DISTRIBUTOR and BCG_MERGE: Tools for Distributed Explicit State Space Generation

Hubert Garavel, Radu Mateescu, Damien Bergamini, Adrian Curic, Nicolas Descoubes, Christophe Joubert, Irina Smarandache-Sturm, and Gilles Stragier

INRIA Rhône-Alpes / VASY
655, avenue de l'Europe, F-38330 Montbonnot St Martin, France

1 Introduction

The explicit-state verification of complex concurrent systems, whose underlying state spaces may be prohibitively large, requires an important amount of memory and computation time. Although explicit state space generation is known to be exponential as the number of concurrent processes in the system increases, it is tempting to push forward the capabilities of verification tools by exploiting the computing resources (memory and processors) of massively parallel machines, such as clusters and grids.

Several distributed algorithms have been proposed for analyzing stochastic Petri nets and Markov chains (e.g., by Nicol and Ciardo, by Haverkort, Bell, and Bohnenkamp, etc.), as well as for model checking (e.g., by Stern and Dill, by Lerda and Sisto, etc.). Our own distributed algorithms [3] allow the construction of Labelled Transition Systems (LTSs) using several machines connected by a network. These algorithms are implemented in the DISTRIBUTOR and BCG_MERGE tools using the facilities of the CADP [2] verification toolbox. In a nutshell, each machine used by DISTRIBUTOR is responsible for generating and storing a fragment of the entire LTS. Upon termination of the distributed state space generation, all these fragments are combined together using BCG_MERGE to obtain the entire LTS.

Between 2000 and 2005, we developed three successive versions (1.0, 2.0, and 3.0) of DISTRIBUTOR and BCG_MERGE. This led to significant functionality improvements. For instance, version 3.0 of DISTRIBUTOR can also reduce LTSs on-the-fly, by applying τ-compression (elimination of τ-cycles denoting divergence) or τ-confluence (a form of partial order reduction preserving branching equivalence) [4] using the algorithms proposed in [6]. However, besides the distributed algorithms themselves, we realized that it was also essential to pay attention to often-neglected practical issues, such as software architecture concepts and user-oriented features pertaining to ergonomy, and this is what the present paper is about.

2 Software Architecture and User-Oriented Features

Source language independence. Developing verification tools for sequential machines is a demanding, long-term effort. But the development effort is

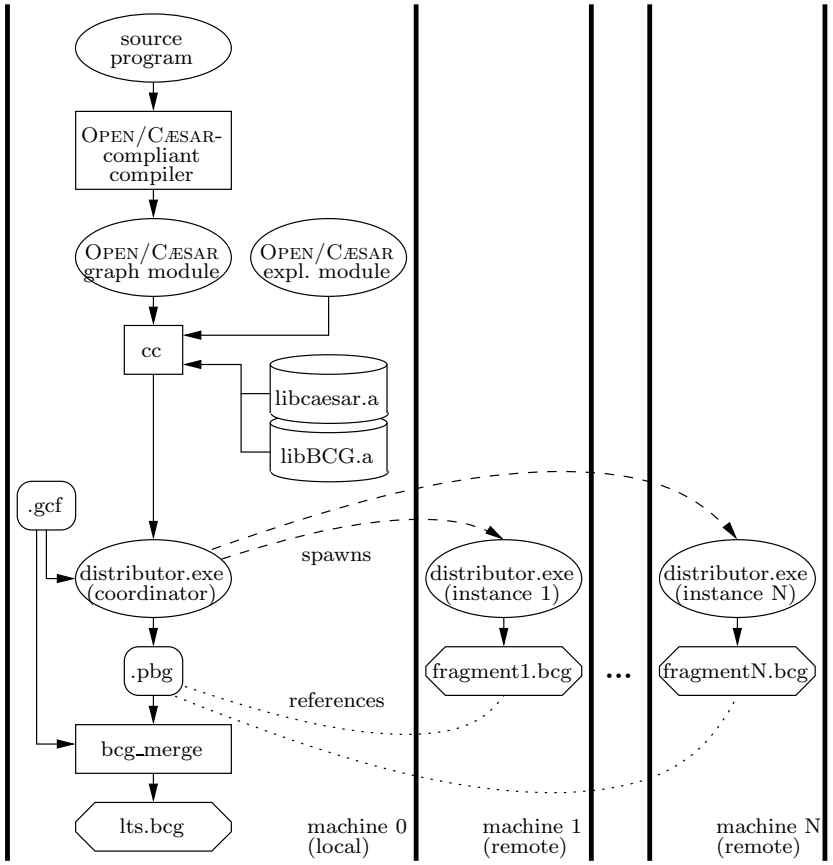
even higher for verification tools intended to work on parallel machines. Therefore, it is desirable to design tools that can support multiple input languages instead of a single one. For this reason, DISTRIBUTOR is implemented using the OPEN/CÆSAR generic environment [1], which is the basis for all on-the-fly verification algorithms implemented within CADP. OPEN/CÆSAR offers an *implicit* representation for LTSs to be explored on-the-fly by providing a language-independent API that basically defines the states, labels, and transitions of the LTS, together with functions for comparing, hashing, accessing the initial state, and computing the successors of a given state. It also provides C libraries containing a rich set of LTS exploration primitives (transition lists, stacks, tables, etc.). Thus, DISTRIBUTOR can be used for any input language equipped with a compiler supporting the OPEN/CÆSAR API.

Platform independence. A key design goal of DISTRIBUTOR and BCG_MERGE is to allow them to run on the largest possible number of computing platforms (networks of workstations, clusters of PCs, or even laptops, etc.) For this reason, DISTRIBUTOR and BCG_MERGE only use the features present in mainstream operating systems and do not to rely on any dedicated middleware (such as MPI, etc.) that is not installed by default. Similarly, they do not assume the existence of a common file system (e.g., NFS, SAMBA, etc.) shared between machines.

Separation between algorithms and communications. Starting from version 2.0 of DISTRIBUTOR and BCG_MERGE, a clear separation was established between, on the one hand, the distributed algorithms themselves and, on the other hand, the primitives used for communication between machines. Such primitives are encapsulated into a dedicated library (named CÆSAR_NETWORK), which provides functionalities such as blocking and non-blocking buffered send/receive. Following the "platform independence" requirement, CÆSAR_NETWORK only requires ordinary TCP sockets and standard remote connection protocols (e.g., rsh/rcp, ssh/scp, etc.) to be available. CÆSAR_NETWORK is also used by other VASY tools for distributed equivalence checking [5] and distributed model checking.

Instances on local and remote machines. The machine on which DISTRIBUTOR and BCG_MERGE are launched by the end-user is called the *local* machine, all the other ones being called *remote* machines. DISTRIBUTOR and BCG_MERGE work by launching distributed processes, called *instances*. Each instance corresponds to a pair (M, D), indicating that a distributed process will be launched on machine M and will store its files in directory D (the *working directory* of the instance) located on some filesystem of M. Several instances with different working directories may execute on the same machine. A working directory may be either local to its machine, or shared between several machines.

Description of network resources. To specify the list of machines and instances involved in the distributed computation, the CÆSAR_NETWORK library uses a dedicated file named *Grid Configuration File* (GCF), whose format is defined in [8]. This file also specifies the various configuration parameters to be used for launching instances and connecting machines: TCP port number(s) used by sockets, remote connection protocol (rsh, ssh, etc.), remote copy protocol



(rcp, scp, etc.), login name(s) used for remote authentication, size of communication buffers, pathname of the CADP installation directory, connection timeout, pathname of the working directory, and list of files to be copied in the working directories upon launching.

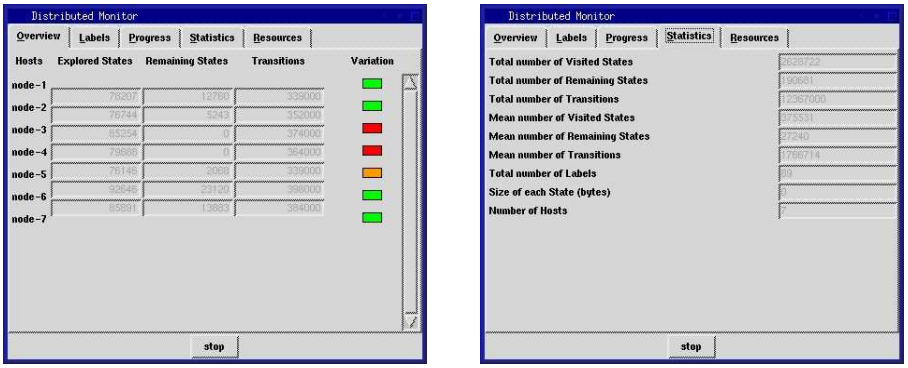
Partitioned Labelled Transition Systems. The result of the distributed LTS generation performed by DISTRIBUTOR is a theoretical model defined in [3] and called *Partitioned* LTS (PLTS for short). A PLTS is a collection of *fragments* (one per instance), each fragment containing a subset of the states and transitions of the entire LTS to be generated. Taken altogether, the fragments form a partition of the entire LTS. Taken individually, each fragment can be seen as an LTS, with the important difference that it may be a disconnected graph, which is never the case with an LTS generated from a "meaningful" specification. The role of BCG_MERGE is to take a PLTS and merge all its fragments into one single LTS.

To represent LTSs, as well as fragments, we use the BCG (*Binary-Coded Graphs*) format of CADP. This format provides an *explicit* representation for LTSs given by their states, labels, and transitions. It allows to store LTSs in

compact, binary files and is equipped with a set of C libraries and tools providing a wide range of functionalities (reading and writing, exploring the transition relation, converting from/to other LTS formats, visualizing graphically, etc.). A collection of BCG files is available on-line in the VLTS benchmark suite [9], which aims at providing realistic examples of LTSs for the assessment of verification and graph manipulation tools. To represent PLTSs, we developed (together with CWI in the framework of the international SENVA research team) a dedicated format named PBG (*Partitioned* BCG *Graph*). A PBG is a text file containing references to a GCF file and to a collection of fragments stored as BCG files.

The overall functioning of DISTRIBUTOR and BCG_MERGE within the OPEN/CÆSAR environment is illustrated in the previous figure.

Initialization and termination protocols. Besides the normal termination of the distributed LTS generation (when each instance has finished its local computations and no more messages are in transit), which is handled using a distributed termination detection algorithm, abnormal termination must also be handled properly. If the distributed LTS generation fails (e.g., because some machine has exhausted its memory) or the user decides to cancel it (e.g., by pressing Ctrl-C), DISTRIBUTOR must stop all the distributed activities of its instances. Therefore, a dedicated protocol is necessary. The solution we adopted in DISTRIBUTOR and BCG_MERGE is based upon a special process, called *coordinator*, which is launched on the local machine and has the charge of initializing the distributed computation (parsing the GCF file, establishing the connections from the local to the remote machines, launching the instances) and of detecting termination.



Real-time monitoring. Because end-users naturally want to observe the progression of distributed computations, DISTRIBUTOR and BCG_MERGE are equipped with (optional) graphical monitors providing information in real-time about computation status (when generating or merging of the PBG) and resource usage (processors and memories). The monitor of DISTRIBUTOR (see the figure above) is driven by the coordinator process, which periodically inspects the status of each instance. The monitor window has five panels, each giving a different view of the distributed computation. The "Overview" panel (on the left) shows, for each instance, the number of explored states (whose successors have been visited), the number of remaining states (visited, but not explored yet), and

the number of transitions in the corresponding fragment; the variation of remaining states (increasing, decreasing, steady) is represented as a coloured box (green, orange, red). The "Statistics" panel (on the right) shows various global data, such as the total and average number of visited and remaining states, of transitions, of labels, etc.

3 Conclusion and Future Work

Versions 3.0 of DISTRIBUTOR and BCG_MERGE are documented [8, 7] and distributed as part of the CADP toolbox. They run on several platforms (Linux, MacOS, Solaris, Windows). Experiments performed on various case-studies and different computing platforms have shown quasi-linear speedups and a good load balancing between machines.

We plan to continue our work along two directions. Using DISTRIBUTOR, generating very large LTSs becomes easier and one is now confronted to the limits of standard 32-bit machines, especially when state numbers become larger than 2^{32} and/or when BCG files become larger than 4 Gbytes. Shifting to 64-bit machines should solve these issues and overcome the current size limitations.

BCG_MERGE is valuable as it allows (at least for small or medium-sized models) to verify that the LTSs generated by DISTRIBUTOR are identical to those generated on a single machine. For large models however, BCG_MERGE may be a bottleneck because of the aforementioned 4 Gbytes limit. This can be avoided by performing verification directly on the PBG file, without invoking BCG_MERGE first. We seek to develop a PBG_OPEN tool connecting the PBG model to the API defined by OPEN/CÆSAR, thus allowing the model checking and equivalence checking tools of CADP to be applied on PBG models directly.

References

1. H. Garavel. OPEN/CÆSAR: An Open Software Architecture for Verification, Simulation, and Testing. In *Proc. of TACAS'98*, LNCS vol. 1384, pp. 68–84.
2. H. Garavel, F. Lang, and R. Mateescu. An Overview of CADP 2001. *European Association for Software Science and Technology (EASST) Newsletter*, 4:13–24, 2002.
3. H. Garavel, R. Mateescu, and I. Smarandache. Parallel State Space Construction for Model-Checking. In *Proc. SPIN'2001*, LNCS vol. 2057, pp. 217–234.
4. J.F. Groote and J. van de Pol. State Space Reduction using Partial τ-Confluence. In *Proc. of MFCS'2000*, LNCS vol. 1893, pp. 383–393.
5. Ch. Joubert and R. Mateescu. Distributed On-the-Fly Equivalence Checking. In *Proc. of PDMC'2004*, ENTCS vol. 128.
6. R. Mateescu. On-the-fly State Space Reductions for Weak Equivalences. In *Proc. of FMICS'05*, ACM, pp. 80–89.
7. Vasy. BCG_MERGE Manual Page. `http://www.inrialpes.fr/vasy/cadp/man/bcg_merge.html`, December 2004.
8. Vasy. DISTRIBUTOR Manual Page. `http://www.inrialpes.fr/vasy/cadp/man/distributor.html`, December 2004.
9. Vasy and Sen2. The VLTS benchmark suite. `http://www.inrialpes.fr/vasy/cadp/resources/benchmark_bcg.html`, March 2003.