On-the-Fly Verification using CADP

Radu Mateescu
INRIA Rhône-Alpes / VASY, 655, avenue de l'Europe
F-38330 Montbonnot Saint Martin, France
Radu.Mateescu@inria.fr

Abstract

On-the-fly verification consists in analysing the correctness of a finite-state concurrent system by constructing and exploring its state space incrementally. This provides a way to fight against state explosion, by enabling the detection of errors in systems with large state spaces. We give an overview of the latest on-the-fly verification techniques developed within the CADP protocol engineering toolbox. These techniques are based upon on-the-fly resolution of (alternation-free) boolean equation systems, implemented in a generic software library named CÆSAR_SOLVE, which is currently used for equivalence checking and model checking.

1 Introduction

A successful approach for automatically analysing the correctness of a concurrent finite-state system consists in building its state space and verifying that it meets an abstract specification, expressed either as an external behaviour (*equivalence checking*), or as a set of temporal logic properties (*model checking*). The main drawback of this approach is *state explosion* (prohibitive size of the state space), which occurs for systems containing many parallel processes and complex data structures. *On-the-fly* verification combats state explosion by constructing the state space in a demand-driven way, therefore enabling the detection of errors in large systems.

Building robust tools for on-the-fly verification is a complex and costly activity. This effort could be significantly reduced by using generic software components from which elaborated verification tools can be constructed. A successful attempt in achieving genericity was made by developing the OPEN/CÆSAR [5] environment, which is a central piece of the CADP verification toolbox [6]. OPEN/CÆSAR precisely defines an application programming interface (API) which represents state spaces or labelled transition systems (LTSs) in an *implicit* manner, by means of their "successor" function. This API allows to build generic verification tools that explore the LTS of a concurrent program on-the-fly, independently from the source language in which the program was written. Such verification tools are directly available for any language equipped with a compiler implementing a translation to the API of OPEN/CÆSAR. CADP currently contains several verification tools built using

Open/Cæsar: Executor (random execution), Exhibitor (searching of regular transition sequences), Ocis (interactive and guided simulation), Evaluator (model checking of regular alternation-free μ-calculus), etc.

Continuing the ideas underlying Open/Cæsar, we developed a generic software library, called Cæsar_Solve, dedicated to on-the-fly resolution of alternation-free boolean equation systems (Bess), represented by their corresponding boolean graphs encoded implicitly in an Open/Cæsar-style. The library currently contains four Bes resolution algorithms, each one implementing a different strategy, and also allows to generate *diagnostics*, i.e., portions of the Bes explaining the truth value of a variable. Cæsar_Solve has been used — in conjunction with Open/Cæsar — for building two on-the-fly verification tools: the equivalence checker Bisimulator and the model checker Evaluator.

The paper is organized as follows. Section 2 recalls the Bes definition and briefly describes the four on-the-fly resolution algorithms. Section 3 presents the principles of the generic library Cæsar_Solve and its applications to equivalence checking and μ-calculus model checking. Section 4 concludes and indicates directions for future work.

2 Boolean equation systems

A boolean equation system (Bes) consists of several blocks of equations having boolean variables in their left-hand sides and disjunctive or conjunctive boolean formulas in their right-hand sides (see Figure 1). Each block of n equations denotes either a minimal (μ) or maximal (ν) fixed point solution of its associated functional defined over $\mathbf{Bool}^n$. Here we consider *alternation-free* Bess (i.e., without cyclic dependencies between blocks), which can be solved in linear time and thus are of practical interest.

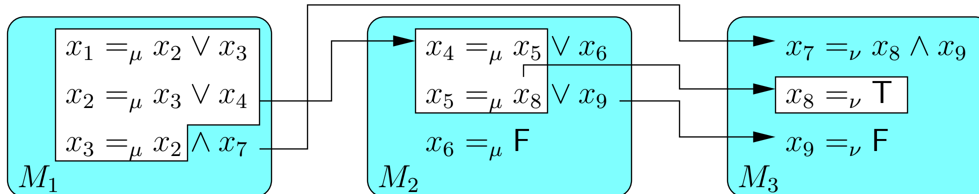

Figure 1: An alternation-free boolean equation system

The *local* (or *on-the-fly*) resolution of a Bes consists in computing the value of a given variable, without necessarily solving the whole system. A useful local resolution scheme for alternation-free Bess can be obtained by associating a resolution routine to each block of the Bes; a routine is invoked whenever the value of a variable defined in its corresponding block is needed. This scheme yields simple algorithms, which must handle a single type of fixed point equations, and also allows to optimize the resolution of each block independently, according to its particular shape.

We developed the resolution algorithms by representing Bess as *boolean graphs* [1], which provide an intuitive way of reasoning about dependencies between boolean variables. All algorithms are based upon the same principle: forward exploration of the boolean graph

starting at the variable of interest, and backward propagation (i.e., substitution) of variables whose values have been established. During the computation, the algorithms also gather information for constructing diagnostics (portions of the boolean graph illustrating the value of a variable) after termination of the resolution, using the approach proposed in [7]. We developed four local resolution algorithms [9]: A1 is depth-first search (Dfs)-based and currently serves as engine in the Evaluator model checker for regular alternation-free μ-calculus [10]; A2 is breadth-first search (Bfs)-based and yields low-depth diagnostics; A3 and A4 are both Dfs-based and exhibit reduced memory consumption for acyclic and disjunctive/conjunctive boolean graphs, respectively. A3 is currently used within Cadp for checking μ-calculus properties on large execution traces [8].

3 On-the-fly verification

Alternation-free Bess allow to encode various verification problems on Ltss, such as equivalence checking, i.e., comparison of two Ltss modulo strong and weak equivalences [2] and model checking, i.e., evaluation of alternation-free μ-calculus formulas on an Lts [1]. This motivated the construction of a generic library within Cadp, called Cæsar_Solve, which implements the four on-the-fly resolution algorithms for alternation-free Bess (see Section 2) and can be applied for building different verification tools. Genericity is achieved by encoding Bess and diagnostics as boolean graphs, through a precise Api inspired by the Open/Cæsar [5] Api for representing implicit Ltss. The basic idea is to encode each equation block of a Bes by the successor function of its corresponding boolean graph; after resolution of a boolean variable, the library can also provide a diagnostic (boolean subgraph rooted at the variable) by giving its successor function.

To use the Cæsar_Solve library, the designer of a verification tool must provide: (1) an implementation in C of the boolean variables as pointers to (application-dependent) memory areas of fixed size, equipped with comparison and hashing functions; (2) an iterator function which enumerates the successors of a given boolean variable; (3) if needed, a function which traverses the boolean subgraph provided as diagnostic in order to interpret it in terms of the application. So far we used Cæsar_Solve as engine for two on-the-fly verification tools with diagnostic developed within Cadp (see Figure 2): the equivalence checker Bisimulator, which handles strong, branching, observational, safety, and $\tau^*.a$ equivalence; and the model checker Evaluator, which handles alternation-free μ-calculus formulas extended with regular expressions and data parameters. Both tools translate their verification problems into Bes resolution and interpret diagnostics by traversing the corresponding boolean subgraphs using the successor function given as output by Cæsar_Solve.

We also identified several particular cases of practical interest where the memory-efficient algorithms A3 and A4 can be used [9]. For equivalence checking, A3 can be applied when one of the Ltss is acyclic (e.g., when checking the inclusion of an execution trace in the Lts of a specification) and A4 can be applied when one Lts is deterministic (e.g., when comparing a protocol Lts with its deterministic service Lts). For model checking, A3 can be applied when the Lts is acyclic (e.g., when checking formulas on execution traces) and A4 can be applied when the formulas are expressed using the operators of Actl [3] and Pdl [4].

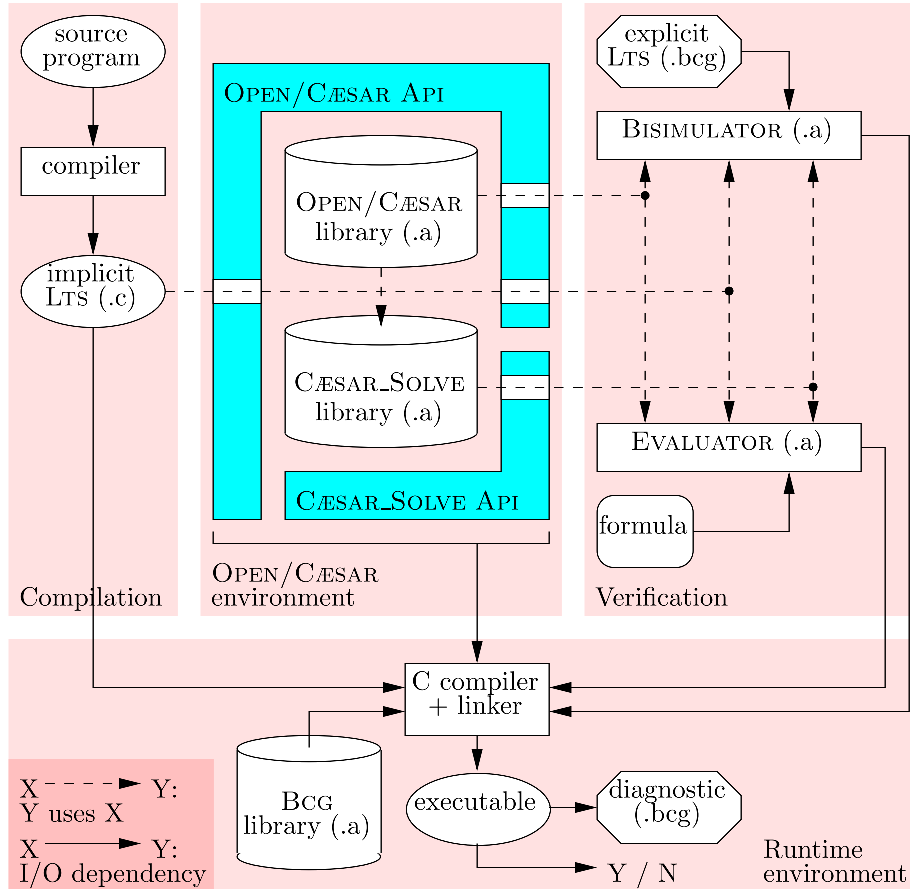

Figure 2: Use of OPEN/CÆSAR and CÆSAR_SOLVE for on-the-fly verification

4 Conclusion and future work

The CÆSAR_SOLVE library is intended to provide a generic support for the construction of on-the-fly verification tools based upon BES resolution. Two directions for continuing this work seem promising. Firstly, to increase the flexibility of CÆSAR_SOLVE, we plan to extend it with new BES resolution algorithms, which either optimize particular cases (e.g., model checking of ACTL and PDL formulas on execution traces, which yields disjunctive/conjunctive BESs with low-breadth, acyclic boolean graphs), or experiment with various exploration strategies (e.g., combined DFS-BFS, random exploration, etc.). Secondly, we envisage new applications of CÆSAR_SOLVE in the field of verification: partial-order reduction by detecting τ-confluent transitions (a prototype tool has already been developed and used for compositional verification [11]); test generation and discrete controller synthesis, both of which could be rephrased in terms of diagnostic generation.

References

[1] Andersen, H.R., *Model Checking and Boolean Graphs*, Theoretical Computer Science 126(1):3–30, 1994.

[2] Cleaveland, R., and B. Steffen, Computing Behavioural Relations, Logically, in *ICALP'91*, LNCS vol. 510, pp. 127–138.

[3] De Nicola, R., and F. Vaandrager, Action versus State based Logics for Transition Systems, in *Semantics of Concurrency*, LNCS vol. 469, pp. 407–419.

[4] Fischer, M. J., and R. E. Ladner, *Propositional Dynamic Logic of Regular Programs*, Journal of Computer and System Sciences 18:194–211, 1979.

[5] Garavel, H., OPEN/CÆSAR: An Open Software Architecture for Verification, Simulation, and Testing, in *TACAS'98*, LNCS vol. 1384, pp. 68–84.

[6] Garavel, H., F. Lang, and R. Mateescu, An Overview of CADP 2001, *EASST Newsletter* 4:13–24, 2002.

[7] Mateescu, R., Efficient Diagnostic Generation for Boolean Equation Systems, in *TACAS'00*, LNCS vol. 1785, pp. 251–265.

[8] Mateescu, R., Local Model-Checking of Modal Mu-Calculus on Acyclic Labeled Transition Systems, in *TACAS'02*, LNCS vol. 2280, pp. 281–295.

[9] Mateescu, R., A Generic On-the-Fly Solver for Alternation-Free Boolean Equation Systems, in *TACAS'03*, LNCS vol. 2619, pp. 81–96.

[10] Mateescu, R., and M. Sighireanu, *Efficient On-the-Fly Model-Checking for Regular Alternation-Free Mu-Calculus*, Science of Computer Programming 46(3):255–281, 2003.

[11] Pace, G., F. Lang, and R. Mateescu, Calculating Tau-Confluence Compositionally, in *CAV'03*, to appear in LNCS, 2003.