C.OPEN and ANNOTATOR: Tools for On-the-Fly Model Checking C Programs *

María del Mar Gallardo[1], Christophe Joubert[2], Pedro Merino[1], and David Sanán[1]

[1] University of Málaga, Campus de Teatinos s/n, 29071, Málaga, Spain
{gallardo,pedro,sanan}@lcc.uma.es
[2] Technical University of Valencia, Camino de Vera s/n, 46022, Valencia, Spain
joubert@dsic.upv.es

Abstract. This paper describes a set of verification components that open the way to perform on-the-fly software model checking with the CADP toolbox, originally designed for verifying the functional correctness of LOTOS specifications. Two new tools (named C.OPEN and ANNOTATOR) have been added to the toolbox. The approach taken fits well within the existing architecture of CADP which doesn't need to be altered to enable C program verification.

1 Introduction

The software model checking problem consists in verifying that a program, *i.e.* an infinite state system described in a high-level language, does not contain errors, such as improper memory access, misuse of system interfaces, or violation of (temporal logic) properties. The verification process is automatic, and the wrong conception of the program is eventually illustrated by means of potential offending behaviors of the system (*e.g.*, counter examples). Traditionally, programs are first analysed to statically remove parts that do not affect the property of interest, using light-weight pre-processing technique such as *program slicing*. The reduced model is then abstracted using *predicate abstraction* and *localization* techniques. Finally, the resulting finite state system is processed by SAT-based, BDD-based or explicit state model checkers.

Existing software model checkers, like SLAM [1] and BLAST [2], are either domain specific (*e.g.*, verification of drivers), language dependent, or based on dedicated algorithms and tools. This paper presents an analysis engine that finds application programming interface (API) usage errors in C programs, similarly to BLAST and SLAM, but rather focusing on a general purpose model checking framework. We describe a set of components, namely C.OPEN and ANNOTATOR, that enable the explicit state verification of C programs by means of the last stable CADP 2006 "Edinburgh" release. The CADP toolbox[3] [3] is a complex software suite integrating numerous verification tools. CADP supports the process algebra LOTOS for specification, and offers various tools for simulation and

* This work has been supported by the Spanish MEC under grant TIN2004-7943-C04
[3] CADP web site: "http://www.inrialpes.fr/vasy/cadp".

formal verification, including equivalence checkers (bisimulations) and model checkers (temporal logics and modal μ-calculus). The toolbox is designed as an open platform for the integration of other specification, verification and analysis techniques. This is realized by means of APIs which on different levels provide means to extend or exploit the functionalities of the toolbox. These APIs have been used by others to link CADP to other specification languages as well as other verification/testing tools. Here we describe how these APIs have been used by C.OPEN to support C program transformation and abstraction based on XML intermediate representation, and by ANNOTATOR to support on-the-fly data flow analysis and program slicing, namely *influence analysis*, of implicit control flow graphs using *boolean equation systems* (BESs). Our efforts have been driven by the intention to avoid changes to the existing components as much as possible, while providing a sound and efficient framework for C program model checking.

Originality. Our approach differs from previous works, like BLAST and SLAM, in several ways:

1. in the first attempt to connect to the CADP toolbox a model generator (C.OPEN) that automatically extracts implicit *labeled transition systems* (LTSs) from programs written in C programming language, and a static analyzer (ANNOTATOR) that works on implicit abstracted *control flow graphs* (CFGs) described as LTSs,
2. in our emphasis on the verification of distributed protocols (*e.g.*, the Peterson's mutual exclusion (PME) protocol between two processes), using well-specified APIs [4], described as multiple (or multi-instantiated) concurrent independent C programs, rather than on sequential (SLAM) or multi-threaded programs (ongoing work of BLAST),
3. in our use of the OPEN/CÆSAR modular architecture and XML, BES and LTS technologies to represent the state-space and verification problem efficiently and to facilitate the connection to other programming languages, like Promela, and
4. in the way we concentrated this research work on the compiler side, similarly to BANDERA and BOGOR [5] model checking frameworks, but using well-established verification tools of the CADP toolbox as back-end.

2 Software Architecture

The toolset encompasses two sorts of tools (see Figure 1) to verify C programs generated via CADP. (i) The C.OPEN tool provides different means to distill an implicit LTS from a C program. (ii) The static analyser ANNOTATOR enables on-the-fly data flow and influence analysis of implicit LTSs describing abstract CFGs.

Distilling implicit LTSs from a C program. C.OPEN [6] is an addon component for CADP to support C program input to the OPEN/CÆSAR environment [7], though we state that the XML API, called PIXL [8], on which the tool is based,

is general enough to attach the C program abstraction process to other verification toolboxes, such as SPIN, via Promela specifications instead of implicit LTSs [4]. The idea that OPEN/CÆSAR environment can be connected to a C compiler and that existing CADP tools can thereby be extended to this new class of specifications is an important step towards re-using well-established verification toolboxes. C.OPEN (400 lines of Shell script) takes as inputs a system

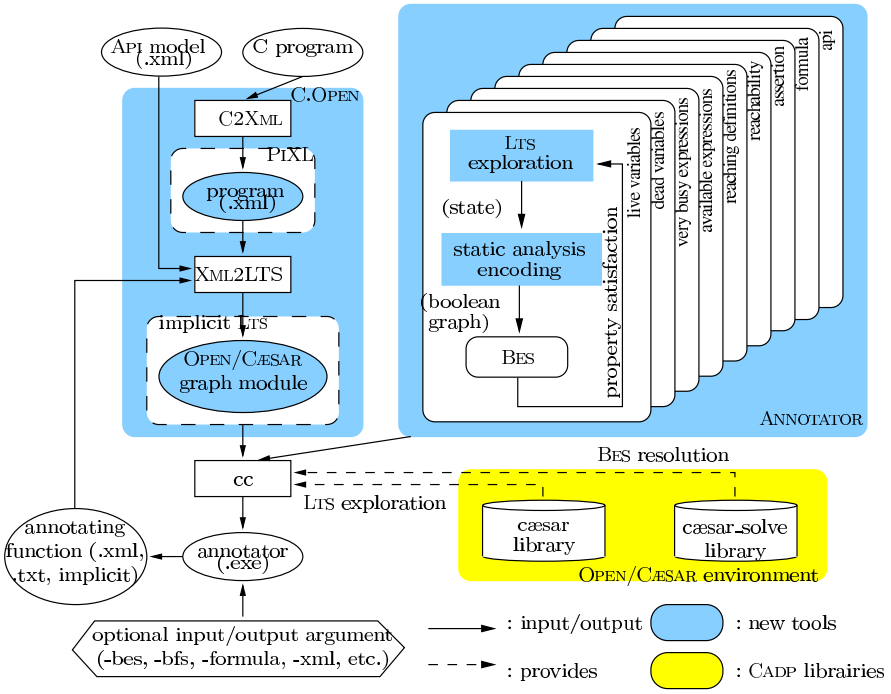

Fig. 1. C.OPEN and ANNOTATOR tools

described by a set of C programs, an operating system API's model represented in XML, and an OPEN/CÆSAR application (*e.g.*, ANNOTATOR). As an output, it generates an executable application (*e.g.*, annotator.exe) by performing the required sequence of tool invocations: 1) a tool, called C2XML (2 000 lines of JAVA), is used with JAVACC and a C grammar (1 000 lines of JAVA), to translate C programs into PIXL compliant XML models; 2) another tool, called XML2LTS (4 500 lines of JAVA), then slices the program models with respect to system APIs to be preserved [9] and it constructs the OPEN/CÆSAR graph module describing the implicit program LTS; 3) finally, the C compiler *cc* is called.

C.OPEN allows to construct abstracted state spaces on-the-fly, and only to the required precision (*w.r.t.* a specific API). It currently offers the possibility to generate either CFG or explicit state space of a program as an implicit LTS.

Analysing implicit CFGs. ANNOTATOR implements standard data flow analysis algorithms on a CFG, by using boolean equation systems (BESs) [10, 11]. It

also computes various influence analyses [12], generally used for compacting the program state representation, by detecting the relevant program variables in each control point, for a property of interest.

Our static analyser takes as inputs a static analysis to carry out and an LTS describing the CFG of a program, in which instructions are abstracted to the strict necessary information (*i.e.*, modified and defined variables, used expressions, and instruction type). This LTS is represented implicitly by its successor function as an OPEN/CÆSAR program provided by compliant compilers, such as C.OPEN, but existing CADP compilers, such as CÆSAR, could be directly extended to provide such CFGS [13].

ANNOTATOR (6 000 lines of C code) consists of several modules, each one containing the BES translation for a particular static analysis (live variables, very busy expressions, available expressions, reaching definitions, reachability, assertion control, formula and API preservation influence analyses). BESS are represented implicitly by their successor function, in the same way as LTSS in OPEN/CÆSAR. They are handled internally by the CÆSAR_SOLVE [14] library, which offers several on-the-fly resolution algorithms, based on different search strategies (*e.g.*, breadth-first). Dependent on the option selected by the user, the analysis result is written to an XML or textual file. These formats allow post-processing of computed analyses, by directly conveying the result as input to compilers reading these formats, such as C.OPEN, allowing further compilation optimizations.

Availability. The proposed tools are publicly available through the following web pages http://www.lcc.uma.es/gisum/tools/smc. C.OPEN and ANNOTATOR, being part of the database of research tools developed using CADP, are also referenced by the CADP web site. Both new tools are rather small, robust and mature (in operation for about a year) and detailed manual pages are provided, as well as more than 25 program examples and step-by-step small case studies. More recently [15], we also defined web services that allow the remote use of C.OPEN and ANNOTATOR, as well as most of CADP verification tools, through the FMICS-JETI platform [16] from a JABC client [17].

Applicability. Concerning applicability, C.OPEN compiles concurrent C programs into the OPEN/CÆSAR intermediate format (*i.e.*, implicit labeled transition system (LTS)), to which efficient CADP model checkers, such as EVALUATOR (evaluation of regular alternation-free μ-calculus formulas) and BISIMULATOR (equivalence checking), are connected. Hence, CTL, ACTL, PDL, PDL-Δ and regular alternation-free μ-calculus properties can be verified on our C input programs. In the PME demonstration, we successfully checked respectively one safety, liveness and fairness property on the C implementation of the protocol and we also reduced the explicit-state space size by 20% using API influence analysis results computed by the ANNOTATOR tool. Furthermore, all analyses that are available in the CADP toolbox can be directly used on our C input programs. Unfortunately, we could not compare our verification framework with

well-established software model checkers, like BLAST or SLAM, since they are not dealing with distributed protocols with well-defined APIs yet.

Scalability. ANNOTATOR has been successfully experimented on very large CFGs, extracted from the VLTS benchmark[4], with size up to 10^6 program counters and instructions. Moreover, C.OPEN and ANNOTATOR allow several levels of abstraction of the program instructions present in the LTS model, giving the possibility to verify further properties or to achieve further reductions on the program model.

3 Conclusion and Future Work

The development of an on-the-fly software model checker "from scratch" is a complex and costly task. The open modular architecture adopted for C.OPEN and ANNOTATOR aims at making this process easier, by using the XML intermediate representation, the well-established verification framework of BESs, together with the generic libraries for LTS exploration and BES resolution provided by CADP. For instance, this tool architecture reduces the effort of implementing a new static analysis to its strict minimum: encoding the mathematical definition of the analysis as a BES, and interpreting the result. We plan to continue our work by extending ANNOTATOR with other static analyses (e.g., reset variables analysis [13]) and by interconnecting the two new CADP components with tools extending SPIN, such as SOCKETMC [4] and αSPIN [18].

References

1. Ball, T., Rajamani, S.K.: The slam toolkit. In Berry, G., Comon, H., Finkel, A., eds.: Proceedings of the 13th International Conference on Computer Aided Verification CAV'2001 (Paris, France). Volume 2102 of Lecture Notes in Computer Science., Springer Verlag (2001) 260–264
2. Beyer, D., Henzinger, T.A., Théoduloz, G.: Lazy shape analysis. In Ball, T., Jones, R.B., eds.: Proceedings of the 18th International Conference on Computer Aided Verification CAV'2006 (Seattle, WA, USA). Volume 4144 of Lecture Notes in Computer Science., Springer Verlag (2006) 532–546
3. Garavel, H., Lang, F., Mateescu, R.: An overview of CADP 2001. European Association for Software Science and Technology (EASST) Newsletter **4** (2002) 13–24 Also available as INRIA Technical Report RT-0254 (December 2001).
4. Camara, P., Gallardo, M., Merino, P., Sanán, D.: Model checking software with well-defined apis: the socket case. In Gnesi, S., Margaria, T., Massink, M., eds.: Proceedings of the 10th International Workshop on Formal Methods for Industrial Critical Systems FMICS'2005 (Lisbon, Portugal), ACM-SIGSOFT (2005) 17–26
5. Robby, Rodríguez, E., Dwyer, M.B., Hatcliff, J.: Checking JML specifications using an extensible software model checking framework. Springer International Journal on Software Tools for Technology Transfer (STTT) **8** (2006) 280–299

[4] VLTS web site: http://www.inrialpes.fr/vasy/cadp/resources/benchmark_bcg.html

6. Gallardo, M., Merino, P., Sanán, D.: Towards model checking c code with open/cæsar. In Barjis, J., Ultes-Nitsche, U., Augusto, J.C., eds.: Proceedings of the 4th International Workshop on Modelling, Simulation, Verification and Validation of Enterprise Information Systems MSVVEIS'2006 (Paphos, Cyprus), Insticc Press (2006) 198–201
7. Garavel, H.: Open/cæsar: An open software architecture for verification, simulation, and testing. In Steffen, B., ed.: Proceedings of the First International Conference on Tools and Algorithms for the Construction and Analysis of Systems TACAS'98 (Lisbon, Portugal). Volume 1384 of Lecture Notes in Computer Science., Berlin, Springer Verlag (1998) 68–84 Full version available as INRIA Research Report RR-3352.
8. Gallardo, M., Martínez, J., Merino, P., Nuñez, P., Pimentel, E.: Pixl: Applying xml standards to support the integration of analysis tools for protocols. Science of Computer Programming (2006)
9. Gallardo, M., Joubert, C., Merino, P., Sanán, D.: On-the-fly API influence analysis of software. In Merino, P., Bakkali, M., eds.: Proceedings of the 2nd International Conference on Science and Technology JICT'07 (Málaga, Spain), Spicum (2007)
10. Gallardo, M., Joubert, C., Merino, P.: On-the-fly data flow analysis based on verification technology. In Drechsler, R., Glesner, S., Knoop, J., eds.: Proceedings of the 6th International Workshop on Compiler Optimization meets Compiler Verification COCV'2007 (Braga, Portugal). Electronic Notes in Theoretical Computer Science, Elsevier (2007) To appear.
11. Gallardo, M., Joubert, C., Merino, P.: Implementing influence analysis using parameterised boolean equation systems. In: Proceedings of the 2nd International Symposium on Leveraging Applications of Formal Methods, Verification and Validation ISOLA'06 (Paphos, Cyprus), IEEE Computer Society Press (2006) To appear.
12. Cámara, P., Gallardo, M., Merino, P.: Abstract matching for software model checking. In Valmari, A., ed.: Proceedings of the 13th International SPIN Workshop on Model Checking of Software SPIN'06 (Vienna, Austria). Volume 3925 of Lecture Notes in Computer Science., Springer Verlag (2006) 182–200
13. Garavel, H., Serwe, W.: State space reduction for process algebra specifications. Theoretical Computer Science **351**(2) (2006) 131–145
14. Mateescu, R.: Caesar_solve: A generic library for on-the-fly resolution of alternation-free boolean equation systems. Springer International Journal on Software Tools for Technology Transfer (STTT) **8** (2006) 37–56
15. Gallardo, M., Joubert, C., Merino, P., Sanán, D.: On-the-fly model checking for C programs with extended CADP in FMICS-jETI. In: Proceedings of the 12th IEEE International Conference on Engineering of Complex Computer Systems ICECCS'07 (Auckland, New Zealand), IEEE Computer Society Press (2007) To appear.
16. Margaria, T., Steffen, B.: Advances in the FMICS-jETI platform for program verification. In: Proceedings of the 12th IEEE International Conference on Engineering of Complex Computer Systems ICECCS'07 (Auckland, New Zealand), IEEE Computer Society Press (2007) To appear.
17. Margaria, T., Nagel, R., Steffen, B.: Remote integration and coordination of verification tools in jETI. In: Proceedings of the 12th IEEE International Conference on the Engineering of Computer-Based Systems ECBS'05 (Greenbelt, MD, USA), IEEE Computer Society Press (2005) 431–436
18. Gallardo, M., Martinez, J., Merino, P., Pimentel, E.: αspin: A tool for abstraction in model checking. Software Tools for Technology Transfer **5**(2-3) (2004) 165–184