# *SYSC 4101 / SYSC5105*

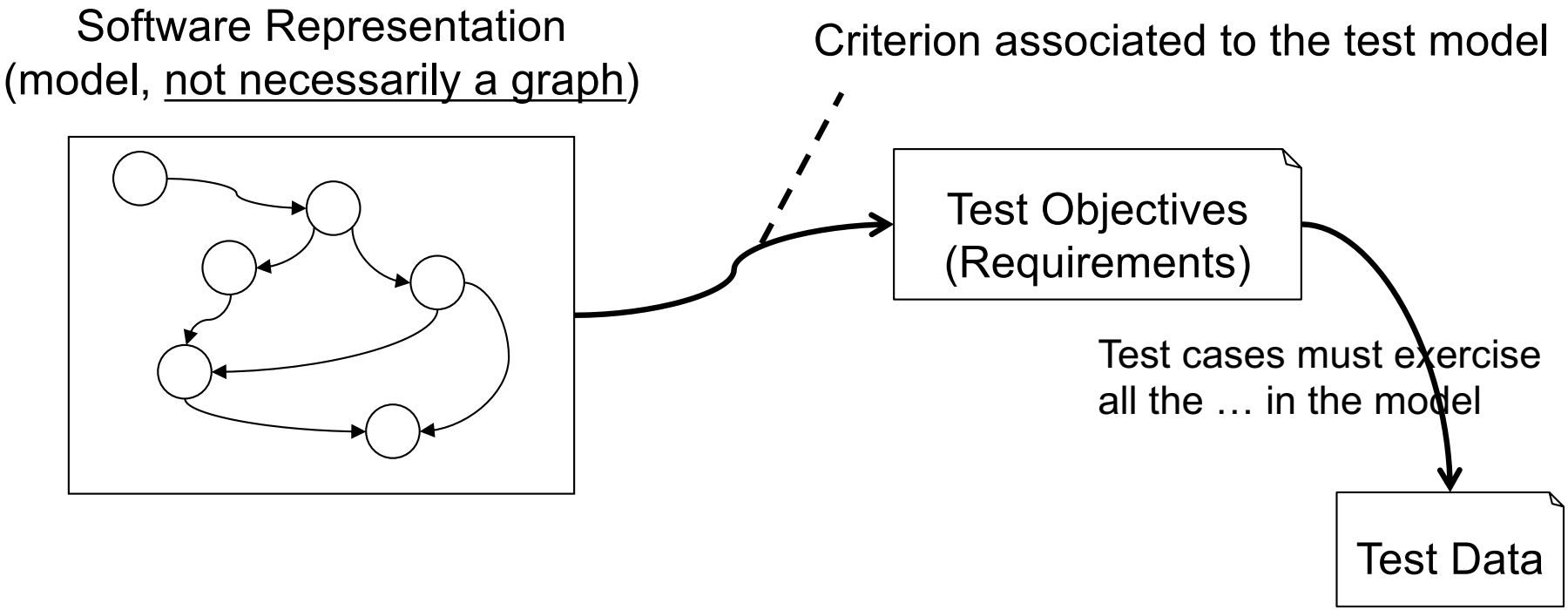
## Definitions—Part II

# *Exhaustive Testing*

- Exhaustive testing == testing using all possible inputs
  - Most of the time impossible !
- Examples:
  - A program that computes the factorial function (n!=n.(n-1).(n-2)…1)
    - Exhaustive testing = running the program with 0, 1, …, 100, …, i.e., <u>all possible integer values</u>!
  - A compiler (e.g., javac)
    - Exhaustive testing = compiling <u>every possible</u> (Java) program

➢ Technique used to reduce the number of inputs (i.e., test cases):
  - Testing criteria group input elements into (equivalence) classes
  - One input is selected in each class (notion of test data adequacy)
  - Criteria are used to decide which test inputs to use
  - Criteria are used to decide when to stop testing

# *Test Data: procedure to select?*




The test model is a representation of

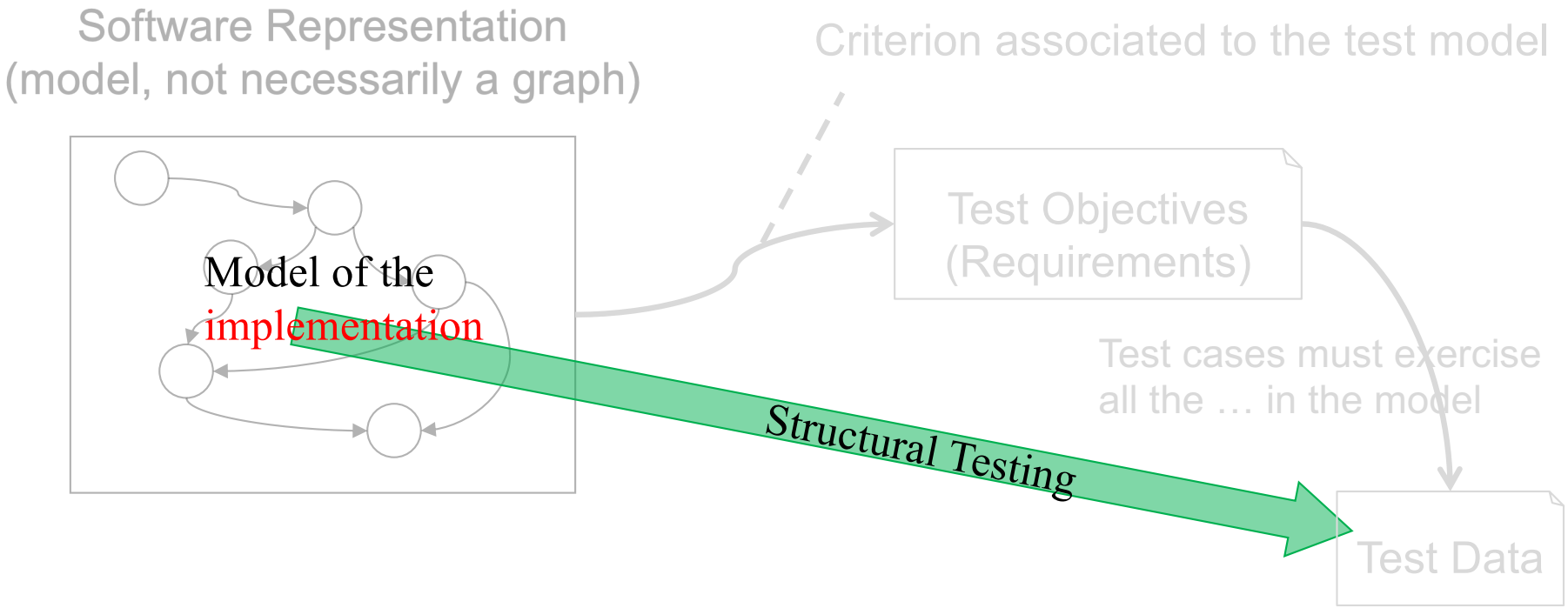
- the specification ⇒ Functional Testing
- the implementation ⇒ Structural Testing

Older terminology:

black-box testing

white-box testing

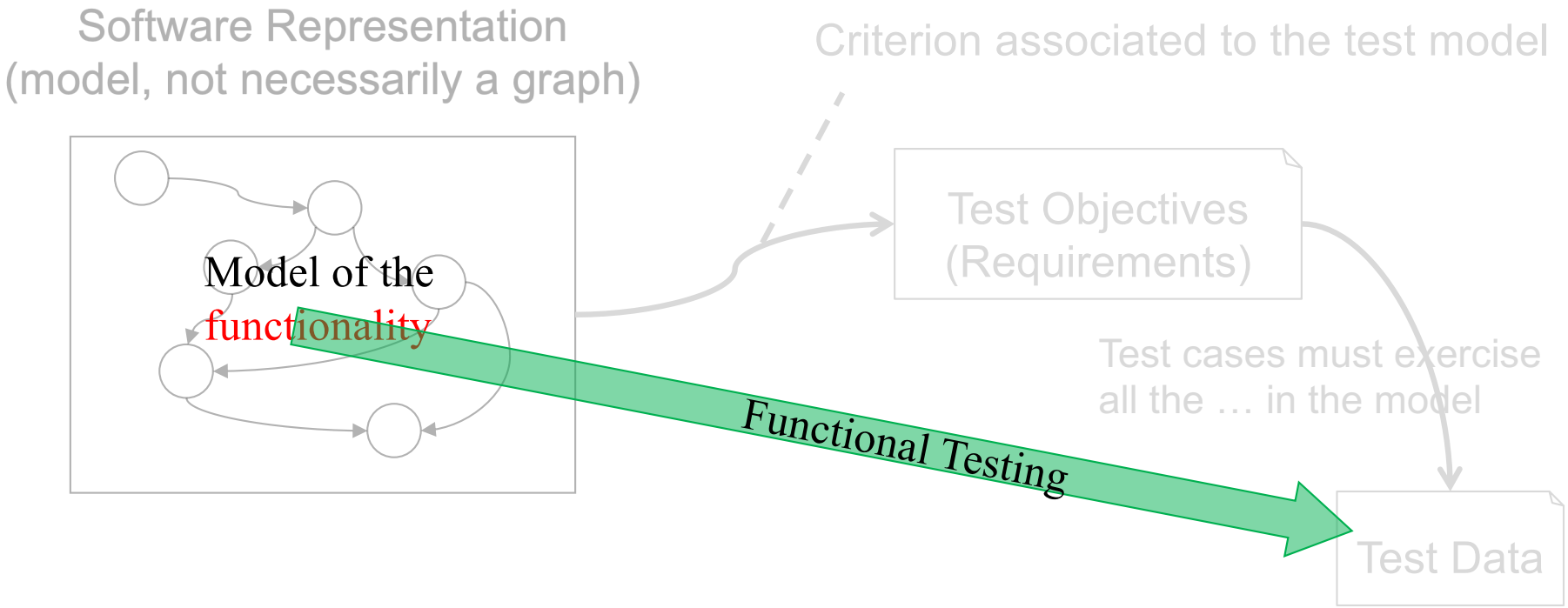
# *Test Data: procedure to select?*




The test model is a representation of

- the <u>specification</u> ⇒ Functional Testing
- the <u>implementation</u> ⇒ Structural Testing

# *Test Data: procedure to select?*




The test model is a representation of

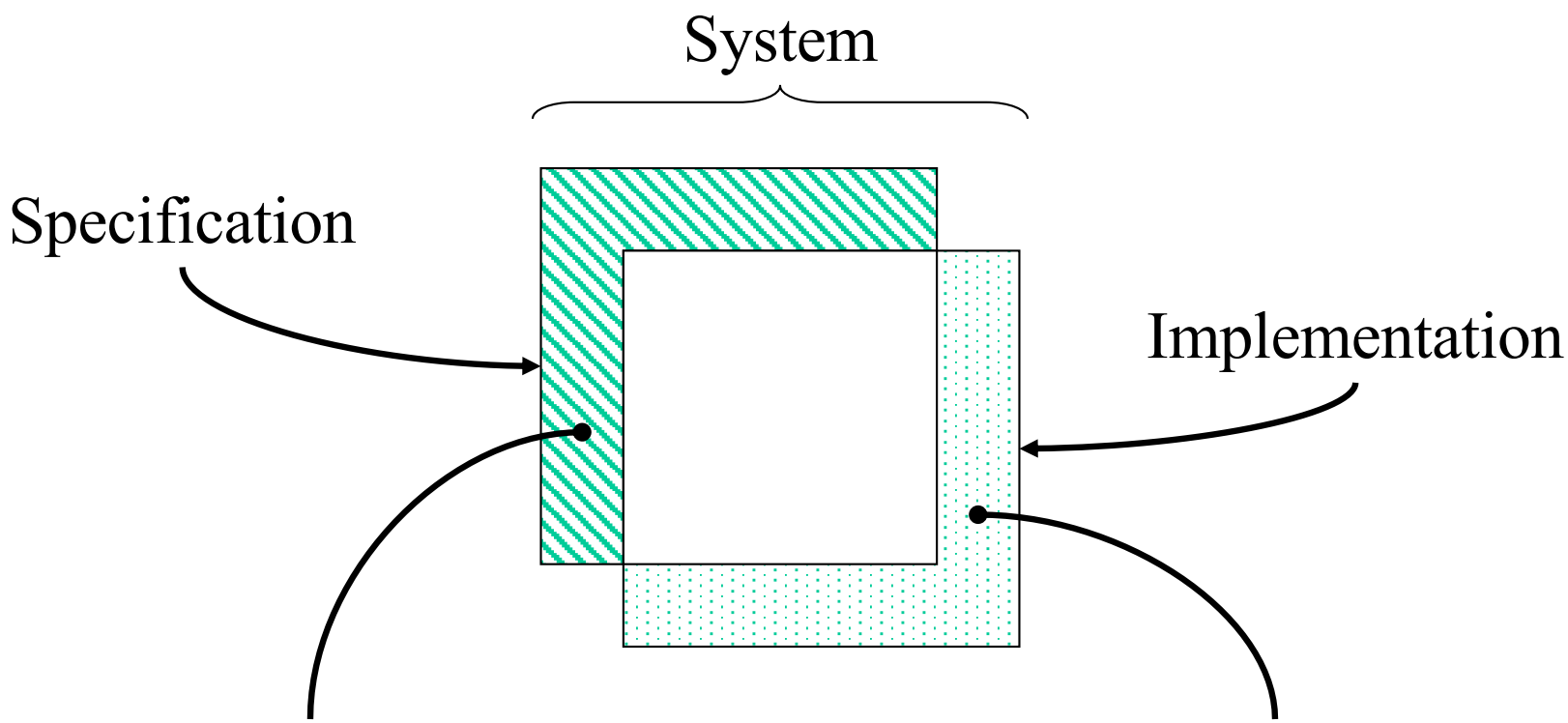
- the <u>specification</u> ⇒ Functional Testing
- the <u>implementation</u> ⇒ Structural Testing

# *Functional vs. Structural Testing*

- ☺ Check conformance with the specification
- ☺ It scales up (different techniques at different granularity levels)
  - ☺ Works for a function
  - ☺ Works for a class
  - ☺ Works for a package/component
  - ☺ Works for a system
- ☹ It depends on the specification and the degree of detail
- ☹ Do not know how much of the system (code) is being tested
  - ☹ What if the system performs some unexpected, undesirable task?

- ☺ Based on control and data flow criteria
- ☺ It allows you to be confident about how much of the system is being tested
- ☹ It does not scale up (mostly applicable at unit and integration testing levels)
  - ☹ Doesn't work for a package/component
  - ☹ Doesn't work for a system
- ☹ It cannot reveal missing functionalities
  - ☹ What if part of the specification is not implemented?

# *Functional vs. Structural Testing*




Missing functionality:
Cannot be revealed by white-box techniques

Unexpected functionality:
Cannot be revealed by black-box techniques

# *Test Model Criterion*

- Given a criterion C for a model M
  - The *coverage ratio* of a test set T is the proportion of the elements in M defined by C that are covered by T.
  - A test set T is said to be adequate for C, or simply C-adequate, when the coverage ratio achieves 100% for criterion C.
- Example 1:
  - M is the control flow graph of a function / C is “all the statements”
  - A test suite exercises 5 (out of 8) statements: 62,5% coverage (ratio)
    - Test suite is not adequate for the all-statements criterion
- Example 2:
  - M is a set of use case scenarios / C is “all the scenarios”
  - A test suite exercises 12 (out of 12) scenarios: 100% coverage (ratio)
    - Test suite is adequate for the all-scenarios criterion
    - Test suite is all-scenarios adequate

# *Test Model Criterion (cont.)*

- A test criterion
  - Specifies a set of test requirements/objectives
  - Test requirements must be satisfied in order to obtain an adequate test suite
- Issue!
  - When applying a criterion on a test model
  - Not all test requirements are feasible
- Revised notion of adequacy
  - The *coverage ratio* of a test set T is the proportion of the feasible elements in M defined by C covered by T.
  - A test set T is said to be adequate for C, or simply C-adequate, when the coverage ratio achieves 100% for criterion C.

# *Theoretical Hierarchy of criteria*

The subsumption relation between criteria for the same model M

- For a given model M: C1 subsumes C2 if any C1-adequate test set is also C2-adequate.

  Beware! This is not a subset relation!

  (The set of model elements to be exercised to satisfy C2 is not a subset of the set of model elements to be exercised to satisfy C1.)

- Example:
  - Consider criteria all-transitions and all-paths for finite state machines, all-paths subsumes all-transitions.
  - Any all-paths adequate test suite necessarily exercises all the transitions
- Usually (but not always), if C1 subsumes C2:
  - Satisfying C1 tends to be more expensive than satisfying C2 (e.g., C1 tends to require more test cases than C2)
  - A C1-adequate test suite tends to detect more faults than a C2-adequate test suite

# *Two ways to use Test Criteria*

- Generate test values / test cases to satisfy the criterion
  - Criterion = selection criterion
  - Need a tool (or human), a generator, that (automatically) generates values to satisfy the criterion.
  - Wish: create a (software) generator?
- Evaluate coverage achieved by externally generated test values / test cases
  - Criterion = coverage criterion
  - Need a tool (or human), a recognizer, that (automatically) decides whether a set of values satisfies a criterion.
  - Wish: create a (software) recognizer?

# *Two ways to use Test Criteria (cont.)*

- Problems:
  - How to create a generator?
  - How to create a recognizer?
- Issue!
  - Both problems are provably undecidable for most criteria
    - i.e., not possible to construct a single algorithm that will always, in every situation, find a correct solution.
  - However, it is often easier to build a recognizer than a generator
    - Coverage analysis tools (recognizer) are quite plentiful

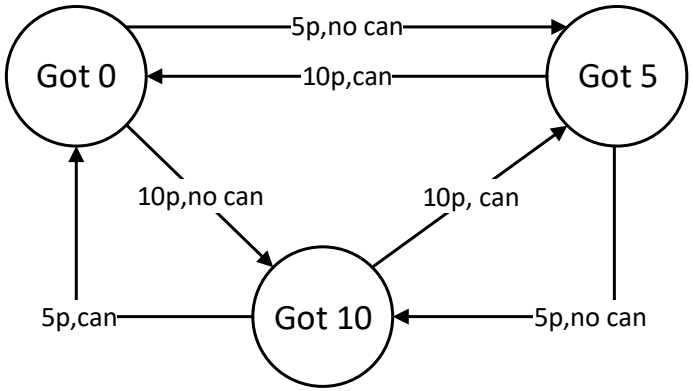
# *Miss-use of terminology*

- Beware of miss-use of term "coverage"
- Your colleague says:
  - "I am checking what my tests exercise with the all-statements selection criterion."
    - Wrong: they are using the all-statements criterion in a recognizer context
    - ➢ Coverage criterion
  - "I am creating tests with the all-scenarios coverage criterion."
    - Wrong: they are using the all-scenarios criterion in a generator context
    - ➢ Selection criterion
  - "I am doing structural testing since I check my tests execute all statements."
    - Wrong: With structural testing, one uses an selection criterion that applies on a model of the implementation (generator context); Here they are using the criterion in a recognizer context.
  - "My tests achieve 100% coverage"
    - What criterion? Different criteria may have extremely different costs!

# *Using a Test Selection Criterion*

| Steps | Example |
|---|---|
| 1. Choose a test model | 1. State machine |
| 2. Select a test criterion | 2. All-transitions |
| 3. Identify test objectives | 3. The transitions are t1, t2, … |
| 4. Create test case specifications | 4. Test case 1 will exercise transitions t1, t4, t5 …<br>Test case 2 will exercise transitions t1, t2, t8 …<br>… |
| 5. Identifying test data/input | 5. To execute test case 1, I need to execute with input value 10 …<br>To execute test case 2, I need to execute with input value 20 …<br>… |
| 6. Identify Oracle | 6. What do you feel you need to check during and at the end of the execution of test case 1, test case 2 … and what is it you should expect? |

# *Example (graph) Model*




- Specification of a vending machine
- Gets 5p or 10p of money
- Sells 15p cans

1. Choose a test model (see on the left)
2. Select a test criterion: all-transitions
3. Identify test objectives

   ~~Got0->Got5~~ ~~Got5->Got0~~ ~~Got0->Got10~~
   ~~Got10->Got0~~ ~~Goy10->Got5~~ ~~Got5->Got10~~
4. Create test case specification

   TCS1 = [Got0, Got5, Got10]

   TCS2 = [Got0, Got10, Got0]

   TCS3 = [Got0, Got10, Got5, Got0]
5. Identify test data

   TC1 = [5p, 5p]

   TC2 = [10p, 5p]

   TC3 = [10p, 10p, 10p]
6. Oracles

   TC1 = machine has 10p in, no can

   TC2 = machine has 0p in, one can

   TC3 = machine has 0p in, two cans

# *Marick's Recommandation*

Brian Marick recommends the following approach:

1. Generate functional tests from requirements and design to try every function.
   - Use a functional selection criterion (generator)
2. Check the structural coverage after the functional tests are all verified to be successful.
   - Use a structural coverage criterion (recognizer)
3. Where the structural coverage is imperfect, generate functional tests (not structural) that induce the additional coverage.

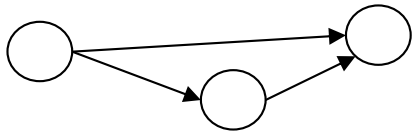
This works because form (structure) should follow function!

- Uncovered code must have some purpose, and that purpose has not been invoked, so some function is untested

# *Test Criteria Based on Structure [Offutt]*

- Graphs

Method body
Methods and calls
Components interactions
State and transitions
…



- Logical Expressions

(not X or not Y) and A and B

Can appear in:
- State machine
- Source code
- Software specification

- Input Domain Characterization

A: {0,1,>1}

B: {600,700,800}

C: {swe,cs,isa,infs}

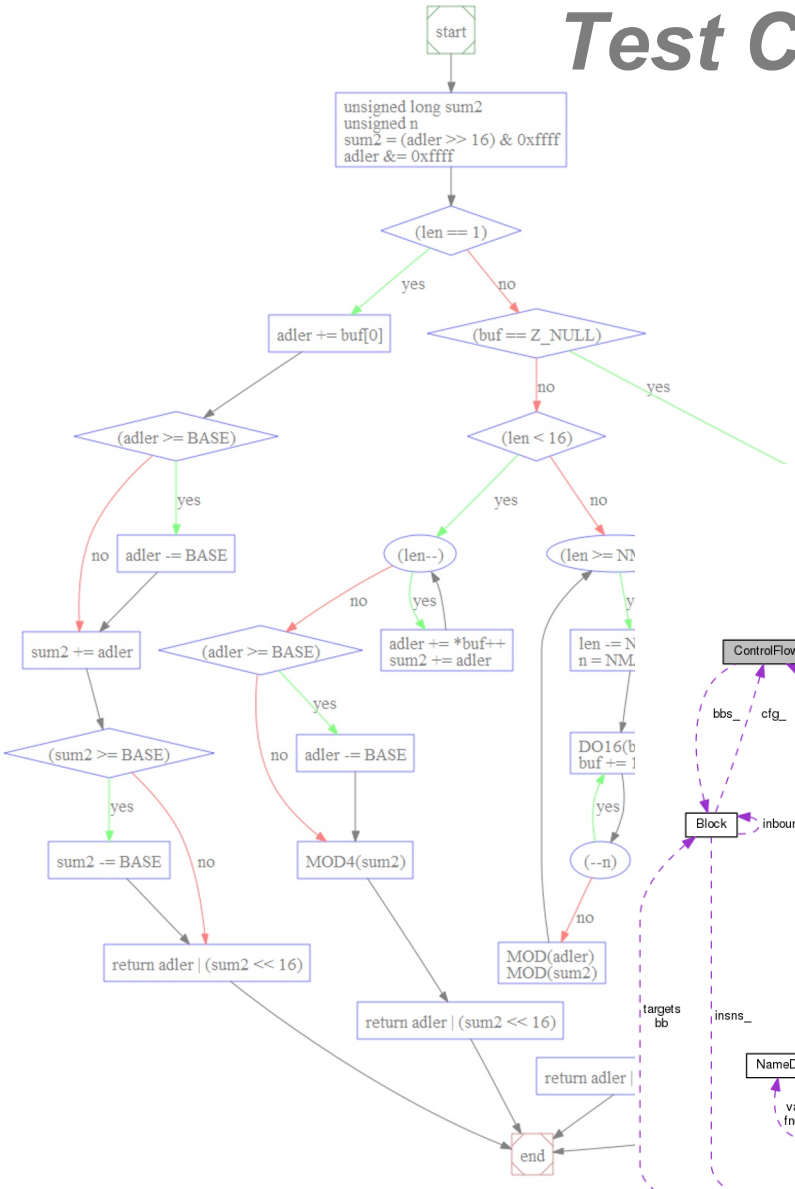
Describes the input domain of the software under test (method, component, system)

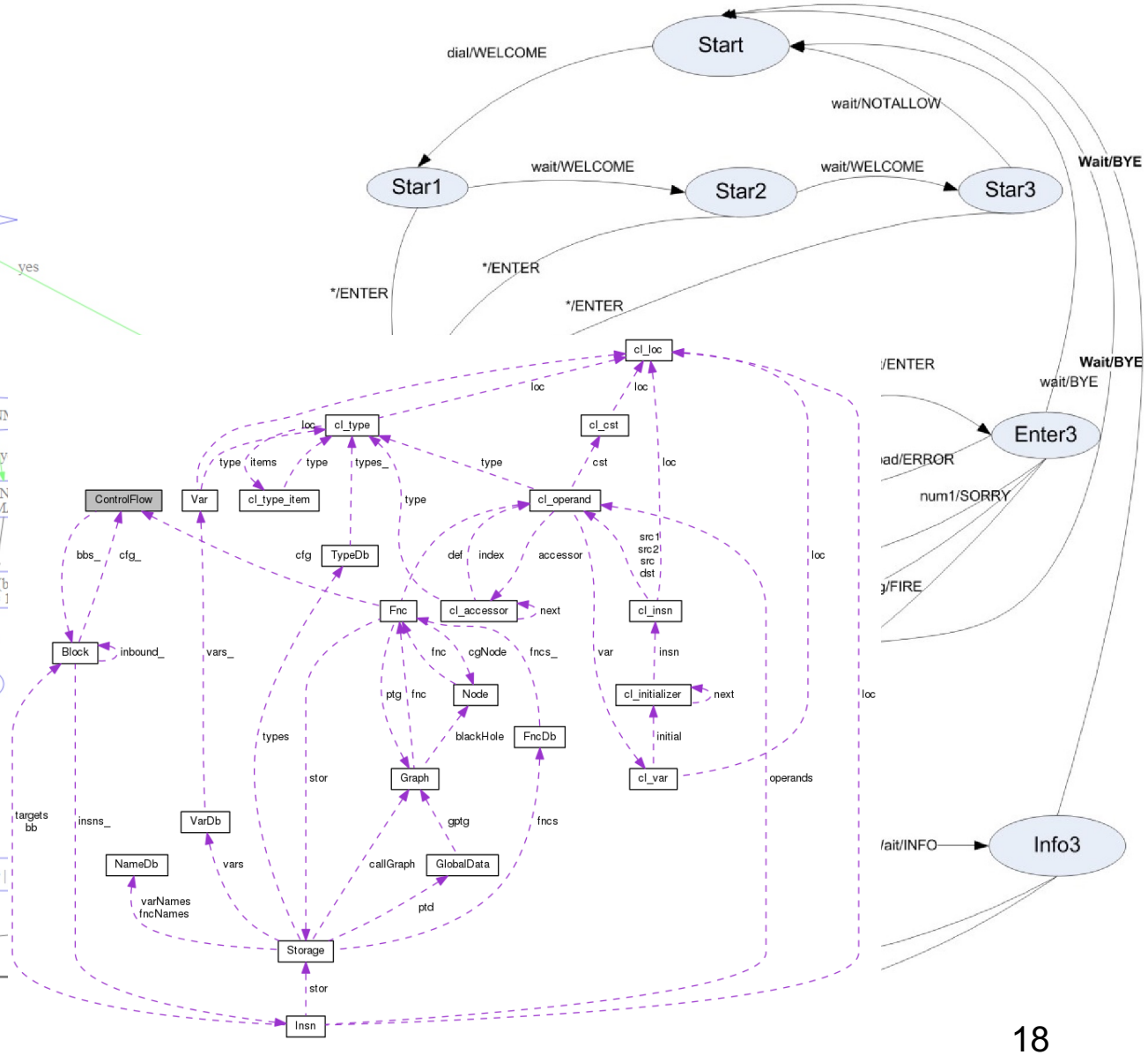
- Syntactic Structures

```
if (x>y)
      z = x - y;
else
      z = 2 * x
```

Based on a grammar, or other syntactic definition
- e.g., mutation testing

# Test Criteria—Graphs



# Test Criteria—Logic Expressions

```
if (specd_prefix != 0 && default_len != 0)
     for (p = include_defaults; p->fname; p++) {
         /* Some standard dirs are only for C++.  */
         if (!p->cplusplus || (cplusplus && !no standard cplusplus includes)) {
           /* Does this di
```

| t | $s_s \rightarrow s_e$ | Input | Gu |
|---|---|---|---|
| $t_0$ | $s_1 \rightarrow s_2$ | prop_opt | N |
| $t_1$ | $s_1 \rightarrow s_3$ | opt_ind, cr | N |
| $t_2$ | $s_2 \rightarrow s_4$ | opt_ind, cr | op |
| $t_3$ | $s_2 \rightarrow s_5$ | opt_ind, cr | op |
| $t_5$ | $s_3 \rightarrow s_4$ | accpt_opt | ac |
| $t_7$ | $s_4 \rightarrow s_4$ | Udata | S_ |
| $t_8$ | $s_4 \rightarrow s_4$ | Send_sq | R_ |
| $t_9$ | $s_4 \rightarrow s_4$ | Send_sq | R_ |
| $t_{10}$ | $s_4 \rightarrow s_4$ | cr | N |
| $t_{11}$ | $s_4 \rightarrow s_4$ | XpSsq, cr | T 0 |
| $t_{12}$ | $s_4 \rightarrow s_4$ | XpSsq, cr | T 0 |
| $t_{13}$ | $s_4 \rightarrow s_4$ | XpSsq, cr | T 12 |
| $t_{14}$ | $s_4 \rightarrow s_4$ | XpSsq, cr | T 12 |
| $t_{15}$ | $s_4 \rightarrow s_4$ | – | S_ |

1. $\overline{(ab)}(d\bar{e}\bar{f} + \bar{d}e\bar{f} + \bar{d}\bar{e}\bar{f})(ac(d+e)h + a(d+e)\bar{h} + b(e+f))$
2. $(a((c+d+e)g + af + c(f+g+h+i)) + (a+b)(c+d+e)i)\ \overline{(ab)}\ \overline{(cd)}\ \overline{(ce)}\ \overline{(de)}\ \overline{(fg)}\ \overline{(fh)}\ \overline{(fi)}\ \overline{(gh)}\ \overline{(hi)}$
3. $(a(\bar{d} + \bar{e} + de\overline{(\bar{f}gh\bar{i} + \bar{g}hi)}\ \overline{(\bar{f}glk + \bar{g}\bar{i}k)}) + \overline{(\bar{f}gh\bar{i} + \bar{g}hi)}\ \overline{(\bar{f}glk + \bar{g}\bar{i}k)}(b + c\bar{m} + f))\ (a\bar{b}\bar{c} + \bar{a}b\bar{c} + \bar{a}\bar{b}c)$
4. $a(\bar{b} + \bar{c})d + e$
5. $a(\bar{b} + \bar{c} + bc\overline{(\bar{f}gh\bar{i} + \bar{g}hi)}\ \overline{(\bar{f}glk + \bar{g}\bar{i}k)}) + f$
6. $(\bar{a}b + a\bar{b})\overline{(cd)}(f\bar{g}\bar{h} + \bar{f}g\bar{h} + \bar{f}\bar{g}\bar{h})\overline{(jk)}((ac+bd)e(f + (i(gj+hk))))$
7. $(\bar{a}b + a\bar{b})\overline{(cd)}\ \overline{(gh)}\ \overline{(jk)}((ac+bd)e(\bar{i} + \bar{g}\bar{k} + \bar{j}(\bar{h} + \bar{k})))$
8. $(\bar{a}b + a\bar{b})\overline{(cd)}\ \overline{(gh)}((ac+bd)e(fg + \bar{f}h))$
9. $\overline{(cd)}(\bar{e}f\bar{g}\bar{a}(bc + \bar{b}d))$
10. $a\bar{b}\bar{c}d\bar{e}f(g + \bar{g}(h+i))\overline{(jk + \bar{j}l + m)}$
11. $a\bar{b}\bar{c}(\overline{(f(g + \bar{g}(h+i)))} + f(g + \bar{g}(h+i))\bar{d}\bar{e})\overline{(jk + \bar{j}l\bar{m})}$
12. $a\bar{b}\bar{c}(f(g + \bar{g}(h+i))(\bar{e}\bar{n} + d) + \bar{n}(jk + \bar{j}l\bar{m})$
13. $a + b + c + \bar{c}\bar{d}ef\bar{g}\bar{h} + i(j+k)\bar{l}$
14. $ac(d+e)h + a(d+e)\bar{h} + b(e+f)$
15. $a((c+d+e)g + af + c(f+g+h+i)) + (a+b)(c+d+e)i$
16. $a(\bar{d} + \bar{e} + de\overline{(\bar{f}gh\bar{i} + \bar{g}hi)}\ \overline{(\bar{f}glk + \bar{g}\bar{i}k)}) + \overline{(\bar{f}gh\bar{i} + \bar{g}hi)}\ \overline{(\bar{f}glk + \bar{g}\bar{i}k)}(b + c\bar{m} + f)$
17. $(ac+bd)e(f + (i(gj+hk)))$
18. $(ac+bd)e(\bar{i} + \bar{g}\bar{k} + \bar{j}(\bar{h} + \bar{k}))$
19. $(ac+bd)e(fg + \bar{f}h)$
20. $\bar{e}f\bar{g}\bar{a}(bc + \bar{b}d)$

# *Test Criteria—Input Domain Characterization*

The **grep** utility searches any given input files, selecting lines that match one or more patterns. By default, a pattern matches an input line if the regular expression (RE) in the pattern matches the input line without its trailing newline. An emp matches at least

A general **ATM** (Automatic Teller Machine) system is implemented as Web service and deployed in the Tomcat server. The user and business data are stored in a MySQL database. The system offers several features, such as withdrawal, deposit, transfer, query, and each of them

**TCAS** is a family of airborne devices that function independently of the ground-based air traffic control (ATC) system, and provide collision avoidance protection for a broad spectrum of aircraft types. All TCAS systems provide some degree of collision threat alerting, and a traffic display. **TCAS I and II** differ primarily by their alerting capability. TCAS I provides traffic advisories (TAs) to assist the pilot in the visual acquisition of intruder aircraft. TCAS I is mandated for use in the U.S. for turbine powered, passenger-carrying aircraft having more than 10 and less than 31 seats. TCAS I is also installed on a number of general aviation fixed wing aircraft and helicopters. TCAS II provides TAs and resolution advisories (RAs), i.e., recommended escape maneuvers, in the vertical dimension to either increase or maintain the existing vertical separation between aircraft. TCAS II is mandated by the U.S. for commercial aircraft, including regional airline aircraft with more than 30 seats or a maximum takeoff weight greater than 33,000 lbs. Although not mandated for general aviation use, many turbine-powered general aviation aircraft and some helicopters are also equipped with TCAS II.

… 50 pages long specification document.

# Test Criteria—Syntactic Structure

```
  if (alignment == 0)
    alignment = DEFAULT_ALIGNMENT;
  if (size == 0)
    /* Default size is what GNU malloc can fit in a 4096-byte block.  */
    {
      /* 12 is sizeof (mhead) and 4 is EXTRA from GNU malloc.
                                Use the values for range checking, because if range checking is off,
                                the extra bytes won't be missed terribly, but if range checking is on
                                and we used a larger request, a whole extra 4096 bytes would be
                                allocated.

                                These number are irrelevant to the new GNU malloc.  I suspect it is
                                less sensitive to the size of the request.  */
      int extra = ((((12 + DEFAULT_ROUNDING - 1) & ~(DEFAULT_ROUNDING - 1))
                                                  + 4 + DEFAULT_ROUNDING - 1)
                                                  & ~(DEFAULT_ROUNDING - 1));

      size = 4096 - extra;
    }

  h->chunkfun = (struct _obstack_chunk * (*)()) chunkfun;
  h->freefun = freefun;
  h->chunk_size = size;
  h->alignment_mask = alignment - 1;
  h->extra_arg = arg;
  h->use_extra_arg = 1;

  chunk = h->chunk = CALL_CHUNKFUN (h, h -> chunk_size);
  if (!chunk)
    {
      h->alloc_failed = 1;
      return 0;
    }
  h->alloc_failed = 0;
  h->next_free = h->object_base = chunk->contents;
  h->chunk_limit = chunk->limit
    = (char *) chunk + h->chunk_size;
  chunk->prev = 0;
  /* The initial chunk now contains no empty object.  */
  h->maybe_empty_object = 0;
  return 1;
}

/* Allocate a new current chunk for the obstack *H
   on the assumption that LENGTH bytes need to be added
   to the current object, or a new object of length LENGTH allocated.
   Copies any partial object from the end of the old chunk
   to the beginning of the new one.  */

void
_obstack_newchunk (h, length)
     struct obstack *h;
     int length;
{
  register struct _obstack_chunk*                          old_chunk = h->chunk;
  register struct _obstack_chunk*                          new_chunk;
  register long               new_size;
  register int obj_size = h->next_free - h->object_base;
  register int i;
  int already;

  /* Compute size for new chunk.  */
  new_size = (obj_size + length) + (obj_size >> 3) + 100;
  if (new_size < h->chunk_size)
    new_size = h->chunk_size;

  /* Allocate and initialize the new chunk.  */
  new_chunk = CALL_CHUNKFUN (h, new_size);
  if (!new_chunk)
    {
      h->alloc_failed = 1;
      return;
    }
  h->alloc_failed = 0;
  h->chunk = new_chunk;
  new_chunk->prev = old_chunk;
  new_chunk->limit = h->chunk_limit = (char *) new_chunk + new_size;

  /* Move the existing object to the new chunk.
     Word at a time is fast and is safe if the object
     is sufficiently aligned.  */
  if (h->alignment_mask + 1 >= DEFAULT_ALIGNMENT)
    {
      for (i = obj_size / sizeof (COPYING_UNIT) - 1;
                                i >= 0; i--)
                              ((COPYING_UNIT *)new_chunk->contents)[i]
                                = ((COPYING_UNIT *)h->object_base)[i];
      /* We used to copy the odd few remaining bytes as one extra COPYING_UNIT,
                               but that can cross a page boundary on a machine
                               which does not do strict alignment for COPYING_UNITS.  */
      already = obj_size / sizeof (COPYING_UNIT) * sizeof (COPYING_UNIT);
    }
  else
    already = 0;
  /* Copy remaining bytes one by one.  */
  for (i = already; i < obj_size; i++)
    new_chunk->contents[i] = h->object_base[i];

  /* If the object just copied was the only data in OLD_CHUNK,
     free that chunk and remove it from the chain.
     But not if that chunk might contain an empty object.  */
  if (h->object_base == old_chunk->contents && ! h->maybe_empty_object)
    {
      new_chunk->prev = old_chunk->prev;
      CALL_FREEFUN (h, old_chunk);
    }

  h->object_base = new_chunk->contents;
  h->next_free = h->object_base + obj_size;
  /* The new chunk certainly contains no empty object yet.  */
  h->maybe_empty_object = 0;
}
```

```
   * clean up to avoid a sublte but serious memory leak when using the
   * same config for multiple JPF objects/runs - this listener is an inner
   * class object that keeps its encapsulating JPF instance alive
   */
  @Override
  public void jpfRunTerminated(Config config){
    config.removeChangeListener(this);
  }
}

/** this is the backbone of all JPF configuration */
Config config;

/** The search policy used to explore the state space */
Search search;

/** interface to the virtual machine used by the search */
VM vm;

/** the report generator */
Reporter reporter;

Status status = Status.NEW;

/** a list of listeners that get automatically added from VM, Search or Reporter initialization */
List<VMListener> pendingVMListeners;
List<SearchListener> pendingSearchListeners;


/** we use this as safety margin, to be released upon OutOfMemoryErrors */
byte[] memoryReserve;

private static Logger initLogging(Config conf) {
  LogManager.init(conf);
  return getLogger("gov.nasa.jpf");
}

/**
 * use this one to get a Logger that is initialized via our Config mechanism. Note that
 * our own Loggers do NOT pass
 */
public static JPFLogger getLogger (String name) {
  return LogManager.getLogger( name);
}

public static void main(String[] args){
  int options = RunJPF.getOptions(args);

  if (args.length == 0 || RunJPF.isOptionEnabled( RunJPF.HELP,options)) {
    RunJPF.showUsage();
    return;
  }
  if (RunJPF.isOptionEnabled( RunJPF.ADD_PROJECT,options)){
    RunJPF.addProject(args);
    return;
  }

  if (RunJPF.isOptionEnabled( RunJPF.BUILD_INFO,options)){
    RunJPF.showBuild(RunJPF.class.getClassLoader());
  }

  if (RunJPF.isOptionEnabled( RunJPF.LOG,options)){
    Config.enableLogging(true);
  }

  Config conf = createConfig(args);

  if (RunJPF.isOptionEnabled( RunJPF.SHOW, options)) {
    conf.printEntries();
  }

  start(conf, args);
}

public static void start(Config conf, String[] args){
  // this is redundant to jpf.report.<publisher>.start=..config..
  // but nobody can remember this (it's only used to produce complete reports)

  if (logger == null) {
    logger = initLogging(conf);
  }

  if (!checkArgs(args)){
    return;
  }

  setNativeClassPath(conf); // in case we have to find a shell

  // check if there is a shell class specification, in which case we just delegate
  JPFShell shell = conf.getInstance("shell", JPFShell.class);
  if (shell != null) {
    shell.start(args); // responsible for exception handling itself

  } else {
    // no shell, we start JPF directly
    LogManager.printStatus(logger);
    conf.printStatus(logger);

    // this has to be done after we checked&consumed all "-.." arguments
    // this is NOT about checking properties!
    checkUnknownArgs(args);

    try {
      JPF jpf = new JPF(conf);
      jpf.run();

    } catch (ExitException x) {
      logger.severe( "JPF terminated");

      // this is how we get most runtime config exceptions
      if (x.shouldReport()) {
        x.printStackTrace();
```

```
overriding function Create_File
  (This : in out SHFS;
  Path : String)
  return Status_Code
is
  pragma Unreferenced (This, Path);
begin
  return Read_Only_File_System;
end Create_File;

-----------------
-- Create_Node --
-----------------

function Create_Node
  (This : in out SHFS;
  Path : String;
  Kind : File_Kind)
  return Status_Code
is
  pragma Unreferenced (This, Path, Kind);
begin
  return Read_Only_File_System;
end Create_Node;

----------------------
-- Create_Directory --
----------------------

function Create_Directory
  (This : in out SHFS;
  Path : String)
  return Status_Code
is
  pragma Unreferenced (This, Path);
begin
  return Read_Only_File_System;
end Create_Directory;

------------
-- Unlink --
------------

overriding function Unlink
  (This : in out SHFS;
  Path : String)
  return Status_Code
is
  pragma Unreferenced (This, Path);
begin
  return Read_Only_File_System;
end Unlink;

----------------------
-- Remove_Directory --
----------------------

overriding function Remove_Directory
  (This : in out SHFS;
  Path : String)
  return Status_Code
is
  pragma Unreferenced (This, Path);
begin
  return Read_Only_File_System;
end Remove_Directory;

------------
-- Rename --
------------

function Rename
  (This : in out SHFS;
  Old_Path : String;
  New_Path : String)
  return Status_Code
is
  pragma Unreferenced (This, Old_Path, New_Path);
begin
  return Read_Only_File_System;
end Rename;

-----------------------
-- Change_Permissions --
-----------------------

function Truncate_File
  (This   : in out SHFS;
  Path   : String;
  Length : File_Size)
  return Status_Code
is
  pragma Unreferenced (Path, Length, This);
begin
  return Read_Only_File_System;
end Truncate_File;

----------
-- Open --
----------

overriding function Open
  (This   : in out SHFS;
  Path   : String;
  Mode   : File_Mode;
  Handler : out Any_File_Handle)
  return Status_Code
is
  FH : SHFS_File_Handle_Access;
  FD : SH_Word;
begin
  if Path'Length = 0 then
```