Complete System Testing

Sanath Kumar Ramesh

CSE 218

May 25, 2011

Today's Papers

1. **Automatic system testing of programs without test oracles**, Christian Murphy, Kuang Shen, and Gail Kaiser.
2. **Automating System Tests Using Declarative Virtual Machines**, van der Burg, S.; Dolstra, E.
3. **A formal analysis of requirements-based testing**, Charles Pecheur, Franco Raimondi, and Guillaume Brat.

Automatic System Testing of Programs without Test Oracles

Christian Murphy, Kuang Shen, Gail Kaiser
Columbia University

Problem Landscape

Problem Landscape

Metamorphic Testing -

Problem Landscape

Metamorphic Testing - System testing where test oracles are not applicable..

Problem Landscape

Metamorphic Testing - System testing where test oracles are not applicable..
What are test oracles? -

Problem Landscape

Metamorphic Testing - System testing where test oracles are not applicable..
What are test oracles? - An oracle is a mechanism for determining whether the program has passed or failed a test.

Problem Landscape

Metamorphic Testing - System testing where test oracles are not applicable..
What are test oracles? - An oracle is a mechanism for determining whether the program has passed or failed a test.
When are test oracles not applicable?

Problem Landscape

Metamorphic Testing - System testing where test oracles are not applicable..

What are test oracles? - An oracle is a mechanism for determining whether the program has passed or failed a test.

When are test oracles not applicable?

- Input, Output cannot be clearly defined..

Problem Landscape

Metamorphic Testing - System testing where test oracles are not applicable..

What are test oracles? - An oracle is a mechanism for determining whether the program has passed or failed a test.

When are test oracles not applicable?

- Input, Output cannot be clearly defined.. Example:
    - What is the input to an airplane autopilot system??

Problem Landscape

Metamorphic Testing - System testing where test oracles are not applicable..

What are test oracles? - An oracle is a mechanism for determining whether the program has passed or failed a test.

When are test oracles not applicable?

- Input, Output cannot be clearly defined.. Example:
  - What is the input to an airplane autopilot system??
  - What is its output??

Problem Landscape

Metamorphic Testing - System testing where test oracles are not applicable..

What are test oracles? - An oracle is a mechanism for determining whether the program has passed or failed a test.

When are test oracles not applicable?

- Input, Output cannot be clearly defined.. Example:
  - What is the input to an airplane autopilot system??
  - What is its output??
  - A Simpler example..

Problem Landscape

Metamorphic Testing - System testing where test oracles are not applicable..

What are test oracles? - An oracle is a mechanism for determining whether the program has passed or failed a test.

When are test oracles not applicable?

- Input, Output cannot be clearly defined.. Example:
  - What is the input to an airplane autopilot system??
  - What is its output??
  - A Simpler example..Input to a program calculating PI

Problem Landscape

Metamorphic Testing - System testing where test oracles are not applicable..

What are test oracles? - An oracle is a mechanism for determining whether the program has passed or failed a test.

When are test oracles not applicable?

- Input, Output cannot be clearly defined.. Example:
  - What is the input to an airplane autopilot system??
  - What is its output??
  - A Simpler example..Input to a program calculating PI
  - Output?????

Problem Landscape

Metamorphic Testing - System testing where test oracles are not applicable..
What are test oracles? - An oracle is a mechanism for determining whether the program has passed or failed a test.
When are test oracles not applicable?

- Input, Output cannot be clearly defined.. Example:
  - What is the input to an airplane autopilot system??
  - What is its output??
  - A Simpler example..Input to a program calculating PI
  - Output?????
- Generally, Scientific calculations, optimizations, machine learning etc fall in this category

Metamorphic Testing - How is it done?

- Simple inputs and outputs to the system are identified

Metamorphic Testing - How is it done?

- Simple inputs and outputs to the system are identified
- System is first tested with them

Metamorphic Testing - How is it done?

- Simple inputs and outputs to the system are identified
- System is first tested with them
- Modify existing test case input to produce new test case such that new output can be predicted from the existing output if x is the old input, $f(x)$ is old output, then create new input x' from x such that $f(x')$ can be predicted from $f(x)$

But..

- Very hard to manually enerate new inputs..

But..

- Very hard to manually enerate new inputs..
- Even harder to validate the new output

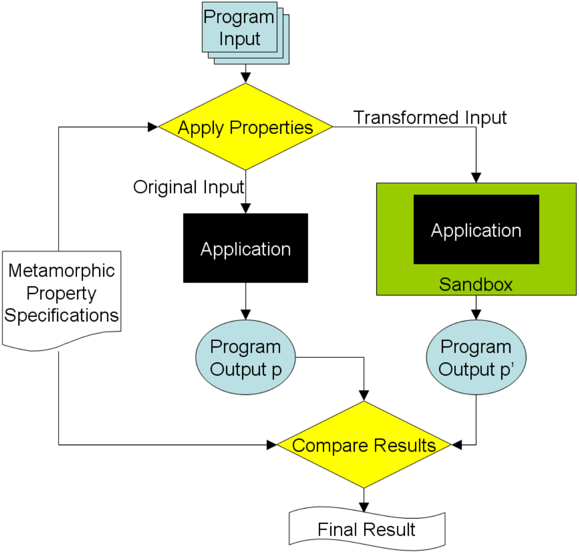
But..

- Very hard to manually enerate new inputs..
- Even harder to validate the new output
- They are even called “Non-Testable” programs
- Alternative??

But..

- Very hard to manually enerate new inputs..
- Even harder to validate the new output
- They are even called "Non-Testable" programs
- Alternative??
  - * **Pseudo-Oracle**
    Make multiple implementations of same program with different algorithms and validate output..!!
  - * **Automated Metamorphic System Testing(Paper's proposal)**
    Automate the process of creating various inputs, running the test and validating the output

Automatic Metamorphic System Testing Framework



Framework Description

Input Transformations

- XML file specifies the transformation type

Framework Description

Input Transformations

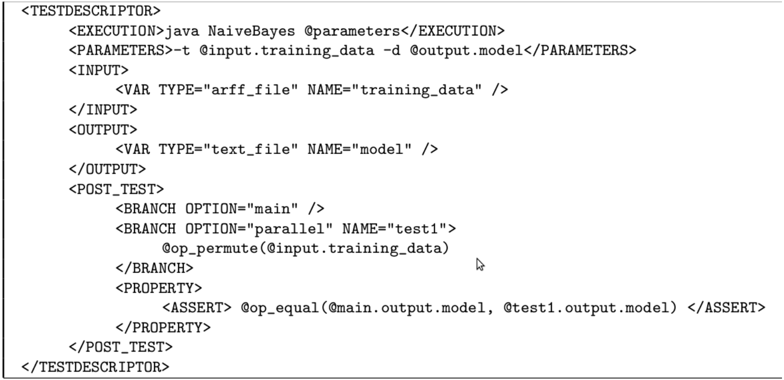
- XML file specifies the transformation type

  6 transformations are implemented:

  1. Adding a constant to numerical values
  2. Multiplying numerical values by a constant
  3. Permuting the order of input data
  4. Reversing the order of input data
  5. Removing part of data
  6. Adding additional data

Framework Description

Metamorphic Property Specification XML



```
<TESTDESCRIPTOR>
      <EXECUTION>java NaiveBayes @parameters</EXECUTION>
      <PARAMETERS>-t @input.training_data -d @output.model</PARAMETERS>
      <INPUT>
            <VAR TYPE="arff_file" NAME="training_data" />
      </INPUT>
      <OUTPUT>
            <VAR TYPE="text_file" NAME="model" />
      </OUTPUT>
      <POST_TEST>
            <BRANCH OPTION="main" />
            <BRANCH OPTION="parallel" NAME="test1">
                  @op_permute(@input.training_data)
            </BRANCH>
            <PROPERTY>
                  <ASSERT> @op_equal(@main.output.model, @test1.output.model) </ASSERT>
            </PROPERTY>
      </POST_TEST>
</TESTDESCRIPTOR>
```

Framework Description

Program Execution

Framework Description

Program Execution

- Copy all files needed for application execution

Framework Description

Program Execution

- Copy all files needed for application execution
- Initialize application inside a sandbox - Amsterdam + “pod” (virtualization layer)

Framework Description

Program Execution

- Copy all files needed for application execution
- Initialize application inside a sandbox - Amsterdam + “pod” (virtualization layer)
- Execute either in

Framework Description

Program Execution

- Copy all files needed for application execution
- Initialize application inside a sandbox - Amsterdam + “pod” (virtualization layer)
- Execute either in
  - Production Environment - Run by end-user after deployment
    - Invoke sanboxed application and “functional” app parallely

Framework Description

Program Execution

- Copy all files needed for application execution
- Initialize application inside a sandbox - Amsterdam + “pod” (virtualization layer)
- Execute either in
  - Production Environment - Run by end-user after deployment
    - Invoke sanboxed application and “functional” app parallely
  - Development Environment - Pre-release testing
    - Can disable parallel execution & Sandboxing
    - Trace dump enable

Framework Description

Program Execution

- Copy all files needed for application execution
- Initialize application inside a sandbox - Amsterdam + “pod” (virtualization layer)
- Execute either in
  - Production Environment - Run by end-user after deployment
    - Invoke sanboxed application and “functional” app parallely
  - Development Environment - Pre-release testing
    - Can disable parallel execution & Sandboxing
    - Trace dump enable
- If test fails, Pop-up message or write to file

Framework Description

Output Comparison

Output Match $\implies$ No fault in program

Output Mismatch $\implies$ Fault Detected !!

Framework Description

Output Comparison

Output Match $\implies$ No fault in program

Output Mismatch $\implies$ Fault Detected !!

- Exact mismatch

Framework Description

Output Comparison

Output Match $\Longrightarrow$ No fault in program

Output Mismatch $\Longrightarrow$ Fault Detected !!

- * Exact mismatch
- * Approximate Mismatch -
  Ex: Floating Point results
  Value of $sine(x)$ and $sine(x + 2\pi)$ should be same. But the precision depends on value of π used in the program.

Framework Description

Output Comparison

Output Match $\implies$ No fault in program

Output Mismatch $\implies$ Fault Detected !!

- Exact mismatch
- Approximate Mismatch -
  Ex: Floating Point results
  Value of $sine(x)$ and $sine(x + 2\pi)$ should be same. But the precision depends on value of π used in the program.

  Use *Heuristic Metamorphic Testing* - Upto 2% mismatch allowed

Empirical Sutdies

Evaluated on 3 machine learning applications from Weka 3.5.8 toolkit (Java)

- **Support Vector Machines** - Supervised classification
- **C4.5** - Decision tree builder
- **MartiRank** - Ranking algorithm
- **PAYL** - Network packets anomoly based intrusion detection system

Empirical Sutdies

Experiment Methodoly

- Insert random mutation in source code
- Determine if mutation can be detected by the testing suite

Three mutant types used:

- Flip comparison operators: $=$ becomes $\neq$
- Flip mathematical operators: $*$ become $\div$
- Off-by-one error: adjust loop variable, array indices etc by one

Results

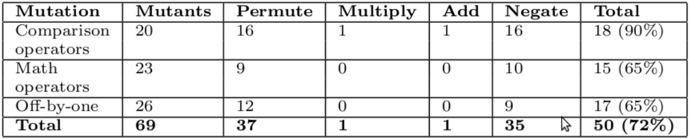
Mutation	Mutants	Permute	Multiply	Add	Negate	Total
Comparison operators	30	17	2	0	0	17 (57%)
Math operators	24	13	0	11	16	18 (75%)
Off-by-one	31	27	0	7	9	31 (100%)
Total	**85**	**57**	**2**	**18**	**25**	**66 (77%)**

Table 1: Results of Mutation Testing for SVM

Mutation	Mutants	Permute	Multiply	Add	Negate	Total
Comparison operators	8	8	0	1	7	8 (100%)
Math operators	15	2	3	1	13	14 (93%)
Off-by-one	5	2	0	0	5	5 (100%)
Total	**28**	**12**	**3**	**2**	**25**	**27 (96%)**

Table 2: Results of Mutation Testing for C4.5

Results



Mutation	Mutants	Permute	Multiply	Add	Negate	Total
Comparison operators	20	16	1	1	16	18 (90%)
Math operators	23	9	0	0	10	15 (65%)
Off-by-one	26	12	0	0	9	17 (65%)
Total	**69**	**37**	**1**	**1**	**35**	**50 (72%)**

Table 3: Results of Mutation Testing for MartiRank

For PAYL:

- Detected 2 of 40 mutants !!

- Reason: PAYL bothers about distribution of data. Input metamorphosis altered content and ordering only, not distribution.

Performance

Quad-core 3GHz CPU running Ubuntu
- 400ms lag in *application startup* for 10MB input file
- “Functional Application” and “Sanboxed Application” ran on different cores without measurable lag to user

Snake Oil

- Can't work with Databases and Network
- Can't work for applications requiring user response
- Fault Localization not possible
- Can't work for binary input/output
- No false +ve rate mentioned in paper

Automating System Tests Using Declarative Virtual Machines

Sander van der Burg, Eelco Dolstra
Delft University of Technology, The Netherlands

Problem Landscape

Problem Landscape

- To run regression suite on entire application

Problem Landscape

- To run regression suite on entire application
- Heavy dependency between modules

Problem Landscape

- To run regression suite on entire application
- Heavy dependency between modules
- Application requires specific *system environment*
  For example:

Problem Landscape

- To run regression suite on entire application
- Heavy dependency between modules
- Application requires specific *system environment*
  For example:
  - **OpenSSH** requires super-user privileges to run; needs multiple user accounts

Problem Landscape

- To run regression suite on entire application
- Heavy dependency between modules
- Application requires specific *system environment*
  For example:
  - **OpenSSH** requires super-user privileges to run; needs multiple user accounts
  - **Quake 3 Arena** needs networked machines to start a client and server

Problem Landscape

- To run regression suite on entire application
- Heavy dependency between modules
- Application requires specific *system environment*
  For example:
  - **OpenSSH** requires super-user privileges to run; needs multiple user accounts
  - **Quake 3 Arena** needs networked machines to start a client and server
  - **Transmission** requires a host behind NAT with UPnP-IGD protocol enabled router

Problem Landscape

- To run regression suite on entire application
- Heavy dependency between modules
- Application requires specific *system environment*
  For example:
  - **OpenSSH** requires super-user privileges to run; needs multiple user accounts
  - **Quake 3 Arena** needs networked machines to start a client and server
  - **Transmission** requires a host behind NAT with UPnP-IGD protocol enabled router

HOW TO AUTOMATICALLY RUN REGRESSION TEST?

Solution

NixOS

Build and instantiate Virtual Machine and run application inside it

What is NixOS?

What is NixOS?

- OS built out of a Purely functional Package Manager - NIX

What is NixOS?

- OS built out of a Purely functional Package Manager - NIX
- Makefile-like script to build and run a Linux OS

What is NixOS?

- OS built out of a Purely functional Package Manager - NIX
- Makefile-like script to build and run a Linux OS
- Can specify packages to be installed inside the built OS

What is NixOS?

- OS built out of a Purely functional Package Manager - NIX
- Makefile-like script to build and run a Linux OS
- Can specify packages to be installed inside the built OS
- Will automatically dependencies and build them

What is NixOS?

- OS built out of a Purely functional Package Manager - NIX
- Makefile-like script to build and run a Linux OS
- Can specify packages to be installed inside the built OS
- Will automatically dependencies and build them
- Supports multiple builds simultaneously with different configurations

What is NixOS?

- OS built out of a Purely functional Package Manager - NIX
- Makefile-like script to build and run a Linux OS
- Can specify packages to be installed inside the built OS
- Will automatically dependencies and build them
- Supports multiple builds simultaneously with different configurations
- Runs NixOS on QEMU/KVM Hardware Emulator

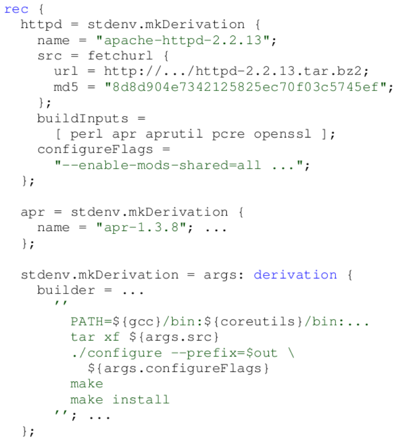
Nix Expression Script

```
derivation {
  name = "foo";
  builder = "${bash}/bin/sh";
  args = [ "-c" "echo Hello $who > $out" ];
  who = "world";
}
```

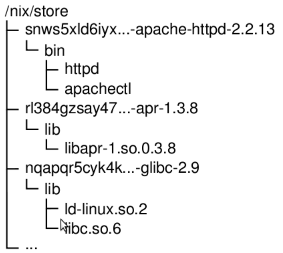
Output: “Hello world” will be written into /nix/store/lw1e3or1p45n2-foo

Nix Expression Script

To build Apache: `nix-build pkgs.nix -A httpd`

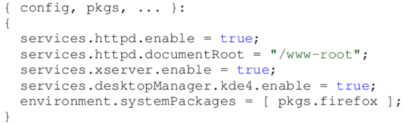


```
rec {
  httpd = stdenv.mkDerivation {
    name = "apache-httpd-2.2.13";
    src = fetchurl {
      url = http://.../httpd-2.2.13.tar.bz2;
      md5 = "8d8d904e7342125825ec70f03c5745ef";
    };
    buildInputs =
      [ perl apr aprutil pcre openssl ];
    configureFlags =
      "--enable-mods-shared=all ...";
  };

  apr = stdenv.mkDerivation {
    name = "apr-1.3.8"; ...
  };

  stdenv.mkDerivation = args: derivation {
    builder = ...
      ''
        PATH=${gcc}/bin:${coreutils}/bin:...
        tar xf ${args.src}
        ./configure --prefix=$out \
          ${args.configureFlags}
        make
        make install
      ''; ...
  };
```



```
/nix/store
├ snws5xld6iyx...-apache-httpd-2.2.13
│ └ bin
│   ├ httpd
│   └ apachectl
├ rl384gzsay47...-apr-1.3.8
│ └ lib
│   └ libapr-1.so.0.3.8
├ nqapqr5cyk4k...-glibc-2.9
│ └ lib
│   ├ ld-linux.so.2
│   └ libc.so.6
└ ...
```

Figure 2. Partial closure of Apache in the Nix store



```
{ config, pkgs, ... }:
{
  services.httpd.enable = true;
  services.httpd.documentRoot = "/www-root";
  services.xserver.enable = true;
  services.desktopManager.kde4.enable = true;
  environment.systemPackages = [ pkgs.firefox ];
}
```

Building NixOS from scratch

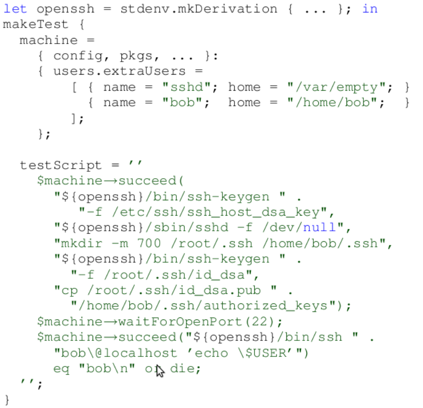
```
nix-build /etc/nixos/nixos -A config.build.system.toplevel
```

Building NixOS from scratch

```
nix-build /etc/nixos/nixos -A config.build.system.toplevel
```

Testing OpenSSH

```
$ nix-build openssh.nix -A vm
$ ./result/bin/run-vm
```

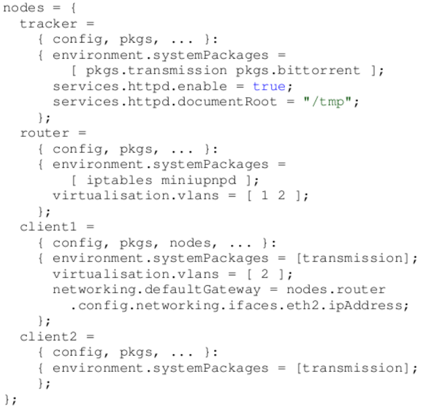


```
let openssh = stdenv.mkDerivation { ... }; in
makeTest {
  machine =
    { config, pkgs, ... }:
    { users.extraUsers =
        [ { name = "sshd"; home = "/var/empty"; }
          { name = "bob";  home = "/home/bob";  }
        ];
    };

  testScript = ''
    $machine→succeed(
      "${openssh}/bin/ssh-keygen " .
         "-f /etc/ssh/ssh_host_dsa_key",
      "${openssh}/sbin/sshd -f /dev/null",
      "mkdir -m 700 /root/.ssh /home/bob/.ssh",
      "${openssh}/bin/ssh-keygen " .
        "-f /root/.ssh/id_dsa",
      "cp /root/.ssh/id_dsa.pub " .
        "/home/bob/.ssh/authorized_keys");
    $machine→waitForOpenPort(22);
    $machine→succeed("${openssh}/bin/ssh " .
      "bob\@localhost 'echo \$USER'")
      eq "bob\n" or die;
  '';
}
```

Figure 4. openssh.nix: Specification of an OpenSSH regression test

Distributed Tests - For Transmission

Network Specification

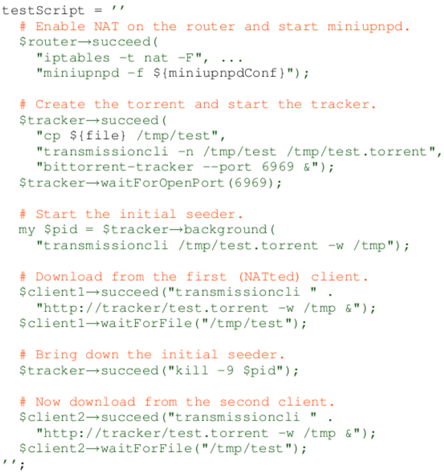


```
nodes = {
  tracker =
    { config, pkgs, ... }:
    { environment.systemPackages =
        [ pkgs.transmission pkgs.bittorrent ];
      services.httpd.enable = true;
      services.httpd.documentRoot = "/tmp";
    };
  router =
    { config, pkgs, ... }:
    { environment.systemPackages =
        [ iptables miniupnpd ];
      virtualisation.vlans = [ 1 2 ];
    };
  client1 =
    { config, pkgs, nodes, ... }:
    { environment.systemPackages = [transmission];
      virtualisation.vlans = [ 2 ];
      networking.defaultGateway = nodes.router
        .config.networking.ifaces.eth2.ipAddress;
    };
  client2 =
    { config, pkgs, ... }:
    { environment.systemPackages = [transmission];
    };
};
```

Figure 6. Network specification for the Transmission regression test

Distributed Tests - For Transmission

Test Script



```
testScript = ''
  # Enable NAT on the router and start miniupnpd.
  $router→succeed(
    "iptables -t nat -F", ...
    "miniupnpd -f ${miniupnpdConf}");

  # Create the torrent and start the tracker.
  $tracker→succeed(
    "cp ${file} /tmp/test",
    "transmissioncli -n /tmp/test /tmp/test.torrent",
    "bittorrent-tracker --port 6969 &");
  $tracker→waitForOpenPort(6969);

  # Start the initial seeder.
  my $pid = $tracker→background(
    "transmissioncli /tmp/test.torrent -w /tmp");

  # Download from the first (NATted) client.
  $client1→succeed("transmissioncli " .
    "http://tracker/test.torrent -w /tmp &");
  $client1→waitForFile("/tmp/test");

  # Bring down the initial seeder.
  $tracker→succeed("kill -9 $pid");

  # Now download from the second client.
  $client2→succeed("transmissioncli " .
    "http://tracker/test.torrent -w /tmp &");
  $client2→waitForFile("/tmp/test");
'';
```

Figure 7. Test script for the Transmission regression test

Evaluation

Coverage:-

Evaluation

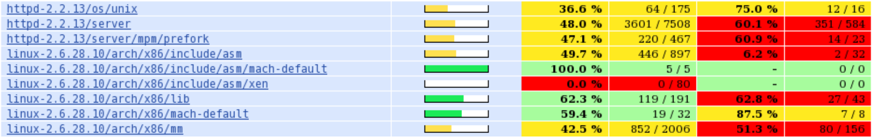
Coverage:-
Apache + SVN + Linux Kernel was instrumented:

Evaluation

Coverage:-
Apache + SVN + Linux Kernel was instrumented:
Run client on one VM, server on another VM; Evaluate coverage together for both!!

Evaluation

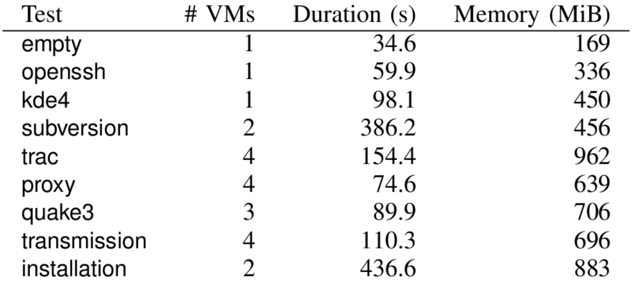
Coverage:-
Apache + SVN + Linux Kernel was instrumented:
Run client on one VM, server on another VM; Evaluate coverage together for both!!



httpd-2.2.13/os/unix		36.6 %	64 / 175	75.0 %	12 / 16
httpd-2.2.13/server		48.0 %	3601 / 7508	60.1 %	351 / 584
httpd-2.2.13/server/mpm/prefork		47.1 %	220 / 467	60.9 %	14 / 23
linux-2.6.28.10/arch/x86/include/asm		49.7 %	446 / 897	6.2 %	2 / 32
linux-2.6.28.10/arch/x86/include/asm/mach-default		100.0 %	5 / 5	-	0 / 0
linux-2.6.28.10/arch/x86/include/asm/xen		0.0 %	0 / 80	-	0 / 0
linux-2.6.28.10/arch/x86/lib		62.3 %	119 / 191	62.8 %	27 / 43
linux-2.6.28.10/arch/x86/mach-default		59.4 %	19 / 32	87.5 %	7 / 8
linux-2.6.28.10/arch/x86/mm		42.5 %	852 / 2006	51.3 %	80 / 156

Evaluation

Resource Consumption on 4-core Intel Core i5 with 6GiB RAM

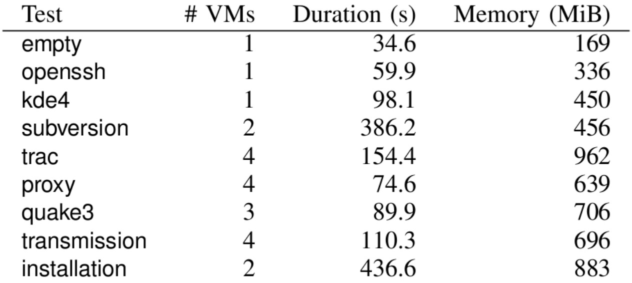


Test	# VMs	Duration (s)	Memory (MiB)
empty	1	34.6	169
openssh	1	59.9	336
kde4	1	98.1	450
subversion	2	386.2	456
trac	4	154.4	962
proxy	4	74.6	639
quake3	3	89.9	706
transmission	4	110.3	696
installation	2	436.6	883

Table I
TEST RESOURCE CONSUMPTION

Evaluation

Resource Consumption on 4-core Intel Core i5 with 6GiB RAM



Test	# VMs	Duration (s)	Memory (MiB)
empty	1	34.6	169
openssh	1	59.9	336
kde4	1	98.1	450
subversion	2	386.2	456
trac	4	154.4	962
proxy	4	74.6	639
quake3	3	89.9	706
transmission	4	110.3	696
installation	2	436.6	883

Table I
TEST RESOURCE CONSUMPTION

>>> Fast enough to do **continuous builds** <<<

Snake Oil

Snake Oil

Seems like an ultimate solution to system testing..

Snake Oil

Seems like an ultimate solution to system testing.. NO !!

Snake Oil

Seems like an ultimate solution to system testing.. NO !!

- GUI testing not possible

Snake Oil

Seems like an ultimate solution to system testing.. NO !!

- GUI testing not possible
- Only for Linux applications

Snake Oil

Seems like an ultimate solution to system testing.. NO !!

- GUI testing not possible
- Only for Linux applications
- Not for scalability testing - Can't spawn 1000 VMs..!!

A Formal Analysis of Requirements-Based Testing

Charles Pecheur,, Franco Raimondi, Guillaume Brat

Problem Landscape

Problem Landscape

- Requirements-based Testing - generate test cases from requirements

Problem Landscape

- Requirements-based Testing - generate test cases from requirements
- Difficult because requirements are in natural language

Problem Landscape

- Requirements-based Testing - generate test cases from requirements
- Difficult because requirements are in natural language
- Very critical for avionics - Ex: MARS Rovers

Problem Landscape

- Requirements-based Testing - generate test cases from requirements
- Difficult because requirements are in natural language
- Very critical for avionics - Ex: MARS Rovers
- Model checkers are slow - Massive state space

Problem Landscape

- Requirements-based Testing - generate test cases from requirements
- Difficult because requirements are in natural language
- Very critical for avionics - Ex: MARS Rovers
- Model checkers are slow - Massive state space
- Requirements are temporal - Ex: if the rover is moving, then all instruments are stored

Solution

Solution

- Express requirements in Linear Temporal Logic

Solution

- Express requirements in Linear Temporal Logic
- FLIP - A formalism prove that an execution path π is an adequate test case for a formula ϕ and an atom *a* appearing in the formula.

Modified Condition/Decision Coverage metric

Modified Condition/Decision Coverage metric

MC/DC metric is critical for avionics software.

Modified Condition/Decision Coverage metric

MC/DC metric is critical for avionics software.
For test suite to achieve MC/DC coverage:

Modified Condition/Decision Coverage metric

MC/DC metric is critical for avionics software.
For test suite to achieve MC/DC coverage:

1. Every basic condition in any decision has been taken on all possible outcomes at least one
2. Each basic condition has been shown to **independently** affect the decision's outcome.

Modified Condition/Decision Coverage metric

MC/DC metric is critical for avionics software.
For test suite to achieve MC/DC coverage:

1. Every basic condition in any decision has been taken on all possible outcomes at least one
2. Each basic condition has been shown to **independently** affect the decision's outcome.

a	b	a ∨ b
T	F	T
F	T	T
F	F	F

- Can't work when conditions are coupled - Ex: $(a \wedge b) \vee (\neg a \wedge c))$

Implementat Details

Implementat Details

- Rules are implemented in *Maude*

Implementat Details

- Rules are implemented in *Maude*
- Verified with NuSMV and Maude module

Implementat Details

- Rules are implemented in *Maude*
- Verified with NuSMV and Maude module
- Restricted to *linear formulae* for requirements

References

1. Christian Murphy, Kuang Shen, and Gail Kaiser. 2009. Automatic system testing of programs without test oracles. In Proceedings of the eighteenth international symposium on Software testing and analysis (ISSTA '09). ACM, New York, NY, USA, 189-200. DOI=10.1145/1572272.1572295

2. van der Burg, S.; Dolstra, E.; , "Automating System Tests Using Declarative Virtual Machines," Software Reliability Engineering (ISSRE), 2010 IEEE 21st International Symposium on , vol., no., pp.181-190, 1-4 Nov. 2010 doi: 10.1109/ISSRE.2010.34

3. Charles Pecheur, Franco Raimondi, and Guillaume Brat. 2009. A formal analysis of requirements-based testing. In Proceedings of the eighteenth international symposium on Software testing and analysis (ISSTA '09). ACM, New York, NY, USA, 47-56. DOI=10.1145/1572272.1572279

Thank you !!