Value Analysis
Cause-Effect Analysis
Error Guessing
STATE TESTING

Cause-Effect Analysis

Cause-Effect Analysis is a systematic means for generating test cases to cover **different combinations of input "Causes"** resulting in **output "Effects."**

A CAUSE may be thought of as a distinct input condition, or an "equivalence class" of input conditions.

An EFFECT may be thought of as a distinct output condition, or a meaningful change in program state.

Causes and Effects are represented as Boolean variables and the logical relationships among them CAN (but need not) be represented as one or more boolean graphs.

Cause-Effect Analysis

- Black-box technique to analyze combinations of input conditions
- Identify causes and effects in specification

↓	↓
inputs	outputs
current state	new state

- Make Boolean Graph linking causes and effects
- Annotate impossible combinations of causes and effects
- Develop decision table from graph with in each column a particular combination of inputs and outputs
- Transform each column into test case

北京交通大学

Cause-Effect Analysis Sample

Given inputs maxint and N compute result :

$$result = \sum_{K=0}^{|N|} k \qquad if\ this <= maxint,\ error\ otherwise$$

- **Valid equivalence classes :**

condition	valid eq. classes
abs(N)	$N < 0,\ N \geq 0$
maxint	$\sum k \leq$ maxint, $\sum k >$ maxint

- **Test Cases :**

maxint	N	result	maxint	N	result
55	10	55	100	0	0
54	10	error	100	-1	1
56	10	55	100	1	1
0	0	0	…	…	…

北京交通大學

C-E Analysis Process Steps

Cause-effect analysis consists of the following 4 steps

- List and label causes (input-events) and effects (output actions) for a module
- Draw a cause-effect graph describing the logical combinations of causes, intermediate causes and resulting effects
- Develop a decision table (causes vs effects) from the graph
- Convert into test cases

北京交通大學

C-E Analysis Process Steps

Cause-effect analysis consists of the following 4 steps

- List and label causes (input-events) and effects (output actions) for a module
- Draw a cause-effect graph describing the logical combinations of causes, intermediate causes and resulting effects
- Develop a decision table (causes vs effects) from the graph
- Convert into test cases

STEP 1

Given inputs maxint and N compute result :

$$result = \sum_{K=0}^{|N|} k \quad \text{if this <= maxint, error otherwise}$$

Input Causes

- $\sum k \leq$ **maxint**
- $\sum k >$ **maxint**
- $N < 0$
- $N \geq 0$

Output Effects

- $\sum k$
- ***error***

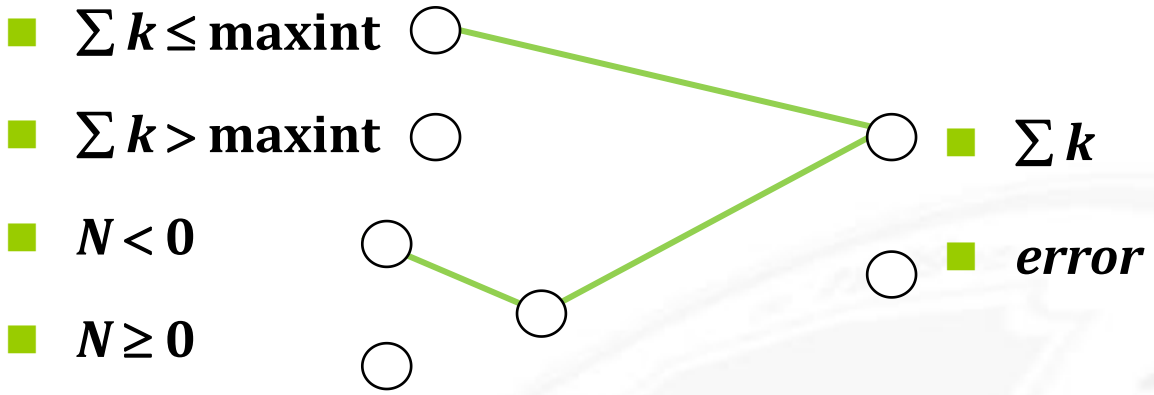
北京交通大学

C-E Analysis Process Steps

Cause-effect analysis consists of the following 4 steps

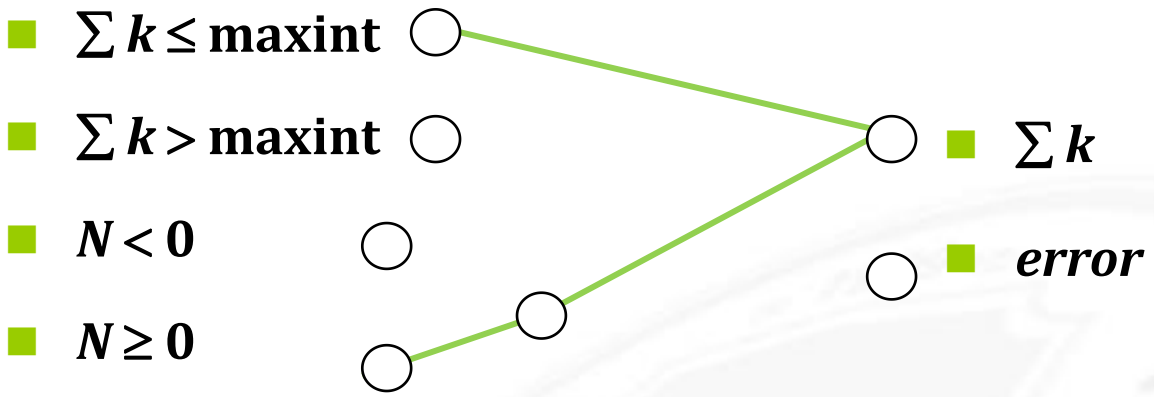
- List and label causes (input-events) and effects (output actions) for a module
- Draw a cause-effect graph describing the logical combinations of causes, intermediate causes and resulting effects
- Develop a decision table (causes vs effects) from the graph
- Convert into test cases

STEP 2



Causes Inputs	$\sum k \le$ maxint	1
	$\sum k >$ maxint	0
	$N < 0$	1
	$N \ge 0$	0
Effects Outputs	$\sum k$	1
	error	0

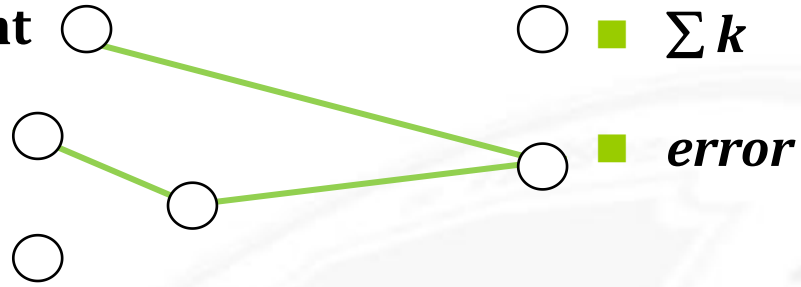
STEP 2



Causes Inputs	$\sum k \le$ maxint	1
	$\sum k >$ maxint	0
	$N < 0$	0
	$N \ge 0$	1
Effects Outputs	$\sum k$	1
	error	0

STEP 2

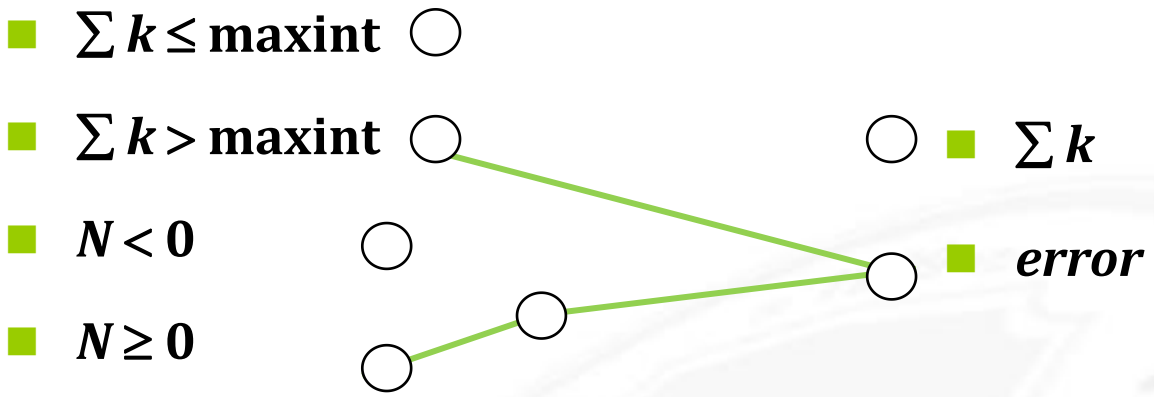
- $\sum k \leq \text{maxint}$
- $\sum k > \text{maxint}$
- $N < 0$
- $N \geq 0$



- $\sum k$
- *error*

Causes	**Inputs**	
	$\sum k \leq \text{maxint}$	0
	$\sum k > \text{maxint}$	1
	$N < 0$	1
	$N \geq 0$	0
Effects	**Outputs**	
	$\sum k$	0
	error	1

STEP 2



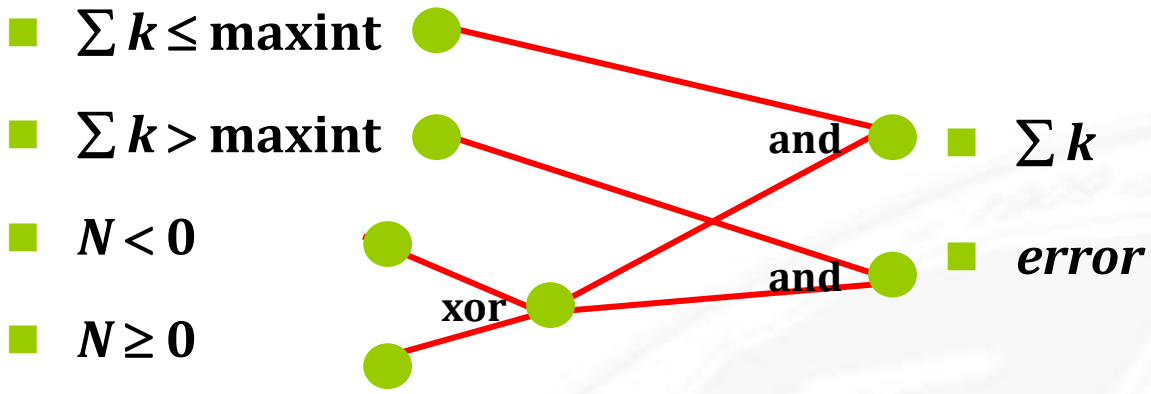
Causes Inputs	$\sum k \le$ maxint	0
	$\sum k >$ maxint	1
	$N < 0$	0
	$N \ge 0$	1
Effects Outputs	$\sum k$	0
	error	1

C-E Analysis Process Steps

Cause-effect analysis consists of the following 4 steps

- List and label causes (input-events) and effects (output actions) for a module
- Draw a cause-effect graph describing the logical combinations of causes, intermediate causes and resulting effects
- Develop a decision table (causes vs effects) from the graph
- Convert into test cases

STEP 3



Causes Inputs	$\sum k \le$ maxint	1	1	0	0
	$\sum k >$ maxint	0	0	1	1
	$N < 0$	1	0	1	0
	$N \ge 0$	0	1	0	1
Effects Outputs	$\sum k$	1	1	0	0
	error	0	0	1	1

Cause-Effect Analysis Sample

Cause	$\sum k \leq$ maxint	1	1	0	0
	$\sum k >$ maxint	0	0	1	1
	$N < 0$	1	0	1	0
	$N \geq 0$	0	1	0	1
Effects	$\sum k$	1	1	0	0
	error	0	0	1	1

北京交通大学

Black-box Testing
Random Testing
Equivalence Partitioning
Boundary Value Analysis
Cause-Effect Analysis
Error Guessing
STATE TESTING

Error Guessing

Testers utilize intuition and experience to identify potential errors and design test cases to reveal them.

Guidelines:

- Design tests for **reasonable but incorrect assumptions** that may have been made by developers.
- Design tests to detect errors in handling **special situations** or cases.
- Design tests to explore **unexpected or unusual** program use or environmental scenarios.

Error Guessing

Examples of conditions to explore:

(1) Repeated instances or occurrences

(2) Blanks or null characters in strings (etc.)

(3) Negative numbers

(4) Non-numeric values in numeric fields

(5) Inputs that are too long or two short Using intuition and experience, identify tests you would

Error Guessing

- Just ‘guess’ where the errors are ……
- Intuition and experience of tester
- Strategy:
  - ✓ Make a list of possible errors or error-prone situations (often related to boundary conditions)
  - ✓ Write test cases based on this list

Error Guessing

- **More sophisticated 'error guessing'**: **Risk Analysis**
- **Try to identify critical parts of program** (high risk code sections):
  - ✓ parts with unclear specifications
  - ✓ developed by junior programmer while his wife was pregnant ……
  - ✓ complex code :
    measure code complexity - tools available (Logiscope,…)

- ***High-risk code will be more thoroughly tested***
  (or be rewritten immediately ……)

STATE TESTING

All software flows from one **state** to another.

For some software, these **state changes** are obvious:

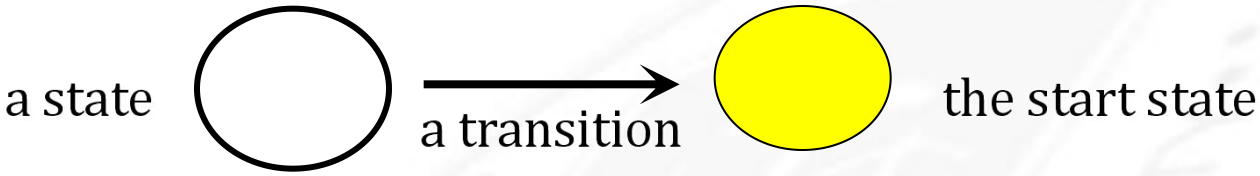
Paint Program

To determine states, we need to build a **state transition diagram or state transition map.**

- Good requirements or specifications often include these.
- There are software tools for drawing them.
- Different diagramming techniques exist --- we'll use just one, but it is not the only one possible.

Definition: State Transition Map

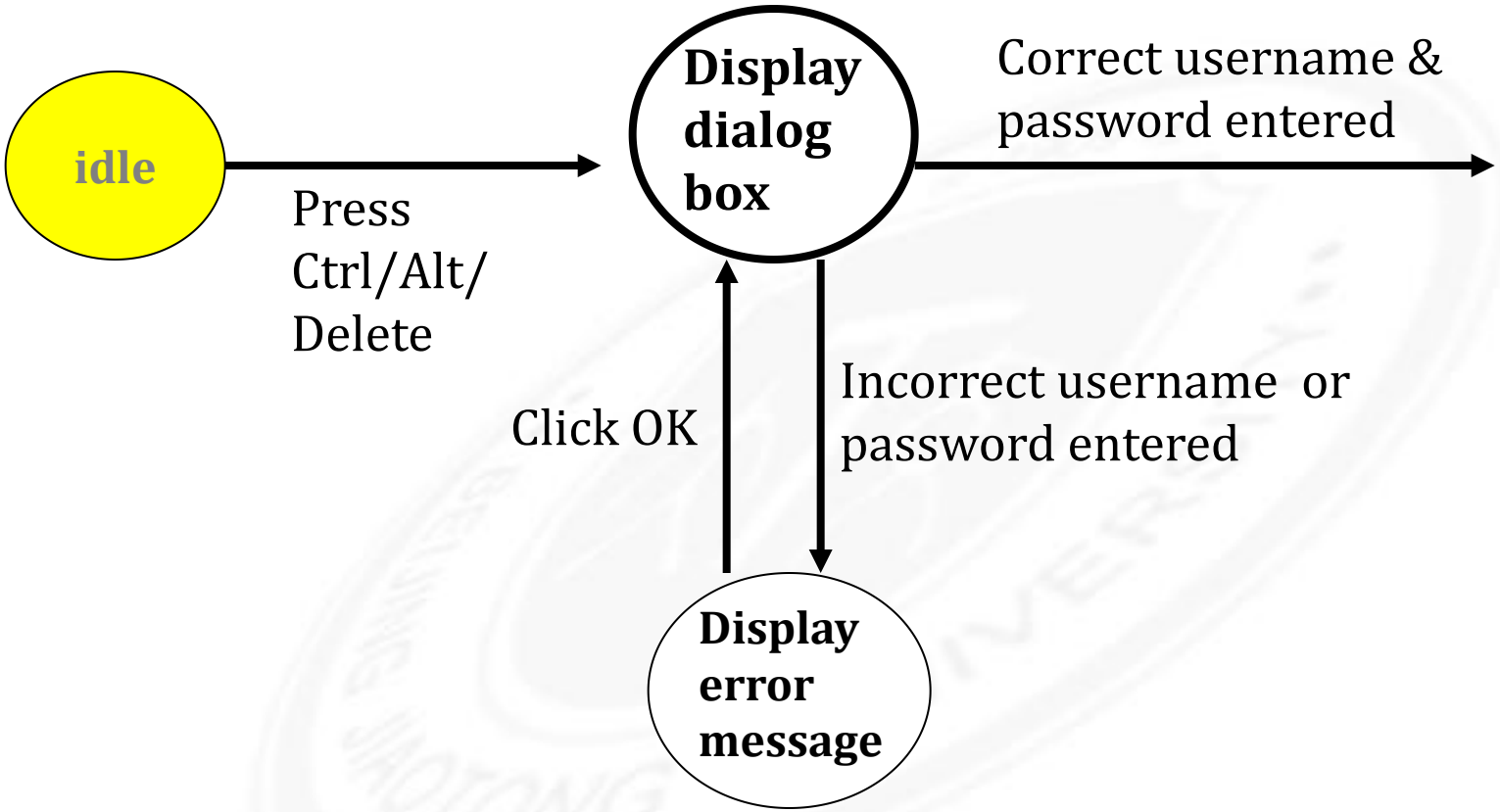
A STM is a graph showing the logic flow from state to state in the program. This is diagrammed using the symbols:



All of these are labeled

- States- to identify them
- Transitions- to identify what triggers movement from one state to another state.

Example of Part of STM



Obviously, these can become quite large!!

What the STM Shows

- Each unique state of the software.
- Input or condition needed to move from one state to the next.
- What is set or what output is produced when a state is entered or exited.

Obviously, we can't investigate all possible paths through the state transition map (this is really the "traveling salesperson problem").

Use Equivalence Partitioning to Choose Test Cases for State Testing

Possible choices for partitioning:

- Try to visit each state at least once.
- Test the most common or popular state-to-state transitions.
- Test the least common paths between states.
- Test all entrances and exits from error states.
- Test random state transitions --- i.e. throw darts a the transition state map!

All of these are testing-to-pass cases.

More on State Testing

- Involves checking all **state variables** which define a state.

- It is suggest that checking with the spec writers and programmers to identify possible states --- but, caution

DO NOT GO TO THE CODE LEVEL!

- *Programmers will tend to want to drag out code if it exists!*

Testing-to-Fail State Testing

Check **race conditions** – a timing problem causes the execution to not proceed as planned.

Try interrupting the program in the middle of its execution as see what happens.

- Is data lost?
- Can the program be restarted in a clean condition?
- Start two instances of the same program and input to both.
- What happens?

Repetition, Stress & Load Testing

Repetition testing can often find memory leaks --- memory not freed completely when it should be.

Stress testing tries to run under bad conditions --- low memory, slow CPU, etc.

Load testing overloads the program with huge data files, long periods of execution, etc.

Round all of this out by just playing like a dumb user! (But, call them an inexperienced user, please!)

Which One to Choose?

- **Black-box testing techniques** *:*
  - ✓ *Equivalence partitioning*
  - ✓ *Boundary value analysis*
  - ✓ *Cause-effect analysis*
  - ✓ *Error guessing*
  - ✓ *……*

Test derivation from formal specification, Which one to use ?

- ✓None of them is complete
- ✓All are based on some kind of heuristics
- ✓They are complementary

Which One to Choose?

- **Always use a combination of techniques**

  - ✓ *When a formal specification is available try to use it*
  - ✓ *Identify valid and invalid input equivalence classes*
  - ✓ *Identify output equivalence classes*
  - ✓ *Apply boundary value analysis on valid equivalence classes*
  - ✓ *Guess about possible errors*
  - ✓ *Cause-effect graphing for linking inputs and outputs*

北京交通大学

To be continued...

See you next week