Test Case Generation

- Test Input to the Software
- Some researchers/authors also define the test case to contain the expected output for the test input

- We will discuss
  - Category-partition Method
  - Iterative-relaxation Method
  - AI Planning Method
Category-partition Method

• Key idea
  – Method for creating functional test suites
  – Role of test engineer
    • Analyze the system specification
    • Write a series of formal test specifications
  – Automatic generator
    • Produces test descriptions
Steps

• Decompose the functional specification into functional units
  - Characteristics of functional units
    • They can be tested independently
    • Examples
      - A top-level user command
      - Or a function

• Decomposition may require several stages

• Similar to high-level decomposition done by software designers
  - May be reused, although independent decomposition is recommended
Steps

• Examine each functional unit
  - Identify parameters
    • Explicit input to the functional unit
  - Environmental conditions
    • Characteristics of the system’s state

• Test Cases
  - Specific values of parameters
  - And environmental conditions
Steps

- “Test cases are chosen to maximize chances of finding errors”
- For each parameter & environmental condition
  - Find categories
    - Major property or characteristic
    - Examples
      - Browsers, Operating Systems, array size
    - For each category
      - Find choices
        » Examples: (IE 5.0, IE 4.5, Netscape 7.0), (Windows NT, Linux), (100, 0, -1)
Steps

• Develop “Formal Test Specification” for each functional unit
  - List of categories
  - Lists of choices within each category

• Constraints

• Automatically produces a set of “test frames”
  - Consists of a set of choices
Related Techniques

- **Condition-table Method**
  - Conditions are categories
  - Value set for each condition is the set of choices
  - Constraints are restricted to categories

- **Cause-effect graph**
  - Identification of functions
  - Identify causes that influence the function’s behavior
  - And all effects
  - Construct a graph that relates causes to effects
Related Techniques

- **Revealing Sub-domains**
  - Contains elements that are either all processed correctly or all incorrectly
  - Executing one element of the sub-domain is sufficient
  - Error specific, not general

- **Equivalence Partitioning**
  - Partition function’s input domain into a procedure partition
  - Uses both specifications and code
Iterative Relaxation

• **Key Idea**
  - Path-oriented testing
  - Problem: Generation of test data that causes a program to follow a given path

• **Technique**
  - Choose arbitrary input
  - Iteratively refine it until all the branch predicates on the given path evaluate to the desired outcome
Example Program

BEGIN
read(x,y,z)

1
u=(x-y)*2

2
w=u

3
w=y

P1
x>y

P2
(w+z)>100

P3
x^2+z^2≥100

P4
u>0

P5
(y-sin(z))>0

write(u)

write(“linear”)

write(“nl:quad”)

write(“nl:sine”)

END
BEGIN
read(x, y, z)

u = (x - y) * 2

w = u

(x > y)
P1

(w + z) > 100
P2

(2x - 2y + z) > 100

x = x - 2
P4

y = y + w

write("linear")

u > 0
P5

(y - sin(z)) > 0

write("nl:sine")

write(u)

y = x * z + 1
P3

x^2 + z^2 ≥ 100

write("nl:quad")

END

input variables x, y, z
Representation

- **E1 OP E2**
  - $(E1 - E2) \text{ OP } 0$

- **If the branch statement contains a Boolean variable, represent**
  - TRUE value by a +ve number
  - FALSE value by a –ve number

- **If (A) then ...**
  - If $(A >= 0)$ then ...

- **(A AND B)**
  - Both $((A >= 0) \text{ and } (B >= 0))$

- **(A OR B)**
  - Either $((A >= 0) \text{ or } (B >= 0))$
BEGIN
read(x, y, z)

u = (x - y) * 2

w = u

(x - y) > 0

P1
x > y

P2
(w + z) > 100

P3
x^2 + z^2 ≥ 100

y = x * z + 1

write("nl:quad")

u - 0 > 0

P4
u > 0

P5
(y - sin(z)) > 0

write("nl:sine")

write(u)

END

input variables x, y, z
Predicate Slice

• **Definition:**
  - The predicate slice of a branch predicate BP on a path is a set of statements that computes values upon which the value of BP may be directly or indirectly data dependent.
BEGIN
read(x,y,z)
\[ u = (x-y) \times 2 \]
\[ x = x - 2 \]
\[ y = y + w \]
write("linear")
\[ w = u \]
\[ w = y \]
\[ (w+z) > 100 \]
\[ x^2 + z^2 \geq 100 \]
\[ y = x \times z + 1 \]
write("nl:quad")
\[ u > 0 \]
write(u)
\[ (y-\sin(z)) > 0 \]
write("nl:sine")
\[ END \]

Input Dependency Set
\{x, y\}
BEGIN
read(x,y,z)

u=(x-y)*2

P1

x>y

P2

w=u

(w+z)>100

P3

x^2+z^2≥100

y=x*z+1

write("nl:quad")

P4

u>0

write(u)

Input Dependency Set
{x, y, z}

P5

(y-sin(z))>0

write("nl:sine")

END
BEGIN
read(x, y, z)
u = (x - y) * 2
w = u
x > y
P1
x = x - 2
y = y + w
write("linear")

P2
(w + z) > 100
P3
x^2 + z^2 ≥ 100
y = x * z + 1
write("nl: quad")

P4
u > 0
write(u)

P5
(y - sin(z)) > 0
write("nl: sine")

END

Input Dependency Set
{x, y}
BEGIN

read(x,y,z)

\( u=(x-y)*2 \)

\( (w+z)>100 \)

\( x=x-2 \)

\( y=y+w \)

write("linear")

\( x^2+z^2 \geq 100 \)

\( y=x*z+1 \)

write("nl:quad")

input variables \( x, y, z \)

END
Values of $x, y, z$

$x - y > 0$

$(2x - 2y + z) - 100 > 0$

$2x - 2y - 0 > 0$

let $y = 0$

$x > 0$

let $x = 1$

$z > 100 - 2x + 2y$

$100 - 2 = 98$

${x, y, z} = \{1, 0, 99\}$
BEGIN
0 read(x,y,z)
1 u=(x-y)*2
2 w=u
3 w=y
4 x=x-2
5 y=y+w
6 write("linear")
7 y=x*z+1
8 write("nl:quad")
9 write(u)
10 write("nl:sine")

END

{x, y, z} = {1, 0, 99}

1 > 0

(1 - 0) * 2 + 99 > 100

(1 - 0) * 2 > 0
AI Planning Method

• **Key Idea**
  - Input to Command-driven software is a sequence of commands
  - The sequence is like a plan

• **Scenario to test**
  - Initial state
  - Goal state
Example

- **VCR command-line software**
- **Commands**
  - **Rewind**
  - If at the end of tape
  - **Play**
  - If fully rewound
  - **Eject**
  - If at the end of tape
  - **Load**
  - If VCR has no tape
Preconditions & Effects

- **Rewind**
  - Precondition: If at end of tape
  - Effects: At beginning of tape

- **Play**
  - Precondition: If at beginning of tape
  - Effects: At end of tape

- **Eject**
  - Precondition: If at end of tape
  - Effects: VCR has no tape

- **Load**
  - Precondition: If VCR has no tape
  - Effects: VCR has tape
Preconditions & Effects

- **Rewind**
  - Precondition: end_of_tape
  - Effects: \(\neg\)end_of_tape

- **Play**
  - Precondition: \(\neg\)end_of_tape
  - Effects: end_of_tape

- **Eject**
  - Precondition: end_of_tape
  - Effects: \(\neg\)has_tape

- **Load**
  - Precondition: \(\neg\)has_tape
  - Effects: has_tape
Initial and Goal States

• Initial State
  - end_of_tape

• Goal State
  - ¬end_of_tape

• Plan?
  - Rewind
Initial and Goal States

• Initial State
  - \neg end\_of\_tape \& has\_tape

• Goal State
  - \neg has\_tape

• Plan?
  - Play
  - Eject