The Journal of Systems and Software 99 (2015) 1–19

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

PROW: A Pairwise algorithm with constRaints, Order and Weight

Beatriz Pérez Lamancha [a,*], Macario Polo [b], Mario Piattini [b]

[a] Software Testing Centre, Republic University, Montevideo, Uruguay
[b] Alarcos Research Group, Castilla-La Mancha University, Ciudad Real, Spain

ARTICLE INFO

Article history:
Received 28 November 2012
Received in revised form 23 July 2014
Accepted 2 August 2014
Available online 16 September 2014

Keywords:
Software testing
Combinatorial testing

ABSTRACT

Testing systems with many variables and/or values is often quite expensive due to the huge number of possible combinations to be tested. There are several criteria available to combine test data and produce scalable test suites. One of them is pairwise. With the pairwise criterion, each pair of values of any two parameters is included in at least one test case. Although this is a widely-used coverage criterion, two main characteristics improve considerably pairwise: constraints handling and prioritisation.

This paper presents an algorithm and a tool. The algorithm (called PROW: Pairwise with constRaints, Order and Weight) handles constraints and prioritisation for pairwise coverage. The tool called CTWeb adds functionalities to execute PROW in different contexts, one of them is product sampling in Software Product Lines via importing feature models. Software Product Line (SPL) development is a recent paradigm, where a family of software systems is constructed by means of the reuse of a set of common functionalities and some variable functionalities. An essential artefact of a SPL is the feature model, which shows the features offered by the product line, jointly with the relationships (includes and excludes) among them. Pairwise testing could be used to obtain the product sampling to test in a SPL, using features as pairwise parameters. In this context, the constraint handling becomes essential. As a difference with respect to other tools, CTWeb does not require SAT solvers.

This paper describes the PROW algorithm, also analysing its complexity and efficiency. The CTWeb tool is presented, including two examples of the PROW application to two real environments: the first corresponds to the migration of the subsystem of transactions processing of a credit card management system from AS400 to Oracle with .NET; the second applies both the algorithm and the tool to a SPL that monitors and controls some parameters of the load in trucks.

1. Introduction

Software development suffers from the impossibility of exhaustive testing. Thus, researchers strive to find a balance between test suite size and coverage (e.g. the ability to find faults in the system under test). Different test generation techniques produce different test suites aimed at reaching a certain test data coverage criterion (i.e. using each test datum at least once, using all pairs of values, etc.).

Combination strategies (also called Combinatorial Interaction Testing, CIT) are a type of test case selection method where test cases are created by the combination of "*interesting values*", which have been previously identified by the tester.

The input is a set of parameters, each with some elements (values). The output is a set of combinations, each one composed by one element of each input set (Grindal et al., 2005). For instance, *t-wise* strategies ensure that all combinations of any t parameters (input sets) are included in at least one test case (Grindal et al., 2005). Pairwise is a particular case of t-wise, where $t=2$. In this case, the coverage criterion is that all the pairs between the values of any two parameters are included in at least one test case.

Fig. 1 shows the possible configuration elements of a web application that could be executed with different options for Operating System, Browser, Word processor and DataBase. A test suite that achieve pairwise coverage for this example is shown in Fig. 2, which contains all the possible combinations between pairs of values for Operating System, Browser, Word processor and DataBase. However, this test suite does not have into account the relationship between pairs of values, and can lead to the inclusion of many test cases that contain semantically meaningless combinations of test data. For the configurations in Fig. 2, the cases combining *Linux* with *IExplorer* or *Linux* with *Microsoft Word* (test cases 1, 12, 13 and 16) make no sense. If the unfeasible test cases are directly removed from the suite, then there will be interesting

* Corresponding author.
E-mail addresses: bperez@fing.edu.uy (B. Pérez Lamancha), macario.polo@uclm.es (M. Polo), mario.piattini@uclm.es (M. Piattini).

http://dx.doi.org/10.1016/j.jss.2014.08.005
0164-1212/

Operating System	Browser	Word Processor	DataBase
Linux	Firefox	Microsoft Word	SQL
Windows	IExplorer	Open Office	Oracle
Mac OS	Opera	Feng Office	
	Chrome	Google Docs	

Fig. 1. Configuration parameters.

Test Case	Operating System	Browser	Word Processor	DataBase
1	Linux	Firefox	Microsoft Word	SQL
2	Windows	IExplorer	Open Office	Oracle
3	Mac OS	Opera	Feng Office	SQL
4	Linux	Chrome	Google Docs	Oracle
5	Windows	Firefox	Feng Office	Oracle
6	Mac OS	IExplorer	Microsoft Word	SQL
7	Windows	Opera	Google Docs	SQL
8	Mac OS	Chrome	Open Office	Oracle
9	Linux	Opera	Open Office	SQL
10	Windows	Chrome	Microsoft Word	Oracle
11	Mac OS	Firefox	Google Docs	SQL
12	Linux	IExplorer	Feng Office	SQL
13	Linux	Opera	Microsoft Word	Oracle
14	Linux	Chrome	Feng Office	SQL
15	Linux	Firefox	Open Office	SQL
16	Linux	IExplorer	Google Docs	SQL

Fig. 2. Test cases obtained using pairwise criterion.

pairs which will remain untested: if test case 1 (*Linux, Firefox, Microsoft Word*) is removed due to the incompatibility of parameters 1 and 3, then the pair (*Firefox, Microsoft Word*), which is legal and should be tested, would be outside the testable configurations. Then, the further removal of invalid test cases is not an efficient solution, since valid pairs will be also removed from the test suite.

Detecting these combinations in advance requires resorting to the semantics of the involved parameters, i.e. what the parameters stand for. The tools must not focus strictly on providing an algorithmic solution to the mathematical problem of combinatorial testing, also account for other complementary features, which are rather important in order to make these tools really useful in practice, such as the ability to handle constraints on the input domains (Calvagna and Gargantini, 2008; Grindal et al., 2005).

A recent software development paradigm where combination testing is being applied is Software Product Line (SPL). A SPL is "a set of software-intensive systems, sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way" (Clements and Northrop, 2001). Products in an SPL share a set of characteristics (commonalities) and differ in a number of variation points, which represent the variabilities of the products.

Variability is a central concept in SPL development. It allows for the generation of different products in the line by reusing core assets. Variability is captured through features. A feature is an increment in program functionality that customers use to distinguish one SPL product from another (Kang et al., 1990). A feature can be a specific requirement, a selection among optional or alternative requirements, or can be related to certain product implementation or characteristics (Griss, 2000). Several documented examples of SPL exists,[1] for example the Nokia SPL where several mobile phones share a set of common characteristics: in this case a feature can be the language, and the feature variants can be: Spanish, French, Chinese, among others. Feature models are used to relate features to each other in various ways, showing sub-features, alternative features, optional features, dependent features or conflicting features (Griss, 2000).

One of the main problems in SPL testing is the selection of the products to test. From a feature model, several products can be built, and the possible combinations of features to obtain products can make it impossible to test all of them. One possible solution is obtaining a Product sampling set to SPL testing. To this end, combinatorial testing strategies can be used, taking the features as parameters and the feature variants as parameters values. More information can be obtained from a feature model: we also would get all the possible relationships between features (includes, excludes) and be able to represent this with the combinatorial technique. These relationships between features restricts the possible combination of products, for example, if a feature "excludes" another feature, then all the products that combines the variants of these features are meaningless, the relationships in a feature model can be handled automatically using constraint handling. Due the relationships in a feature model defines the products that can be generated in the SPL, the functionality of constraints handling that could be desirable in combinatorial testing algorithms in SPL testing becomes essential.

The contribution of this article is twofold: first, it presents an algorithm called PROW (Pairwise with constRaints, Order and Weight) for generating pairwise test suites powered by domain semantics and knowledge, which are captured through:

- Constraints: The algorithm makes it possible to exclude all the pairs between parameter values that are semantically meaningless.
- Weight assignment: The algorithm is also capable of including those pairs which require more frequent testing, assigning a weight to each pair between parameter values, this value is referred in this paper as *"pair weight"*. With this, the tester can specify the most important pairs from a testing point of view.
- Ordering for test case prioritisation: The test suite generated by the algorithm is ordered using the summation of the *"pair weights"* involved in the test case. The ordered test suite can be used to test planning, since the most important test cases must be tested early. This is especially useful in regression testing, when the time for testing is not sufficient to execute the complete test suite.

The PROW algorithm has been implemented in a publicly available web tool, under a GNU license called CTWeb. This is the second contribution: CTWeb tool provides features that allow the use of PROW efficiently and also for product sampling in SPL. Some of the functionalities that provide for CTWeb to improve PROW include:

- Product sampling: Allows to load a feature model and automatically processes the features and their relationships to obtain the parameters to execute PROW. The result is the set of products to be tested in the SPL.
- Setting the base test suite: Test cases obtained with combinatorial testing may not exactly be the most widely used in reality, even if weight is assigned to prioritise them. This functionality allows the tester to define a set of test cases as the basis on which to run PROW. This base test suite is uploaded in a tab-separated file and CTWeb mark all pairs covered by the base test suite as visited prior to running the algorithm PROW. This functionality can be used in both, combinatorial testing and product sampling. The result is a set of test cases that satisfies pairwise coverage and contains the base test suite as a subset.

[1] Product Line Hall of Fame: http://splc.net/fame.html.