Constraint-Based Software Testing*

Sebastian Bardin[1], Bernard Botella[1], Frédéric Dadeau[3], Florence Charreteur[2], Arnaud Gotlieb[2], Bruno Marre[1], Claude Michel[4], Michel Rueher[4], and Nicky Williams[1]

[1] CEA LSL, 91191 Gif sur Yvette
[2] INRIA Rennes-Bretagne Atlantique, LANDE Project, 35042 Rennes Cedex
[3] University of Franche-Comté, LIFC / INRIA CASSIS Project, 25030 Besançon
[4] University of Nice-Sophia Antipolis, CeP project, Ecole Polytechnique, 06903 Sophia Antipolis cedex

Abstract. Constraint-Based Testing (CBT) is the process of generating test cases from programs or models by using the Constraint Programming technology. Recently, this method received much attention due to several Research projects launched in France and abroad. This paper aims at presenting the main CBT tools developped by four Research labs: CEA *Laboratoire de Sûreté des Logiciels*, INRIA *Lande research team*, *Laboratoire d'Informatique de Franche-Comté*, and *CeP team* of University of Nice–Sophia Antipolis. The paper concludes by drawing some perspectives on open problems in CBT.

1 Introduction

These last years, much attention has been devoted to the use of Constraint Programming techniques in the automation of program verification. In 2000, Podelski outlined [13] that program verification could be seen as an instanciation of constraint solving while Flanagan proposed theoretic foundations to constrained interpretations of programs [7]. Program verification often reduces to the problem of showing that a constraint system is satisfiable or unsatisfiable. For example, showing a particular property at a given point in a source code leads to solve a constraint system that characterizes a path through a particular state of the program. Recent work focussed on the use of constraint solvers for bounded model checking of source code, but it is in automated software testing that constraint solving for program verification has reached a certain level of maturity.

In this domain, France has been a pioneer since the early work of Marre, Dick and Faivre[11, 6]. Several research projects were launched such as the *RNTL projects INKA (2000-2002) and DANOCOPS (2004-2006)*, that initiated the study of constraint-based testing techniques for embedded C and C++ programs. These projects resulted not only in the development of the INKA tool which was the first to use constraint programming in automatic structural test

* partially granted by the SESUR CAVERN project ref. ANR-07-SESU-003.

data generation for C programs [9, 10] but also to the development of prototypes tools for specification notations such as OCL and JML[3]. The *ACI V3F project (2003-2006)* studied the problems of floating-point computations within various Constraint-Based Testing approaches. Thanks to this project, we started working together around the development of specialized floating-point constraint solvers [2]. Another result of the *ACI V3F* was the first international CSTVA workshop (Constraints in Software Testing, Verification and Analysis), we organized in Nantes in 2006. CSTVA[5] brought together people of distinct communities to discuss fruitful ideas around the use of constraints in program testing. From these discussions emerged the idea of exploiting abstractions to enhance current constraint propagation in solvers dedicated to program testing. The current *SESUR–2007 CAVERN (Constraints and Abstractions for program VERificatioN) project* which started early 2008 aims to explore the combination of Constraint Programming and Abstractions techniques for automated testing of programs. The originality of this project lies in the use of abstractions to develop dedicated propagation-based constraint solvers targeted to handle specific features of imperative programs such as iterative computations, references, dynamic structures and floating-point computations.

Since 2000, several Constraint-Based Testing tools have been developed to investigate constrained models of programs or specification models with the goal of generating test cases against various testing objectives. These tools share some characteristics such as being based on constraint propagation, abstract domains computations and labeling, The purpose of this short paper is to give an overview of some of these tools: PathCrawler and Euclide for C, Cpbpv for Java, Osmose for executable code, Jaut for Bytecode Java,Gatel for Lustre, JMLTT for JML.

2 Tools

– PathCrawler [14] is a tool prototype developed by the LSL laboratory of CEA List. PathCrawler automatically generates test-case inputs guaranteeing full structural coverage of the C function under test: all feasible paths, all reachable branches, all k-paths,... PathCrawler runs an automatically-instrumented version of the function under test on each test-case as soon as it is generated in order to recover the path covered by this test-case and ensure that the next test-case covers a path which is not covered yet. Test-case generation is implemented using constraint logic programming. To ensure that PathCrawler can be used on real-life programs, current work focusses on the treatment of called functions, the treatment of the precondition on the effective input parameters under which the function is to be tested, the treatment of pointer casts, the treatment of integer overflows and floating-point numbers and the early detection of infeasible path prefixes.
– Euclide [8] is a new Constraint-Based Testing tool for verifying safety-critical C programs. By using a mixture of symbolic and numerical analyses

[5] http://www.irisa.fr/manifestations/2006/CSTVA06

(namely static single assignment form, constraint propagation, integer linear relaxation and search-based test data generation), it addresses three distinct applications in a single framework: structural test data generation, counter-example generation and partial program proving. The core algorithm of the tool takes as input a C program and a point to reach in the code. As a result, it outcomes either a test datum that reaches the selected point, or an "unreachable" indication showing that the selected point is unreachable. Optionally, the tool takes as input additional safety properties that can be given under the form of pre/post conditions in ACSL or assertions directly written in the code. In this case, EUCLIDE can either prove that these properties or assertions are verified or find a counter-example when there is one. As these problems are undecidable in the general case, EUCLIDE only provides a semi-correct procedure (when it terminates, it provides the right answer) for them. Current Research works around the tool focus on modular integers and floating-point computations and better constraint solving procedures based on relational abstract domains computations.

- CPBPV [5] is a novel constraint-programming framework for bounded program verification. The CPBPV framework uses constraint stores to represent the specification and the program and explores execution paths non-deterministically. The input program is partially correct if each constraint store so produced implies the post-condition. CPBPV does not explore spurious execution paths as it incrementally prunes execution paths early by detecting that the constraint store is not consistent. CPBPV is parameterized with a list of solvers which are tried in sequence, starting with the least expensive and less general. Experimental results often produce orders of magnitude improvements over earlier approaches, running times being often independent of the variable domains. Moreover, CPBPV was able to detect subtle errors in some programs while other frameworks based on model checking have failed.
- OSMOSE [1] is a tool dedicated to machine code analysis. Potential applications include validation of COTS and mobile codes as well as malware comprehensive analysis. There are two main specific challenges in machine code analysis: low-level semantics of data (bitwise instructions, machine arithmetics with overflows and flags, etc.), and unstructured control-flow (e.g. `goto x`, where `x` is only known at run-time). From a test data generation perspective, OSMOSE follows both the path-based approach and the concolic execution paradigm, mixing concrete execution and symbolic reasoning. The tool is geared toward full coverage of branches or instructions. The main original features of the test data generation technology are the following: (1) path predicates are expressed in the bit-vector theory, and a novel CLP-based approach has been developped to solve such constraints; (2) the concolic paradigm has been adapted to unstructured control-flow, and it appears to be both a very powerful and easy to implement mean of recovering a high-level view (CFG) of the program. (3) specific heuristics have been designed to discard *a priori* paths redundant with the current coverage objective.

– Jaut (Java Automatic Unit Testing) [4] is a test data generator for programs in bytecode Java. Given a bytecode instruction of the method under test, it aims to find an input memory state (values of the parameters and of the instances in the heap) to cover this instruction. Repeating this process, the structural coverage criterion of all-the-statements can be fulfilled. In Jaut, bytecode instructions are modelled with constraints, as well as the conditions that permit to reach the goal. An input memory state to cover the goal instruction can so be found by constraint solving. The tool currently deals with arithmetic operations on integers, heap manipulation and conditional instructions. The implementation of a strategy to avoid enumerating all the paths that lead to the goal until finding an executable one is in progress. Current work also includes dealing with polymorphic function calls.
– Gatel [12] is a test environment for synchronous models of reactive systems described in Lustre/SCADE. The core of the tool is a CLP interpretation of the Lustre language, together with a resolution procedure dedicated to the underlying linear temporal logic. This framework allows to automate a wide range of usual testing activities. Initially designed for generating test sequences according to a test objective, Gatel also addresses symbolic simulation, model debug, conformance checking, coverage analysis/completion, test suite evaluation. Current work is twofold: updating the CLP interpretation and procedure to the latest version of the SCADE Suite (enriched with built-in state machines), upgrading the procedure with powerful abstractions to scale up to real-life industrial models.
– JML-Testing-Tools is an automated animation and test generation tool based on JML annotations [3]. The animation feature is used to simulate the execution of the model, using constraint solving techniques, in order to ensure the conformance of the model behaviors w.r.t. the informal requirements. The test generation part works by first computing boundary test targets, satisfying JML preconditions of the Java methods, according to specific object-oriented data coverage criteria, such as null pointers, aliasings, etc. coupled with a boundary analysis of numerical values. It then builds complete execution sequences, in terms of method invocations, using the symbolic animation of the model, in order to cover these targets, thus producing the test cases.

3 Perspectives

Scalability is the main challenge that the tools presented here have to face to. Dealing with more than hundred of thousands lines of code, with dynamic constructions such as huge dynamic data structures, with non-linear numerical constraints extracted from complex statements are some of the problems we have to deal with. Research works were launched to address these problems in all the research teams mentionned in this paper. Next step will be the dissemination of the *Constraint Programming technology in Program Verification* to Industry.

References

1. Sebastien Bardin and Philippe Herrmann. Structural testing of executables. In *1th Int. Conf. on Software Testing, Verification and Validation (ICST'08)*, pages 22–31, 2008.
2. B. Botella, A. Gotlieb, and C. Michel. Symbolic execution of floating-point computations. *The Software Testing, Verification and Reliability journal*, 16(2):pp 97–121, June 2006.
3. F. Bouquet, F. Dadeau, and B. Legeard. Automated boundary test generation from JML specifications. In *FM'06, 14th Int. Conf. on Formal Methods*, volume 4085 of *LNCS*, pages 428–443, Hamilton, Canada, August 2006. Springer-Verlag.
4. F. Charreteur and A. Gotlieb. Raisonnement contraintes pour le test de bytecode java. In *quatrimes Journes Francophones de Programmation par Contraintes (JFPC'08)*, pages 11–20, Nantes, France, Juin 2008.
5. H. Collavizza, M. Rueher, and P. Van Hentenryck. Cpbpv: A constraint-programming framework for bounded program verification. In *Proc. of CP2008*, LNCS 5202, pages 327–341, 2008.
6. J. Dick and A. Faivre. Automating the generation and sequencing of test cases from model-based specifications. In *Proc. of the FME'03: Industrial Strength Formal Methods*, LNCS 670, 1993.
7. Cormac Flanagan. Automatic software model checking via constraint logic. *Sci. Comput. Program.*, 50(1-3):253–270, 2004.
8. A. Gotlieb. Euclide: A constraint-based testing platform for critical c programs. In *2th International Conference on Software Testing, Validation and Verification (ICST'09)*, Denver, CO, Apr. 2009.
9. A. Gotlieb, B. Botella, and M. Rueher. A clp framework for computing structural test data. In *Proceedings of Computational Logic (CL'2000)*, LNAI 1891, pages 399–413, London, UK, July 2000.
10. A. Gotlieb, T. Denmat, and B. Botella. Goal-oriented test data generation for pointer programs. *Information and Software Technology*, 49(9-10):1030–1044, Sep. 2007.
11. Bruno Marre. Toward Automatic Test Data Set Selection using Algebraic Specifications and Logic Programming. In Koichi Furukawa, editor, *Proc. of the Eight ICLP*, pages 202–219, Paris, Jun. 1991. MIT Press.
12. Bruno Marre and Benjamin Blanc. Test selection strategies for lustre descriptions in gatel. *Electronic Notes in Theoretical Computer Science*, 111:93 – 111, 2005.
13. Andreas Podelski. Model checking as constraint solving. In Jens Palsberg, editor, *Proceedings of SAS: Static Analysis Symposium*, volume 1824 of *LNCS*, pages 22–37. Springer-Verlag, 2000.
14. N. Williams, B. Marre, P. Mouy, and M. Roger. Pathcrawler: Automatic generation of path tests by combining static and dynamic analysis. In *In Proc. Dependable Computing - EDCC'05*, pages 281–292, 2005.