Combinatorial Testing Of ACTS: A Case Study

Mehra N.Borazjany, Linbin Yu, Yu Lei - UTA

Raghu Kacker, Rick Kuhn - NIST

4/17/12

Outline

- Introduction
- Major Features of ACTS
- Input Parameter Modeling
- Experiments
- Conclusion

Motivation

- ACTS is a combinatorial testing tool developed by NIST and UTA
- An ACTS user asked: Have you tested ACTS using ACTS?
- Two objectives
  - Gain experience and insights about how to apply CT in practice.
  - Evaluate the effectiveness of CT applied to a real-life system.

Major Challenges

- How to model the input space of ACTS, in terms of parameters, values, relations and constraints?
  - In particular, how to model a system configuration and the GUI interface?
- How to avoid potential bias as we are the developers of ACTS?
  - What information we know about ACTS can be used in the modeling process?

Major Results

- Achieved about 80% code coverage, and detected 15 faults
- Modeling is not an easy task, especially when the input space has a more complex structure
  - Abstract parameters/values often need to be identified
  - Hierarchical modeling helps to reduce complexity
- Relations are particularly difficult to identify
  - May depend on implementation, and a finer degree of relation may be needed

Major Features of ACTS

- T-Way Test Set Generation
  - Allows a test set to be created from scratch or from an existing test set
- Mixed Strength (or Relation Support)
  - Multiple relations may overlap or subsume each other
- Constraint Support
  - Used to exclude invalid combinations based on domain semantics
  - Integrated with a 3rd-party constraint solver called Choco
- Three Interfaces: Command Line, GUI, and API

Modeling SUT: An Example Configuration

Parameters:

num1:[-1000, -100, 1000, 10000]
num2:[-2, -1, 0, 1, 2]
bool1:[true, false]
bool2:[true, false]
Enum1:[v1, v2, v3, v4, v5, v6, v7, v8, v9]
Enum2:[1, 2]

Relations:

[4,(bool1, bool2, Enum1, Enum2, num1, num2)]
[5,(bool1, bool2, Enum1, Enum2, num1, num2)]
[2,(bool1, bool2, Enum1)]
[2,(Enum1, Enum2, num1)]
[3,(bool1, bool2, Enum1, Enum2, num1)]

Constraints :

enum2="1" && num2+ num1=9999
(num1*num2= 1000) => bool1
num2/num1 <=500 => bool2
enum1="v1"|| num2-num1=9998
num1%num2<900 => num2<0

Modeling SUT: Individual Parameters

Type	Value per parameter
Boolean	Invalid
Integer	[true,false] (default)
Range	One or more (valid values)
Enum	

Type-Value combinations
Boolean type with Invalid value
Boolean type with Default value
Boolean type with one or more value
Integer type with Invalid value
Integer type with one or more value
Enum type with Invalid value
Enum type with one or more value

applicable only for robustness testing of the command line

Modeling SUT: Multiple Parameters

# of Parameters	Parameter Type
Invalid (0 or 1)	A single type
Two	Mixed types
Three or more	

Example:
# of Parameters: ***Three or more***
Parameter Type: ***Mixed types (at least one parameter of each type)***

→

num1:[-1000, 10000]
num2:[-2, -1, 0, 1, 2]
bool1:[true,false]
bool2:[true, false]
Enum1:[v1, v2, v3, v4, v5]
Enum2:[1, 2]
Enum3:[#]

When we derive concrete test cases, we want to cover individual parameters identified earlier at least once.

Modeling SUT: Relations

Individual Relations

Type	Strength
Default	2
User-defined (valid)	3-5
User-defined (invalid)	6

Multiple Relations

# of user-defined relations	Relation between user-defined and default relations
0	Overlap
1	Subsume
Two or more	Subsume default

Modeling SUT: Relation Examples

relation values	**Example**
default	[4,(bool1, bool2, Enum1, Enum2, num1, num2)]
Subsume-default	[4,(bool1, bool2, Enum1, Enum2, num1, num2)] (default) [5,(bool1, bool2, Enum1, Enum2, num1, num2)]
Overlap	[2,(bool1, bool2, Enum1)] [2,(Enum1, Enum2, num1)]
Subsume	[3,(bool1, bool2, Enum1, Enum2, num1)] [2,(bool1, bool2, Enum1, Enum2, num1)]

When we derive concrete test cases, we want to cover individual relations identified earlier at least once.

Modeling SUT: Individual Constraints

Boolean	Arithmetic	Relational
or	+	=
and	*	>
=>	/	<
!	-	≥
	%	≤

Try to test every 2-way combination of the three types of operators

Modeling SUT: Multiple Constraints

# of Constraints	Related Parameters	Satisfiability
0	Some parameters in a relation	Solvable
1	No parameters are not related	Unsolvable
Multiple		

When we derive concrete test cases, we want to cover individual constraints identified earlier at least once.

Modeling SUT: Putting It Together

Test Factors	Test Values
Parameters	Invalid
	Two (1 Integer,1 Enum)
	Three or more (at least 1 Integer,1 Enum, 1 Boolean)
Relations	Invalid parameter (just in CMD interface)
	Default relation
	Two (default and subsume-default)
	Multiple relations (default plus at least 2 subsume)
	Multiple relations (default plus at least 2 overlap)
Constraints	None
	Unsolvable
	Invalid
	One
	Multiple not-related constraints
	Multiple related constraints

Modeling CLI

Test Factors	Test Values	Description
M_mode	scratch	generate tests from scratch (default)
	extend	extend from an existing test set
M_algo	ipog	use algorithm IPO (default)
M_fastMode	on	enable fast mode
	off	disable fast mode (default)
M_doi	specify the degree of interactions to be covered	
M_output	numeric	output test set in numeric format
	nist	output test set in NIST format (default)
	csv	output test set in Comma-separated values format
	excel	output test set in EXCEL format
M_check	on	verify coverage after test generation
	off	do not verify coverage (default)
M_progress	on	display progress information (default)
	off	do not display progress information
M_debug	on	display debug info
	off	do not display debug info (default)
M_randstar	on	randomize don’t care values
	off	do not randomize don’t care values

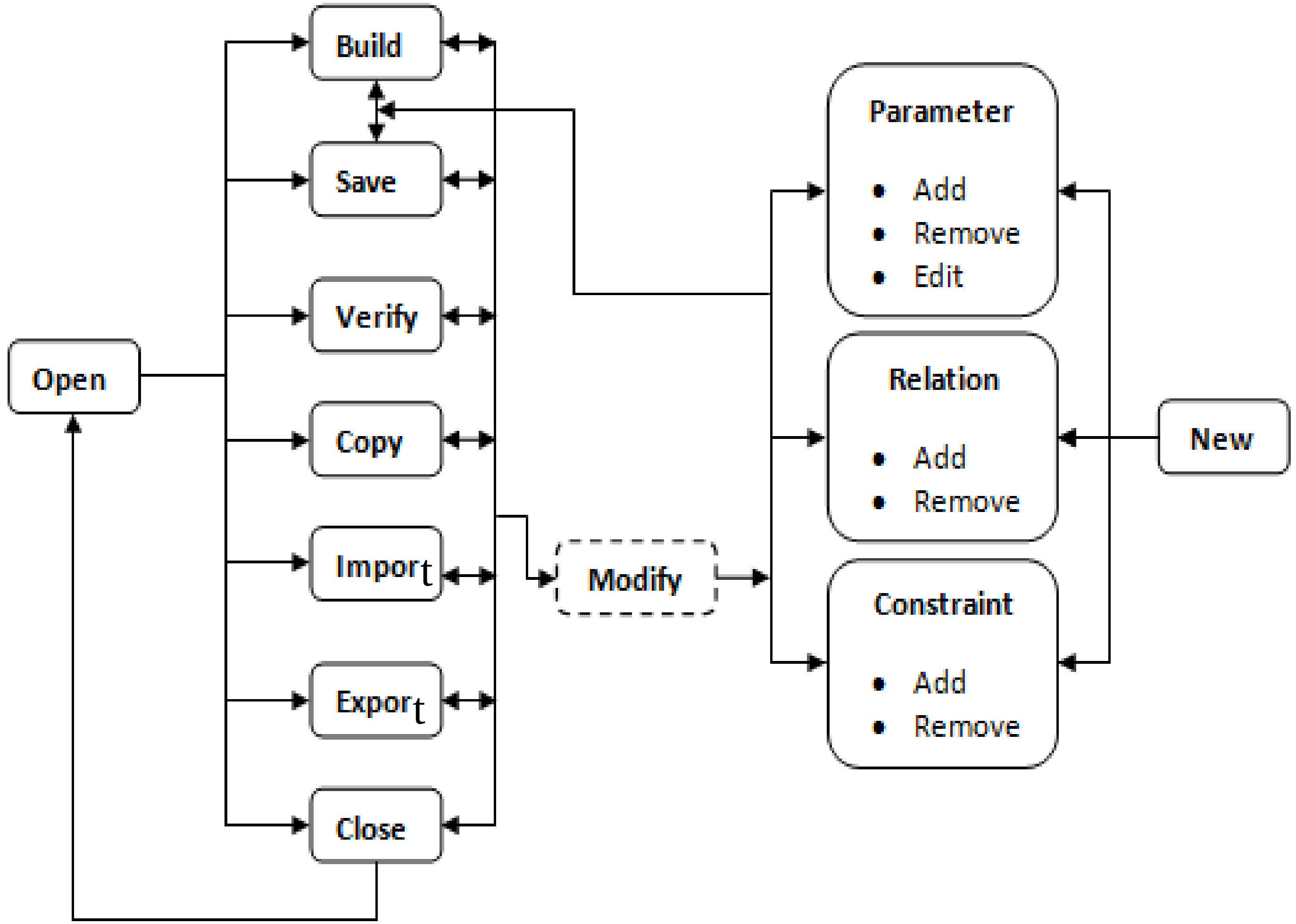
Modeling GUI: Individual Use Cases

- Identify basic use cases and then model each use case separately:
  - Create New System
  - Building the Test Set
  - Modify system (add/remove/edit parameters and parameters values, add/remove relations, add/remove constraints)
  - Open/Save/Close System
  - Import/Export test set
  - Statistics
  - Verify Coverage

Modeling GUI - Add Parameter

Test Factors	Test Values
Parameter name	invalid (space, special_char, number, duplicate name)
	String only
	String plus numeric
Parameter type	Boolean
	Enum
	Number
	Range
In-out	input
	Output
Value	Default
	Valid
	Invalid (Space, duplicate value, invalid range of numbers or characters)

Modeling GUI: Use Case Graph



Modeling GUI: Test Sequence Generation

- Test sequences are generated from the use case graph to achieve 2-way sequence coverage
- If a use case U can be exercised before another use case V, then there must exist a test sequence in which U can be exercised before V

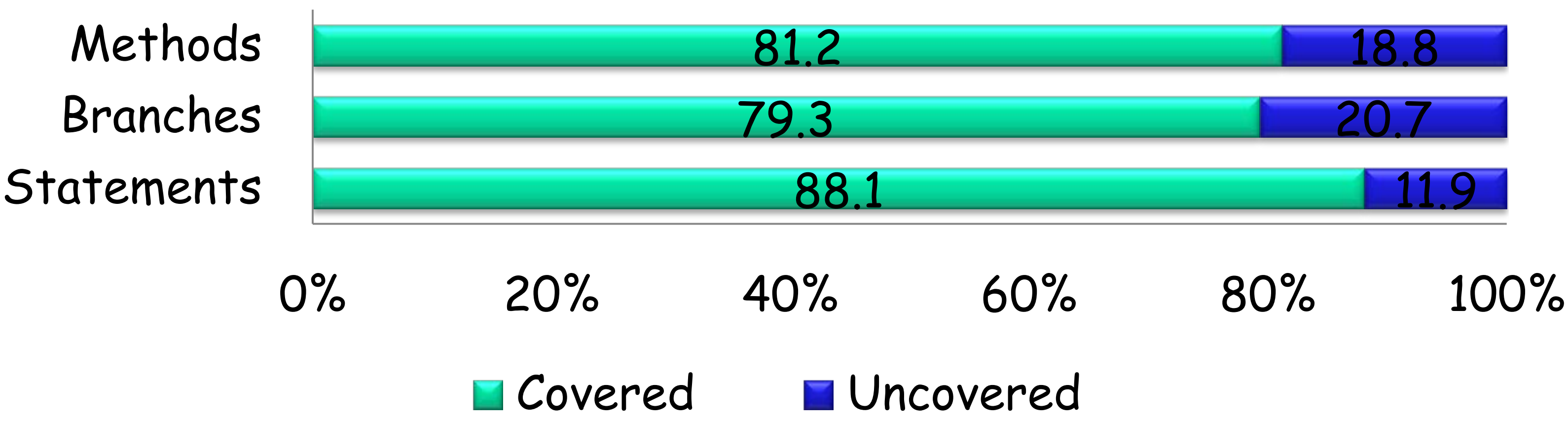
Experimental Design

- Two major metrics:
  - How much code coverage can be achieved?
  - How many faults can be detected?
- Used *clover* to collect code coverage
- Generated test cases with t=2 and extended them to t=3
- 420 test cases for t=2 and 1105 test cases for t=3

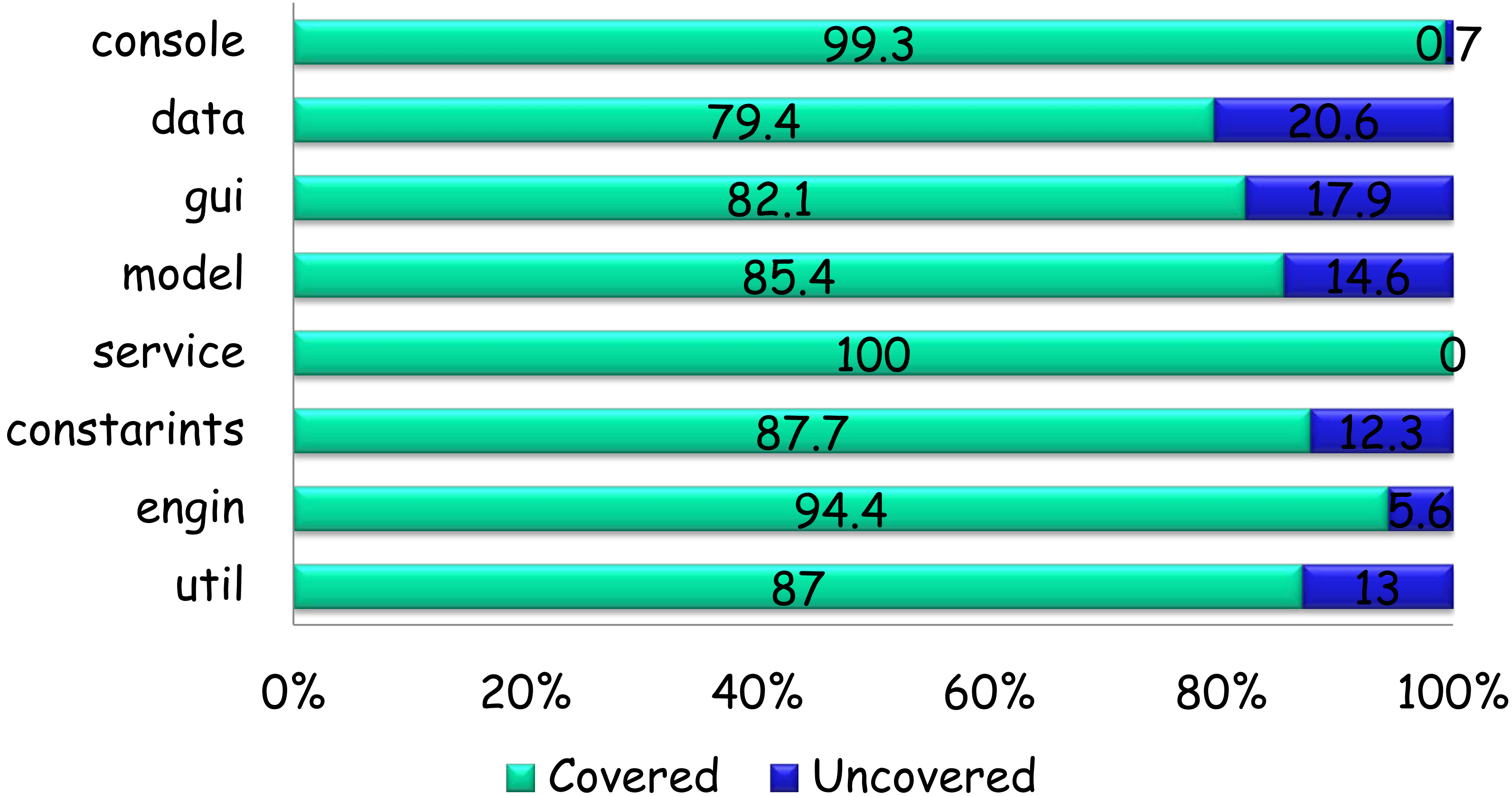
ACTS version 1.2 statistics

LOC	24,637
Number of Branches	4,696
Number of Methods	1,693
Number of Classes	153
Number of Files	110
Number of Packages	12

Code Coverage



Statement Coverage for ACTS packages



Fault Detection

- Detected a total of 15 faults: 10 (positive testing) + 5 (negative testing)
- 8 faults were detected by 2-way test sequences, but not detected by individual use cases
  - For example, a sequence of three use cases, "open, import, build", detected a fault that was not detected by testing the use cases separately
- These faults, however, are not "interaction faults"
  - In the example, "import" created an error state which was not exposed until "build" is exercised.
- 3-way testing did not detect any new faults than 2-way testing

Conclusion

- IPM is a significant challenge of CT
  - The effectiveness of CT largely depends on the quality of the input model
- Significant insights are obtained from this study, but the result of fault detection is a bit puzzling
  - No real interaction faults found, and 3-way testing did not find more faults than 2-way testing
- More research is needed to develop practically useful guidelines, with significant examples, for IPM.
  - More case studies are planned as future work

Thank You