2013 IEEE Sixth International Conference on Software Testing, Verification and Validation

ACTS: A Combinatorial Test Generation Tool

Linbin Yu[1], Yu Lei[1], Raghu N. Kacker[2], D. Richard Kuhn[2]

[1]*Department of Computer Science and Engineering*
University of Texas at Arlington
Arlington, TX 76019, USA
linbin.yu@mavs.uta.edu, ylei@cse.uta.edu

[2]*Information Technology Laboratory*
National Institute of Standards and Technology
Gaithersburg, MD 20899, USA
{raghu.kacker, kuhn}@nist.gov

***Abstract*—In this paper, we introduce a combinatorial test generation research tool called Advanced Combinatorial Testing System (or ACTS). ACTS supports t-way combinatorial test generation with several advanced features such as mixed-strength test generation and constraint handling. To facilitate its use and integration with other tools, ACTS provides three types of external interface, including a graphic user interface, a command line interface, and an application programming interface. ACTS is a freely distributed research tool and has been downloaded by more than 1200 companies and organizations.**

Keywords-component; ACTS; Combinatorial Testing; Test Generation

I. INTRODUCTION

Combinatorial testing applies the theory of combinatorial design to software testing and has been shown to be a very effective testing strategy [1-4]. The key insight behind combinatorial testing is that most faults are caused by interactions among only a few parameters [2]. Given a system with k parameters, t-way combinatorial testing requires all combinations of values of t parameters be covered by at least one test, where t is usually a small integer. Assuming that the test parameters are modeled properly, t-way combinatorial testing is able to expose all the faults that are triggered by no more than t parameters.

In this paper, we introduce a combinatorial test generation research tool called Advanced Combinatorial Testing System (or ACTS). ACTS was first released in 2006 and was formally known as FireEye [5]. ACTS is developed by the US National Institute of Standards and Technology and the University of Texas at Arlington. ACTS supports t-way combinatorial test generation with several advanced features such as mixed-strength test generation and constraint handling. To facilitate its use and integration with other tools, ACTS provides three types of external interface, including a graphic user interface (GUI), a command line interface (CLI), and an application-programming interface (API). ACTS is a freely distributed research tool [11] and has been downloaded by more than 1200 individuals and/or organizations.

The remainder of this paper is organized as follows. Section II introduces the major features of ACTS. Section III describes the external interfaces of ACTS. Section VI describes the test generation engine of ACTS. Section V discusses other test generation tools. Section VI shows the adoption of ACTS. Section VII provides the concluding remarks and also discusses the future work.

II. MAIN FEATURES

In this section, we introduce the major features of ACTS.

A. T-Way Test Generation

ACTS supports t-way combinatorial test generation for up to 6-way coverage. ACTS implements several combinatorial test generation algorithms including IPOG [5], IPOG-D [5], IPOG-F [6], IPOG-F2 [6] and PaintBall. The IPOG algorithm is the main test generation algorithm implemented in ACTS. IPOG typically makes a good balance between the size of a test set and the time required to generate the test set. In addition, as discussed later, features such as mixed strength test generation and constraint handling are only supported for the IPOG algorithm. Algorithms IPOG-F and IPOG-F2 are two variants of the IPOG algorithm. The two variants can produce smaller test sets but typically take more time. The IPOG-D algorithm is another variant of IPOG and works best for larger systems. PaintBall is a random test generation algorithm. Similar to IPOG-D, PaintBall is also preferred for larger systems.

A special form of 1-way testing, called base-choice testing, is implemented in ACTS. In a system under test (SUT), each parameter has one or more values designated as "base choices". Base-choice testing requires that every parameter value be covered at least once and in a test in which all the other values are "base choices". Informally, base choices are "more important" values, e.g. default values, or values that are used most often in operation.

B. Test Set Extension

ACTS supports two test generation modes, namely, scratch and extend. The former allows a test set to be built from scratch, whereas the latter allows a test set to be built by extending an existing test set. In the extend mode, an existing test set can be a test set that is generated by ACTS, but is incomplete because of some newly added parameters and values, or an increased test strength. An existing test set can also be a test set that is supplied by the user and imported into ACTS. Extending an existing test set can save earlier effort that has already been spent in the testing process.

978-0-7695-4968-2/13 $26.00 © 2013 IEEE
DOI 10.1109/ICST.2013.52

C. Mixed Strength Test Generation

This feature allows different parameter groups to be created and covered with different strengths. For example, consider a system consisting of 10 parameters, P1, P2, …, and P10. A default relation can be created that consists of all the parameters with strength 2. Then, additional relations can be created if some parameters are believed to have a higher degree of interaction, based on the user's domain knowledge. For instance, a relation could be created that consists of P2, P4, P5, P7 with strength 3 if the four parameters are closely related to each other, and their 3-way interactions could trigger certain software faults.

ACTS allows arbitrary parameter relations to be created, where different relations may overlap or subsume each other. In the latter case, relations that are subsumed by other relations will be ignored by the test generation engine.

D. Constraint Handling

In practical applications, certain combinations may not be valid from the domain semantics, so they must be excluded from the resulting test set. A test that contains an invalid combination will be rejected by the system if adequate input validation is performed or otherwise may cause the system to fail. In either case, the test will not be executed properly. This may compromise test coverage, if some valid combinations are only covered by this test. ACTS allows the user to specify a set of constraints that must be satisfied by a test. The specified constraints will be taken into account during test generation so that the resulting test set will only contain tests that satisfy these constraints.

E. Coverage Verification

Coverage verification is used to verify whether a test set satisfies t-way coverage, i.e., whether it covers all the t-way combinations. A test set to be verified can be a test set generated by ACTS or a test set supplied by the user. To ensure its validity, the coverage verification procedure is implemented in a way that is independent from the implementation of the test generation algorithms in ACTS. This helps to avoid possibilities where both the verification procedure and the test generation procedure are affected by the same fault.

III. The External Interfaces of ACTS

ACTS is written in Java and thus can be executed on different platforms including Windows, Linux, and MacOS. To facilitate its use and integration with other tools, ACTS provides three types of external interface, including GUI, CLI, and API.

Figure 1 shows the main GUI of ACTS. The system test configurations are displayed on the left side using a tree structure where each configuration contains parameters, constraints, and relations of the SUT. The right side is a tabbed pane consisting of two tabs. The first tab displays a test set, and the second tab displays some useful statistics such as number of tests, execution time, and others. The GUI includes easy-to-use dialogs that allow the user to create and modify a system configuration. It also allows the user to configure different options that controls the test generation process.

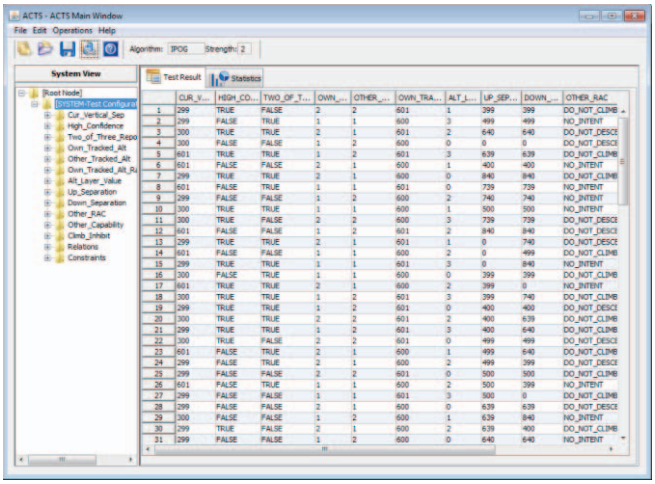

Figure 1. The Main Window of ACTS

The CLI is designed for advanced users and for batch processing and test scripting. The user can specify different options for the test generation process. Figure 2 shows the screen shot of the command line interface of ACTS.

```
D:\ACT\API\cmd test>java -Ddoi=2 -Dcheck=on -Dprogress=on -Ddebug
=off -Doutput=csv -jar 27.jar ActsConsoleManager  D4.txt 1.csv
System Name: C1

Test Generation Profile:
========================================
Degree of interaction: 2
Mode: scratch
Algorithm: ipog
Progress Info: on
Debug mode: off
Ignore constraints: No
Verify Coverage: on

Test Generation Process:
========================================
[2]: 16 tests, 0.015
[3]: 20 tests, 0.0

DONE!

Statistics:
========================================
Number of Tests: 20
Execution Time (seconds): 0.062

Coverage Check:
------------------

Coverage Statistics:
========================================
Total Count of All Possible Tuples: 96
Total Count of Coverable Tuples: 96
Count of Covered Tuples: 96
Count of Missed Tuples: 0
Coverage has been verified!
Output file: 1.csv
```

Figure 2. The Command Line Interface of ACTS

In addition to the GUI and CLI, ACTS provides a set of APIs that allows the user to access the major features in a programmatic manner. Figure 3 shows the Java Doc page of ACTS API, which can be found in [11].

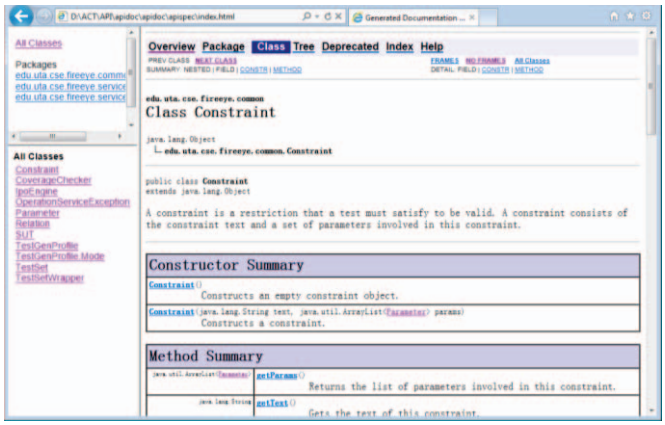

Figure 3. The Java Doc for ACTS API

IV. THE TEST GENERATION ENGINE OF ACTS

The test generation engine of ACTS implements several test generation algorithms. In the following, we first introduce several core data structures that are used in these generation algorithms. We then discuss several key features of ACTS.

A. Core Data Structures

In ACTS, several core data structures are defined and shared among different test generation algorithms.

- **SUT (System Under Test)**. This class represents a system configuration for which a t-way test set is to be generated. An SUT consists of a set of parameters, and may optionally contain a set of constraints and relations. In addition, an existing test set may be included in an SUT. An existing test set may or may not be t-way complete. In the latter case, the existing test set can be extended by ACTS to achieve t-way coverage.
- **Parameter**. A parameter contains a list of values it can take. ACTS supports three types of value, i.e., Boolean, Integer (Int) and Enumerative (Enum). A Boolean value is either "true" or "false". An Int value can be any integer in [0, 2^{32}-1]. An Enum value can be any string. During test generation, only the indices of the parameter values, instead of the real parameter values, are used. However, the real parameter values are needed for the user to write constraints and for the test generation engine to handle constraints.
- **Relation**. A relation contains a set of parameters and a test strength for which these parameters should be covered. A system may contain multiple relations, where different relations may overlap or subsume each other. These multiple relations, if specified, are taken into account during test generation.
- **Constraint**. Constraints are restrictions that must be satisfied by a test. A constraint contains a string that represents the text of the constraint expression and also references to the parameters involved in the constraint. A constraint parser is used to check the syntax of the constraint and also converts the constraint expression to objects that are used in constraint handling. More information about constraint can be found in Section IV.C.
- **Test Set**. A test set contains a list of parameters and a matrix of values in which each row represents a test. The generated test set can be exported using NIST, CSV, or Excel format.

B. Mixed Strength Test Generation

ACTS allows the user to create different parameter groups with different strengths. Since different relations may subsume each other, ACTS will first check and remove relations that are subsumed by others. A relation R is subsumed by another relation R', if R' contains all parameters in R and the test strength of R' is equal to or greater than the test strength of R. For example, consider the following two relations, where R2 is subsumed by R1:

(1) R1: ({P1, P2, P3, P4, P5}, 2)
(2) R2: ({P2, P3, P4}, 2)

Relations may also overlap with each other. Consider the following two relations as an example:

(3) R3: ({P1, P2, P3, P4, P5}, 3)
(4) R4: ({P4, P5, P6}, 2)

R3 and R4 overlap because 2-way parameter combination (P4, P5) of R4 is contained by any 3-way parameter combination of R3 that contains P4 and P5, e.g., 3-way combination (P1, P4, P5) . In ACTS, we generate, for each relation, all the parameter combinations that need to be covered and then remove duplicate parameter combinations and those that are contained by other parameter combinations.

C. Constraint Handling

ACTS allows the user to specify constraints as logic expressions. There are three types of operators in a constraint expression: (1) Boolean operators, including !, &&, ||, =>; (2) Relational operators, including =, !=, >, <, >=, <=; and (3) Arithmetic operators, including +, -, *, /, %.

Figure 4 shows the grammar of the constraint expressions supported by ACTS.

```
<Constraint>  ::= <Simple_Constraint>
                | <Constraint> <Boolean_Op> <Constraint>
<Simple_Constraint> ::= <Term> <Relational_Op> <Term>
<Term> := <Parameter>
        | <Parameter> <Arithmetic_Op> <Parameter>
        | <Parameter> <Arithmetic_Op> <Value>
<Boolean_Op>    := "!" | "&&" | "||" | "=>"
<Relational_Op> := "=" | "!=" | ">" | "<" | ">=" | "<="
<Arithmetic_Op>:= "+" | "-" | "*" | "/" | "%"
<Value> := <Integer_Value> | <Boolean_Value> | <Enum_Value>
```

Figure 4. The Constraint Grammar of ACTS

The following are several examples of constraints that can be specified in ACTS:

- (OS = "Windows") => (Browser = "IE" || Browser = "FireFox" || Browser = "Chrome"), where OS and Browser are two parameters of type Enum. This constraint specifies that if OS is Windows, then Browser must be IE, FireFox, or Chrome.
- (P1 > 100) || (P2 > 100), where P1 and P2 are two parameters of type Int. This constraint specifies that P1 or P2 must be greater than 100.
- (P1 + P2 > P3), where P1, P2, and P3 are parameters of type Int. This constraint specifies that if the summation of P1 and P2 must be greater than P3.
- (P1 = true || P2 >= 100) => (P3 = "ABC"), where P1 is a Boolean parameter, P2 is a parameter of type Int, and P3 is of type Enum. This constraint specifies that if P1 is true and P2 is greater than or equal to 100, then P3 must be ABC.

ACTS integrates an open source constraint solver called Choco [10] for constraint handling. The main operation performed during constraint handling is to check if a test satisfies all the constraints. In order to use Choco, this check is converted to solving a constraint satisfaction problem (CSP). Since the notion of a test does not exist in Choco, one key step in this conversion is to encode a test into a set of constraint objects that can be handled by Choco. Note that the Choco solver is independent from the rest of ACTS and can be replaced by other constraint solvers. More details about constraint solving in ACTS can be found in our recent work [22].

D. Coverage Check

To minimize the chance of coding errors, the coverage check process employs a straightforward algorithm, i.e., without optimization. There are three major steps. First, it generates all possible parameter combinations according to the given test strength. Second, for each parameter combination, it generates all possible value combinations. If constraints are specified, only valid value combinations are generated. Third, for each test in the test set, it marks all the value combinations that can be covered by this test.

V. Other Combinatorial Test Generation Tools

Many tools have been developed for combinatorial test generation. A list of available tools can be found in [20]. All of the existing tools support the core feature of t-way combinatorial test generation. However, they differ in several aspects: (1) core test generation algorithms, which may result in different runtime performances in terms of test set size and test set generation time; (2) support for advanced features such as mixed strength test generation and constraint handling, which may affect their applicability to a given system; and (3) external interfaces such as GUI, command-line interface, and API, which is an important factor which it comes to adoption in practice.

A detailed comparison of the existing tools is beyond the scope of this paper. In the following, we discuss the major types of core test generation algorithm and mention several example tools for each type. The test generation algorithms can be roughly classified into two groups, computational methods and algebraic methods. Computational methods usually involve an explicit enumeration of all possible combinations, and employ a greedy or heuristic search strategy to select tests. Example tools that use the greedy strategy include AETG-web [9], ACTS [5], PICT [7], T-tuples [15] and DDA [18]. An example tool that uses the heuristic search strategy is CASA [12]. In contrast, algebraic methods build a t-way test set based on some pre-defined formulas, without enumerating any combinations. Example tools that use algebraic methods include TConfig [13] and CTS [14]. Recently, algorithms are proposed that transform the problem of combinatorial test generation into a formal logic problem that can be solved by existing SAT modulo theory solvers. Example tools that use these algorithms include EXACT [21] and ATGT [16].

We are continuing to improve ACTS. In particular, we proposed an efficient algorithm and several optimizations to improve the performance of constraint handling [22].

VI. Adoption of ACTS

ACTS has been downloaded by more than 1200 individual and/or institutions. We are continuing to receive download requests every day. Figure 5 shows a distribution of ACTS users in terms of different industries.

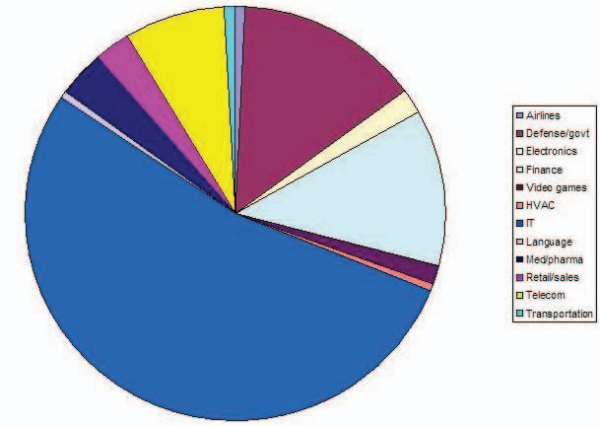

Figure 5. Distribution of ACTS Users

VII. Conclusion

In this paper, we introduced a combinatorial test generation research tool called ACTS, which supports t-way combinatorial test generation for up to 6-way coverage. ACTS supports mixed-strength test generation, constraint handling, test set extension, and coverage verification, and provides three external interfaces, i.e., a graphic user interface, a command line interface, and an application-programming interface. ACTS is a freely distributed research tool that can be downloaded at [11].

We are continuing to improve ACTS as we respond to bug reports and feature requests from the users. Moreover, we plan to add two new major features. First, we plan to support negative testing, where each test contains only one "invalid" value. Negative testing is critical to ensure robustness of a system. Second, we plan to enhance constraint handling with support for some higher-level constraints. For example, consider a system consisting of 10 Boolean parameters. An operator "count" could be very convenient to specify a constraint that the number of "false" values in a test should be no more than a given number. Similar higher-level constraints can facilitate the specification of constraints for many applications.

ACKNOWLEDGMENT

This work is partly supported by three grants (70NANB9H9178, 70NANB10H168, 70NANB12H175) from Information Technology Laboratory of National Institute of Standards and Technology (NIST) and a grant (61070013) of National Natural Science Foundation of China.

DISCLAIMER: Identification of commercial products in this report does not imply recommendation or endorsement by NIST.

REFERENCES

[1] D.R. Wallace, D.R. Kuhn. Failure modes in medical device software: An analysis of 15 years of recall data. International Journal of Reliability, Quality and Safety Engineering 2001

[2] D.R. Kuhn, M.J. Reilly. An investigation of the applicability of design of experiments to software testing. Proceedings of 27th NASA/IEEE Software Engineering Workshop, Greenbelt, Maryland, 2002; 91–95.

[3] Tung YW, Aldiwan WS, Automating test case generation for the new generation mission software system. Proceedings of IEEE Aerospace Conference, Big Sky, Montana, 2000

[4] D.R. Kuhn, D.R. Wallace, Jr. A.M. Gallo, Software fault interactions and implications for software testing, Software Engineering, IEEE Transactions on, 2004

[5] Y. Lei, R. Kacker, D.R. Kuhn, V. Okun, J. Lawrence, IPOG: A general strategy for t-way software testing. In: 14th international conference on the engineering of computer-based systems, 2007, pp 549–556

[6] M. Forbes, J. Lawrence, Y. Lei, R.N. Kacker, and D.R. Kuhn Refining the In-Parameter-Order Strategy for Constructing Covering Arrays, NIST Journal of Research, 113(5):287-297, Sept./Oct., 2008

[7] Czerwonka J., Pairwise testing in real world. In: 10th Pacific northwest software quality conference, pp 419–430, 2006

[8] D. M. Cohen, S. R. Dalal, M. L. Fredman, and G. C. Patton. The AETG system: An approach to testing based on combinatorial design. IEEE Transactions On Software Engineering, 23(7):437–444, 1997.

[9] S. R. Dalal, A. Jain, G. Patton, M. Rathi, and P. Seymour, AETG Web: A Web Based Service for Automatic Efficient Test Generation from Functional Requirements, Proceedings of the Workshop on Industrial Strength Formal Techniques (WIFT), Boca Raton, FL, October 1998.

[10] The Choco Constraint Solver, http://www.emn.fr/z-info/choco-solver/index.html.

[11] ACTS Home Page, http:// csrc.nist.gov/acts/.

[12] M.B. Cohen, M.B. Dwyer, J. Shi, Constructing interaction test suitees for highly-configurable systems in the presence of constraints: a greedy approach. IEEE Trans. Softw. Eng. 34, 633–650, 2008

[13] Williams AW. Determination of test configurations for pair-wise interaction coverage. Proceedings of 13th International Conference on the Testing of Communicating Systems, Ottawa, Canada, 2000; 59–74.

[14] Alan Hartman, Leonid Raskin. Software and hardware testing using combinatorial covering suitess. Graph Theory, Combin. Algor. 6, 3, 237–266.

[15] Calvagna A, Gargantini A., T-wise combinatorial interaction test suitees construction based on coverage inheritance. In: Software Testing, Verification and Reliability, 2009

[16] A. Calvagna and A. Gargantini. A logic-based approach to combinatorial testing with constraints. In B. Beckert and R. Hähnle, editors, Tests and Proofs, Second International Conference, TAP 2008, Prato, Italy, April 9-11, 2008.

[17] M. B. Cohen, C. J. Colbourn, P. B. Gibbons, W. B. Mugridge, Constructing test suites for interaction testing. Proceedings of 25th IEEE International Conference on Software Engineering, 2003.

[18] Charles J. Colbourn , Myra B. Cohen, A Deterministic Density Algorithm for Pairwise Interaction Coverage, Proc. of the IASTED Intl. Conference on Software Engineering, 2004

[19] M. A. Chateauneuf, C. J. Colbourn, D. L. Kreher, Covering arrays of strength 3. Designs, Codes, and Cryptography 1999;16:235–242.

[20] Pairwise Testing Home Page: http://pairwise.org

[21] Jun Yan, Jian Zhang, Backtracking Algorithms and Search Heuristics to Generate Test Suites for Combinatorial Testing, 30th Annual International Computer Software and Applications Conference (COMPSAC'06), 2006

[22] Linbin Yu, Mehra Nourozborazjany, Yu Lei, Raghu N. Kacker and D. Richard Kuhn, An Efficient Algorithm for Constraint Handling in Combinatorial Test Generation, the sixth IEEE International Conference on Software Testing, Verification and Validation (ICST 2013, to appear), 2013,.

Appendix: Tool Demonstration Plan

We plan to demonstrate several major use cases of ACTS. The demonstration has three major parts, one for each external interface, i.e., GUI, CLI, and API, respectively.

Part 1: GUI Demonstration

Scenario 1: Generate a t-way test set

a) Open the GUI and show all the menu and options.
b) Create a new system configuration with 2 int parameters, 2 boolean parameters and 1 enum parameter.
c) Build test set using default options, and *s*how the resultant test set and statistics information.
d) Open the options window, explain each option.
e) Change one or more options and generate a new test set.
f) Save the system.

Scenario 2: Generate a t-way test set with constraints

a) Open the system configuration created in Scenario 1
b) Add one or more constraints into the system configuration
c) Generate a 3-way test set using default options
d) Show the statistics tab and compare the number of covered tuples with and without constraints.

Scenario 3: Generate a t-way test set with mixed strength

a) Open the system configuration created in Scenario 1
b) Add one or more relations into the system configuration
c) Generate a 3-way test set using default options
d) Show the statistics tab and compare the number of covered tuples with and without constraints.

Scenario 4: Extend an existing test set

a) Generate a 2-way test set using scratch mode
b) Set test strength to 3 and Run coverage checker (coverage not satisfied)
c) Rebuild using extend mode. Show the results.
d) Check coverage again (should satisfy now)

Part 2: CLI Demonstration

Scenario 5: Generate a t-way test set

a) Run ACTS on the command line without an input file to show the commmand line usage information.
b) Open a text editor to show a configuration file that is written in advance
c) Run ACTs on the command line with the configuration file to generate a t-way test set
d) Open a text editor to show the output file which contains the resultant test set

Part 3: API Demonstration

Scenario 6: Generate a t-way test set

a) Show the Java Doc of ACTS API
b) Open Eclispes and show an example progam that uses ACTS API.
c) Explain the major data structures and steps involved in the use of the API.
d) Execute the example program
e) Show the test set generated by the example program.