STG: A Tool for Generating Symbolic Test Programs and Oracles from Operational Specifications *

Duncan Clarke, Thierry Jéron, Vlad Rusu and Elena Zinovieva
IRISA/INRIA Rennes, Campus de Beaulieu, Rennes, France
{dclarke,jeron,rusu,enz}@irisa.fr

ABSTRACT

We report on a tool we have developed that automates the derivation of tests from specifications. The tool implements conformance testing techniques to derive symbolic tests that incorporate their own oracles from formal operational specifications. It was applied for testing a simple version of the CEPS (Common Electronic Purse Specification).

1. INTRODUCTION

The work that we present is an attempt to leverage the ideas underlying protocol conformance testing [9] and high-efficiency test generation as embodied in the TGV [5], TorX [1] tools, to automate the generation of tests for a more general class of applications. TGV, like most existing tools, performs its analysis by enumerating the speciﬁcation’s state space. This leads to two problems: (1) state-space explosion, as the variables in the speciﬁcation are instantiated with all of their possible values, and (2) tests that are not readily understandable by humans. To avoid these problems we are applying symbolic techniques to perform our analysis.

In [8] the authors present a method for the generation of symbolic test cases from system speciﬁcations and test purposes expressed in the Input/Output Symbolic Transition System (IOSTS) formalism. The models of speciﬁcation, test purpose and test are symbolic in the sense that items represented by variables over data domains in the parameterization of the model and in the control and computation steps of the speciﬁcation remain variables in the generated tests. At no point in the analysis is it necessary to enumerate the state space of the speciﬁcations, test purposes or generated tests.

In addition to the elimination of the need for state space enumeration, the principal beneﬁts of this approach are fourfold: (1) The derivation of tests and oracles [6] from formal, operational speciﬁcations can be fully automated; (2) The tests are symbolic in the parameters of the speciﬁcation, so a single test can be generated and applied to implementations based on different speciﬁcation parameter values; (3) The resulting tests are concrete, in the sense that once parameters are instantiated the tests can be translated easily to a test language and applied directly to real implementations; and (4) the theory underlying the derivation of tests guarantees certain desirable properties, such as no false positive outcomes, no false negative outcomes, etc.

2. STG: THE SYMBOLIC TEST GENERATION TOOL

Based on the theory of symbolic test generation presented in [8] we have created the STG tool that implements the process illustrated in Figure 1. The features currently implemented by the tool are found inside the dashed box. The processes shown above the box are the subject of ongoing research, and are brieﬂy described in Section 3.

Currently STG supports two processes (cf. Figure 1), which are brieﬂy described below.

Symbolic test generation. The process of symbolic test generation consists of computing, from a speciﬁcation and a test purpose, a test case that covers all the behaviors of the intersection of these two elements. This process consists of the following steps:

- product between test purpose and speciﬁcation. This allows to select a subgraph of the speciﬁcation that formally leads to the satisfaction of the test purpose. That is, only the subgraph leading to accepting states of the test purpose is kept. Here, “formally” refers to the fact that symbolic variables and parameters are not interpreted, thus, actual reachability of the accepting states is not guaranteed. This problem is addressed in the simplification step.

- closure and determinization of the product. This operation attempts, through a set of heuristics, to produce a trace-equivalent system that has no (or fewer) internal actions and is deterministic. The heuristics will successfully terminate when applied to command/response systems.

- adding verdicts. This step consists in labeling some states with verdicts. “Pass” means that the test purpose was satisfied and no errors were detected (i.e., there were no observable differences between implementation and speciﬁcation). “Inclusive” means that, although no errors were detected, the test purpose cannot be satisfied any more. “Fail” means that an error was detected.

*Supported in part by DYADE action FormalCard, a joint project of INRIA and Bull/CP-8.
3. FUTURE WORK

A first direction of our future work is to use a higher-level language (e.g., LOTOS, SDL, etc.) for specifying systems at the user level, which translates automatically to IOSTS (cf. Figure 1). Second, we plan to work on the implementation of the mechanism to automatically compute test purposes from the system specification (cf. Figure 1) using, for example, coverage criteria [7] instead of test purposes written by hand.

4. REFERENCES