Verifying distributed systems with unbounded channels

Régis Gascon & Éric Madelaine
INRIA Sophia Antipolis, CNRS - I3S - Univ. Nice Sophia Antipolis
2004, Route des Lucioles, BP 93, F-06902 Sophia-Antipolis Cedex - France
Email: First.Last@sophia.inria.fr

Abstract—**In the Vercors platform, we have implemented a prototype algorithm for model-checking systems of distributed components with unbounded FIFO channels. We describe this algorithm, and comment on the pragmatical ways to use it on simple cases.**

Usually, verification tools deal with infinite-systems by specifying abstractions that preserve the expected properties to produce finite-state models. Such models can be represented explicitly, by BDD-like structures, or even generated on-the-fly. Checking of properties is then performed by a (potentially exhaustive) search of the resulting state-space.

True "infinite-state" representations and algorithms allow one to represent in a finite way some infinite-state systems, and to prove some properties that would not be preserved by finite abstractions. This is typically the case for unbounded channels, for which there exist several (semi) decidable logic fragments, and associated algorithms. Nevertheless, there are few implementation of these theories, and we had to implement our own variant of a model-checking algorithm for finite sate machines communicating through Fifo channels.

In this paper, we present the foundation of this algorithm, and discuss our implementation, insisting on its pragmatic aspects.

I. Theoretical foundations

A. Communicating Finite State Machines

The formal definitions we use are adaptions of standard formalisms such as [2]. We consider *communicating finite state machines* (CFSM) that are finite automata extended with queues. A CFSM is a structure $\langle Q, I, C, \Sigma, A, \delta\rangle$ such that Q is a finite set of locations, $q_0 \in Q$ the initial location, C a finite set of channels, Σ a communication alphabet, A a set of actions and $\delta \subseteq Q \times Op \times Q$ the transition function. Op is the set of operations defined by:

- for every $\tau \in A$ we have $\tau \in Op$,
- for every $c \in C$ and $a \in \Sigma$ we have $c?a \in Op$ (receive), and $c!a \in Op$ (send).

We will shortly write $q \xrightarrow{op} q'$ whenever $\langle q, op, q'\rangle \in \delta$.

A network of CFSM is defined by a set of CFSM $\mathcal{M} = \{M_1, \dots, M_m\}$ and a set of channels $\mathcal{C} = \{c_1, \dots, c_n\}$. We denote each CFSM of the network $M_i = \langle Q_i, (q_0)_i, C_i, \Sigma_i, A_i, \delta_i\rangle$. We suppose without loss of generality that $\mathcal{C} = \bigcup_{1 \le i \le m} C_i$. We also define $\Sigma = \bigcup_{1 \le i \le m} \Sigma_i$.

For example, Fig. 1 shows a machine that receives messages on channels `ta` and `ts`, sends on channels `it` and `fm`, and has one internal action named `noExecT`.

A *configuration* of the network is a tuple $\langle \overline{q}, w_1, \dots, w_n\rangle \in (Q_1 \times \cdots \times Q_m) \times (\Sigma^*)^n$ where $\overline{q} = \langle q_1, \dots, q_m\rangle$ is the *global state* of the network and w_i the content of channel c_i for every $i \in \{1, \dots, n\}$. Each location $\overline{q}(i)$ is called the *local state* of M_i. The CFSMs can send and get messages from the channels which are unbounded and Fifo. By lack of space, we omit here the (straighforward) formal definition of the transition relation between configurations.

B. Reachable configurations

An *execution* (or a *run*) is a sequence of configurations respecting the one-step transition relation. We say that $\langle \overline{q'}, w'_1, \dots, w'_n\rangle$ is reachable from $\langle \overline{q}, w_1, \dots, w_n\rangle$ iff there is a run from the second configuration to the first one.

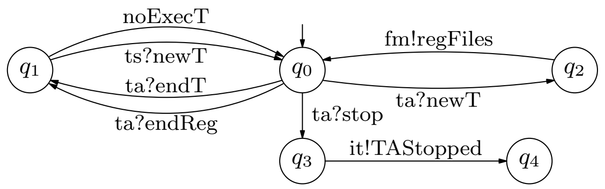

Figure 1. Simplified behaviour of TA

Our tool relies on an semi-algorithm for computing the exact set of reachable configurations. We cannot ensure termination because CFSMs are Turing powerful. We will use additional features to stop the algorithm in the sequel.

The set of reachable configurations in a network may be infinite because the channels are unbounded. A set of queue contents is represented by a union of deterministic finite automata (DFA) in the same way than [4]. Each DFA in the union is decomposed into a tuple of DFAs and each channel corresponds to a particular DFA in the tuple.

Given a set of configurations X, we define $\mathrm{Post}(X)$ to be the set of one-step successors of configurations in X, i.e.

$$\mathrm{Post}(X) = \{\langle q', w'_1, \dots, w'_n\rangle \mid \exists \langle q, w_1, \dots, w_n\rangle \in X \text{ and } op \in Op \text{ s.t. } \langle q, w_1, \dots, w_n\rangle \xrightarrow{op} \langle q', w'_1, \dots, w'_n\rangle\}.$$

For any set of configurations X represented with DFAs, computing the representation of $\mathrm{Post}(X)$ is easy. The effect of send operations on the queue representations is to append a new final state to the DFA encoding the receiving channel. The effect of receive operations is to change the starting state of the DFA if the message can be read. Thus, we can try to compute the set of reachable configurations as the fixpoint of $(\mathrm{Reach}_i)_{i\geq 0}$ such that $\mathrm{Reach}_0 = X_0$ and $\mathrm{Reach}_{i+1} = \mathrm{Reach}_i \cup \mathrm{Post}(\mathrm{Reach}_i)$ where X_0 is the set of initial configurations.

C. Acceleration

This computation does not terminate if the number of states is infinite. But we can improve this method. Consider a loop transition of the form $q \overset{c!a}{\rightarrow} q$. We can directly and exactly represent the result of iterating this cycle with a DFA that accepts the language a^*. This is called an *acceleration*. Accelerable cycles correspond to regular languages that can be represented with DFAs. A method to select accelerable cycles and compute the result of acceleration is described in [2]. We can use this to improve the algorithm. Let $\mathrm{Adm}(X)$ be a user defined set of accelerable cycles from configurations in X. Given a set of configurations X and a cycle $\gamma \in \mathrm{Adm}(X)$, we set

$$\begin{aligned}\mathrm{Post}^*_\gamma(X) &= \{\langle q, w'_1, \ldots, w'_n\rangle \mid \exists\langle q, w_1, \ldots, w_n\rangle \in X \\ &\qquad \text{s.t. } \langle q, w_1, \ldots, w_n\rangle \overset{\gamma^*}{\rightarrow} \langle q, w'_1, \ldots, w'_n\rangle\}\end{aligned}$$

where $\langle \overline{q}, w_1, \ldots, w_n\rangle \overset{\gamma^*}{\rightarrow} \langle \overline{q}, w'_1, \ldots, w'_n\rangle$ iff there is a run from $\langle \overline{q}, w_1, \ldots, w_n\rangle$ to $\langle \overline{q'}, w'_1, \ldots, w'_n\rangle$ applying a finite number of times the successive operations of γ. The computation of Reach can be rewritten as the fixpoint of $(\mathrm{Reach}'_i)_{i\geq 0}$ such that Reach'_0 is the set of initial configurations and

$$\mathrm{Reach}'_{i+1} = \mathrm{Reach}_i \cup \bigcup_{\gamma\in\mathrm{Adm}(\mathrm{Reach}'_i)} \mathrm{Post}^*_\gamma(\mathrm{Reach}'_i) \cup \mathrm{Post}(\mathrm{Reach}'_i).$$

Our tool implements this approach with heuristics for the selection of cycles. The set $\mathrm{Adm}(X)$ we consider is a subset of the set of cycles that can be theoretically accelerated.

II. Presentation of the method and tool

A. The implementation

Our verification tool is part of the Vercors platform for the specification and verification of distributed component systems [6]. The platform allows a graphical specification of the system from which is generated a formal model. Then various tools can be used for safety analysis or code generation, including classical finite-state model-checkers combined with abstraction techniques.

The present work is a first tentative to use infinite-system methods in the Vercors platform. Implementing infinite-state algorithms make the verification closer to the semantics of the formal model. The case of unbounded Fifo channels is certainly the one where the theory is the most advanced, and useful for our application domain. Other domains would also be interesting, in particular some decidable logics for counters.

Our prototype has been implemented in JAVA as an Eclipse plugin. The CFSMs are defined using Vercors graphical editor and saved in separate files. Several instances of the same CFSM can be loaded and the messages and channels of the different instances can be renamed. So, the tool also allows for a basic form of parametrization of the topology and instantiation of the system.

The current implementation contains a heuristic for acceleration: the set of accelerated cycle is restricted to sequences of send operations, sequences of receive operations and sequences of send/receive operations restricted to a single write channel and a different single read channel. This is a smaller set of cycles that in the theoretical results of [2]. In practice the other possible cases would be much more costly.

B. Exploring the state space

The most convenient scenario for our semi-algorithm is when the computation converges. In this case, we obtain the exact set of reachable states. However, in many cases the computation does not terminate, for instance because cycles induce non regular sets of queue contents, but also when the selected cycles produce growing DFA representations. We have implemented two mechanisms to control these cases, and to reduce the practical complexity of the search:

- The user can specify a global state or a configuration (global state + channels state) that is searched for reachability. In case of success, this is naturally smaller than exploring the full state space.
- Bounds on the search can be set, in terms of number of iterations of the main loop, of size of the generated DFAs, or of number of generated states.
- The user can define filters, that will reduce the amount of information displayed during the search; this is quite significant, because the display of DFAs in terms of regular expressions is costly.

In this prototype we have concentrated on functionalities rather than fine tuning of the data representation; a lot of space complexity could certainly be gained easily. Nevertheless, let us give an example showing basic figures about the algorithm.

The Integrated Toolkit (IT) in Fig. 2 is a system made of several (hierarchical) distributed components, typical of grid applications (see [5]. The detailed role of these components is not important here. It is modelled by a total of 6 CFSMs, communicating through 5 Fifo channels. We model this example by associating a CFSM and a request queue to every components. The behaviour of *primitive components* (TA, TS, JM, FIP, FTM) and the control part of *composites* (IT, FM) are described by CFSMs: the behaviour of component TA is shown in Fig. 1, and the part of the CFSM associated to IT that rules the stop procedure in Fig. 3. Components can send messages to the queues of other components but can read messages only in their own queue.

The computation of its full reachable state-space would not converge. Typical properties one could check are:
(1) is there states where every component is waiting for new requests ?
(2) can a component (e.g. TA) be stopped with a non-empty queue ?

When checking for reachability of configuration (1), we first tried to bound the number of iterations of the main loop. After 15 iterations, the program already indicates that the queue

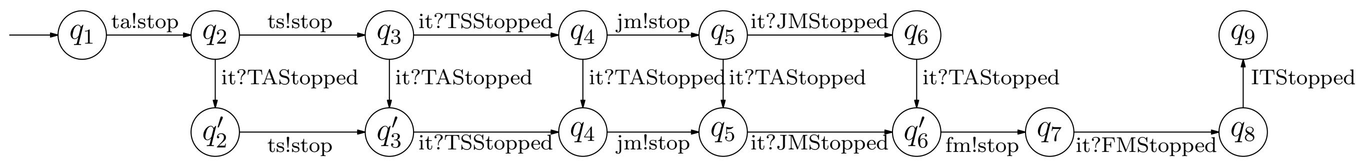

Figure 3. Stopping procedure of IT

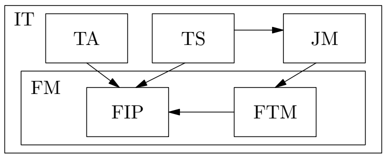

Figure 2. Integrated Toolkit (IT)

representation is very large (2460 DFAs and a total of 19096 states). Displaying the content of channels in this case is not very useful because the corresponding regular expressions are too long. So our next try was to limit the size of the DFAs (this does not mean bounding the size of the queues). The resulting state-space is much smaller (78 automata and 275 states), and allowed us to examine the queue contents in a reasonable form; e.g. the content of channel ta is $(\mathrm{NewT}^* \cdot \mathrm{EndReg}^* \cdot \mathrm{EndT}^*)^*$, meaning that ta contains (at least) a message.

Similarily, we checked that configuration (2) can be reached, that is a situation usually considered as a bug in some distributed component systems (when other components are waiting these requests to be served).

III. CONCLUSION

We have implemented an algorithm of an *infinite-state* model-checker for communicating finite-state machines with unbounded Fifo channels. This prototype is part of the Vercors platform, and is used for the verification of properties of asynchronous component systems.

The algorithm is a semi-decision algorithm, able to construct exact representations of the system configurations when the (unbounded) channel contents are regular. We have discussed several ways of controlling the search, and shown an example of usage of the tool.

We have shown some properties that were proved, involving unbounded channel content, that could have not have been treated using finite abstractions.

Similar research exist in other teams, in particular the LASH tool that implements efficient data structures based on [2], but does not provide any state-space or search procedure. The work by [3], and also the TReX tool [1] also provide analysis procedures for unbounded channels, but based on over-approximations of configuration sets rather than exact representations.

The main focus of our work here was on the applicability of the method in the context of the Vercors platform. Much work is still required, in particular in practical complexity, but also in term of usability by non-specialist users.

In the longer term, we will study other (semi) decidable logic fragments, in particular for representing counters and arithmetics, and also the combination of such fragments.

REFERENCES

[1] A. Annichini, A. Bouajjani, and M. Sighireanu. Trex: A tool for reachability analysis of complex systems. In *CAV'01*, volume 2102 of *LNCS*, pages 368–372. Springer, 2001.

[2] B. Boigelot, P. Godefroid, B. Willems, and P. Wolper. The power of QDDs (extended abstract). In *SAS'97*, volume 1302 of *LNCS*, pages 172–186. Springer, 1997.

[3] T. Le Gall, B. Jeannet, and T. Jéron. Verification of communication protocols using abstract interpretation of fifo queues. In *AMAST'06*, volume 4019 of *LNCS*, pages 204–219. Springer, 2006.

[4] S. Roy and B. Chakraborty. A finite union of DFAs in symbolic model checking of infinite systems. In *CIAA'06*, volume 4094 of *LNCS*, pages 277–278. Springer, 2006.

[5] E. Tejedor, R. Badia, P. Naoumenko, M. Rivera, and C. Dalmasso. Orchestrating a safe functional suspension of GCM components. In *CoreGRID integration workshop. Integrated research in grid computing*, 2008.

[6] The Vercors platform: VERification of models for distributed communicating COmponants, with safety and Security. `http://www-sop.inria.fr/oasis/index.php?page=vercors`.