Project 22703: SYRF

Deliverable 3.1.2

Symbolic abstraction of automata

---

1 Approach

Automatic verification and model-checking rely on precise formal semantics, as provided in synchronous reactive formalisms. Then many avenues of analysis techniques can be applied, each relying on some form of symbolic abstraction of the underlying reactive automata. These techniques usually depend on *some* form of representation of the reachable state space of the full reactive system, or an upper approximation of this space of configurations. We proceeded this year with the practical and theoretical work defined in this domain in year 1, and applied it on case studies. We also identified new "properties of interest" and developed relevant algorithmic methods.

The deliverable contains contribution from three distinct sites (INRIA Meije, INRIA Spectre, and INRIA Ep-Atr).

2 INRIA Meije contribution

2.1 FSM minimisation

The prototype tool Fc2symbmin implementing *symbolic bisimulation* was heavily tested and upgraded, noticeably on the small SAAB climatic chamber example. The results were found to overwhelm our expectations concerning the reduction folding of many symbolically labelled transitions into respectively few. A new robust version of the tool is now available. We now plan to apply these techniques on the full SAAB example.

2.2 Compositional reduction of Boolean Equation Systems

The downsizing of boolean equation systems is more complex to define as its automata counterpart, due to the non-existence of obvious unique canonical forms. *Optimisation heuristics* and related efficient algorithms have been a long-time concern, mostly in hardware synthesis. We borrowed techniques from this source, but also introduced new reduction methods as appropriate in the specific context of synchronous reactive systems. The tool Xeve, which operates on full BDD representation of Mealy automata, was enriched with reencoding functions, taking advantage of structural information (drawn from Esterel programs for the time being).

We conducted experiments "by hand" mimicking the effect of compositional reduction of boolean equation systems (were components are optimized before they get linked into a larger systems). The results are fully promising. The corresponding software linker is very close from completion (should be ready by the time of the second year review). It currently works at the level of proprietary SC format. The application of these techniques to DC format would (only) require a preprocessor internal to DC to separate the boolean control part from the data flow part in programs. This processing is also a key step in model-checking.

Compositional reduction and linking of programs at the boolean equation system level was also found to be extremely valuable to allow the combinaison of a program under analysis with a synchronous reactive observer, saving the duty to reoptimise and recompute a simplified form of the full compound boolean equation system (and symbolic state space if already computed). Experiments in this direction have also been conducted, in connexion with the Dassault-Aviation company.

2.3 Easy formulation of particular sets for model-checking

The purpose of synchronous reactive observers is to reduce general properties and property checking to the checking of a small set of predefined properties, on a larger compound systems resulting of the combinaison of the analysed program with the observer. Typical predefined properties are the eventual emission of a given output signal (indicating reachability of a certain observer state), or its converse (livelock states which could never in the future lead to further signal emission). Divergent states, which potential lead to infinite loops of mute behaviours, but only under specific input sequences, are also useful to identify. All these properties (and sets of states enjoying them) can be made relative to partial masking of the transition relation or the traversal of subspace of states. We developed a prototype extension of the XEVE tool [Boua98] introducing "BDD-aware" algorithms implementing the previous predefined properties, together with syntactic ways to specify parameter masks. These functions should now be tested on real applications. A further problem is raised by the need of *transparent* boxes, were the internal registers of the analysed Mealy automaton are available for observation (so that one can check for instance the reachability of a non-mutually exclusive configuration betwen two components); then the register naming information must be passed (and preserved) across DC internal format transformations.

3 INRIA Spectre contribution

This new work has been done in cooperation between EDF (Jean-François Héry and Jean-Claude Laleuf) and Inria-Spectre. It takes place in task 3.1, because it concerns analysis of automata without explicitely building their state graphs, and makes use of the BDD technology. It is also an instance of new "properties of interest", and is linked to (asynchronous) code distribution of synchronous network in a way. This work is described in detail in Annex 3.1.2.??.

3.1 Stability of Synchronous Systems

A synchronous system is said to be *stable* if, whenever its inputs stop changing, its outputs reach a stable value after a finite number of steps. Intuitively, it means that the system cannot involve an autonomous change of its outputs: every change of the outputs are the consequence of an input change (possibly occurring several steps before). This property appears to be a strong consistency requirement in industrial sampled systems: this work was raised by real problems in nuclear plants control software. In fact, stability is a generic consistency property of reactive programs — like deadlock or starvation freeness in parallel systems.

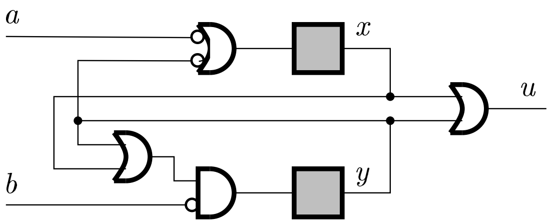

Figure 1: a Mealy machine as an operator network

In synchronous programming, one can easily write unstable programs: for instance, in LUSTRE, the equation

x = true u/zenon/2/ra/98/tmp/meijett -¿not pre(x);

defines a Boolean variable that changes at each cycle whatever be the inputs to the program. In ESTEREL, since "tick" has been introduced in the language, the same phenomenon can appear. The situation is even worse in SIGNAL, where programs can oversample their inputs.

3.2 Checking Stability of Boolean Reactive Systems

First notice that stability is neither trivial to specify, nor to verify. As a temporal logic formula [Pnu77] on inputs ι and outputs ω, it would be written like

$$\Box(\Box(\iota \equiv \bigcirc\iota) \Rightarrow \Diamond\Box(\omega \equiv \bigcirc\omega))$$

and verifying it by model-checking [VW86] is likely to be very expensive, as it may involve the construction of the whole state graph of the program.

In this work, we propose heuristics to check stability of Boolean reactive systems, without building their state graph. We consider such systems as Mealy machines, descvribed either by systems of Boolean equations, or by operator networks (circuits). For instance, the system of Boolean equations:

$$\begin{array}{rcl} x' & = & \overline{y} + \overline{a} \\ y' & = & \overline{b}\,(x+y) \\ u & = & x+y \end{array}$$

defines the machine with 2 input variables, a and b, 2 state variables (memories), x and y, and 1 output variable, u, where a "primed" state variable denotes its "next" value. Fig. 1 shows the operator network of the same machine, made of combinational gates ("and", "or", "not") and memories.

On such machines, we analyze a stronger property than the stability defined before: we will check that, whenever the inputs remain stable, all the *memories* stabilize after a finite number of steps.

To give an idea of the kind of heuristic we apply, let us consider the most effective one (other conditions are proposed in Annex 3.1.2.??): let x be a state variable, the behavior of which is defined by the equation "$x' = f_x$", and let us consider the Shannon decomposition of f_x according to x: $f_x = x.\overline{reset_x} + \overline{x}.set_x$. The cofactors directly give the condition — on inputs and other memories — set_x (resp., $reset_x$) on which x changes from 0 to 1 (resp., from 1 to 0). If x is unstable, it should be able to change in both direction with the same values for inputs. In other words, there should exist two instants t_1 and t_2, such that the inputs have the same value at t_1 and t_2, while set_x is true at t_1 and $reset_x$ is true at t_2. Formally, if we note ι the vector of common input values at t_1 and t_2, and μ_i the vector of memory values at $t_i, i = 1 \ldots 2$, we get the necessary condition

$$\mathcal{U}_x \;=\; \exists\iota, \exists\mu_1, \mu_2,\; set_x(\iota, \mu_1).reset_(\iota, \mu_2)$$

If this condition is identically false, the memory x is stable, and can be considered as an input in applying the same criterion to other memories.

Example: Let us come back to the machine of Fig. 1. We have

$$set_x \;=\; \overline{y} + \overline{a} \quad , \quad reset_x \;=\; y.a \quad , \quad set_y \;=\; x.\overline{b} \quad , \quad reset_y \;=\; b$$

Starting with the inputs $\{a, b\}$ as only stable variables, we compute

$$\begin{aligned}
\mathcal{U}_x &= \exists y_1, y_2,\; set_x(a, b, y_1).reset_x(a, b, y_2) \\
&= \exists y_1, y_2,\; (\overline{y_1} + \overline{a})(y_2.a) \\
&= \exists y_1, y_2,\; \overline{y_1}.y_2.a \\
&= a \\
&\not\equiv 0
\end{aligned}$$

so nothing can be concluded about the stability of x. On the other hand, we have

$$\begin{aligned}
\mathcal{U}_y &= \exists x_1, x_2\; set_y(a, b, x_1).reset_y(a, b, x_2) \\
&= \exists x_1, x_2\; (x_1.\overline{b}).b \\
&= 0
\end{aligned}$$

So, y is stable, and this knowledge can be used in computing $\mathcal{U}_x$ again:

$$\begin{aligned}
\mathcal{U}_x &= \exists y_1,\; set_x(a, b, y_1).reset_x(a, b, y_1) \\
&= (\overline{y_1} + \overline{a}).(y_1.a) \\
&= 0
\end{aligned}$$

and x is found stable too.

4 INRIA Ep-Atr contribution

4.1 Intensional Approaches for Symbolic Methods

Last year, we have presented *Implicit Labeled Transition Systems* as intermediate models for discrete event systems. We have studied operations of parallel composition and event hiding, as well as an equivalence criterion based on strong bisimulation semantics, namely the equivalence called *symbolic bisimulation*.

The aim of the work is to rely on intensional descriptions of the systems for symbolic abstractions purposes, based on behavioral equivalences. The intensional approach we propose has the main advantage to remain at an interesting level of abstraction to handle sets (of states, of events) in which algorithms and properties can entirely be expressed, thus providing us with a high level of expressiveness. This abstract level avoids us to bother with a particular choice of sets implementation, such as BDDs, even if all operations are actually based on this representation for sets. Moreover, the intensional formalism is completely compatible with the symbolic techniques, since intensionally described sets can afterwards be implemented by any kind of methods, such as decision diagrams. However, other tools with different implementations might be considered using the same high level framework (e.g. Mathematica, Maple) [Dut92, GJ94].

Intensional models have already been the subject of previous work [LBMR96], under the name of *polynomial dynamical systems.* They were the base of the temporal logics verification tool SIGALI for the SIGNAL language we develop in our group in Rennes. The task 3.1 of the SYRF project has led us to enrich the scope of this verification tool by implementing equivalence checking, such as strong bisimulation on the basis of the intensional philosophy. This new functionality of SIGALI is of high interest since the SIGNAL language is used in a lot of areas (controller synthesis [DLB94], [ML98b, MBLL98] (see also Part II of the report), robotics [MRMC98],...) where models equivalence checking is crucial. Indeed, using reduction techniques, we hope for dealing with smaller models, then improving the efficiency of the tool SIGALI.

4.1.1 Results achieved this year

This year, we aimed to complete the equivalence checking methodology by integrating in SIGALI the construction of a reduced model according to a given equivalence. This work has the advantage to design the reduced model still in an intensional manner, whereas in classic approaches symbolic models might become explicit when reduced [Fer84]. However, our method still has some explicit aspects features (we cannot yet avoid the equivalence class enumeration). Note that the reduced model is built "on the fly" during this unique enumeration phase.

Regarding the underlying theory we have presented last year, we would like to emphasize, this year, the new presentation of the semantic models, that we call *Intensional Labeled Transition Systems.* The models are still based on the specific domain of SIGNAL programs boolean abstraction, that is $\mathbf{Z}_3$, but the theory could be generalized to other domains (e.g. the family of finite domains $\mathbf{Z}_p$, with p prime), provided we clearly identify the hypothesis for algorithms termination (for example, the finitely branching property of the underlying explicit models) and the decidability properties that are needed to apply the techniques. But this is out of the scope of the project.

4.2 Technical paper

The full report (given in appendix) is organized as follows: First, we introduce the theoretical framework, the so-called *intensionally Labeled Transition Systems (iLTS)*, as the labels of the transitions are sets defined "intensionally". Also parallel composition and event hiding are considered as basic combinators for the modular design of systems. Next, the equivalence of *symbolic bisimulation* is defined with an associated study of some properties, s.a. a congruence theorem and other characterizations.

Then, next section, the models become "fully" intensional, called *Intensional Labeled Transition Systems (ILTS)*: the set of all possible transitions of the system is itself describe intensionally, just like symbolic methods proceed classically. On the basis of those models, we propose algorithms to compute the symbolic bisimulation between two ILTS, but mostly a method to build the reduced model according to this equivalence.

The integration environment in which our methods are to be applied is presented: we present the language SIGNAL and explain how the Intensional Labeled Transition System is (automatically) derived from a SIGNAL program. Then, extensions of the verification tool SIGALI are briefly commented.

4.3 A Design Environment for Discrete-Event Controllers based on the Signal Environment

4.3.1 Foreword

The Signal language is dedicated to reliable specifications of real-time reactive systems. Many applications require high reliability and safety. Traditionally, these requirements are checked *a posteriori* using simulation techniques and/or property verification. Control theory of discrete event systems allows to use constructive methods, that ensure, *a priori*, required properties on the system behavior. In this approach, the validation phase is reduced to properties not guaranteed by the programming process.

4.4 Approach

Different theories dedicated to this kind of control of discrete event systems have emerged since the 80's. Usually, the starting point of these theories is: given a model for the system and the control objectives, a controller must be derived by various means such that the resulting behavior of the closed loop system meets the control objectives.

In our case, the specification of the physical model (or the plant) is achieved using the synchronous data-flow language Signal. Control theory is then applied on an equational model of the control part (i.e. logical) of Signal programs. For this purpose, the boolean part of the program is translated into a polynomial dynamical system (PDS) over Z/3Z. Furthermore, using such algebraic methods and polynomial dynamical system as a formal model, we are able to synthesize controllers satisfying various kinds of control objectives (they can be expressed as *invariance, reachability* and *attractivity* [DLB94] as well as as order relations over the states of the plant [ML98a, MBS98]. As Signal is a data-flow language, meaning that the system reacts to inputs sent by the environment and produces outputs resulting from internal transformation, it is natural for us to use an input/output approach. The physical model is then represented by a polynomial dynamical system while the control of the system is performed by restricting the controllable input values to values suitable for the control goal. This restriction is obtained by incorporating new algebraic equations into the initial system.

4.4.1 Results achieved this year

1. **The Design Environment for Discrete-Event Controllers:**

   We developed around the Signal environment a tool that allows the visualization of the synthesized controller by an interactive simulation of the controlled system[ML98b].

   In a first stage, the user specifies in Signal both the physical model and the control objectives to be ensured. A second stage is performed by the Signal compiler which translates the initial Signal program into a PDS, and the control objectives in terms of polynomial relations and polynomial operations. The controller is then synthesized using our formal

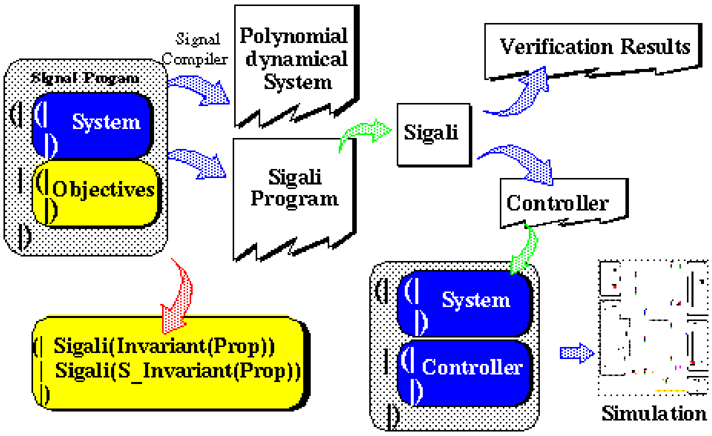

Figure 2: Brief overview of the tool

calculus tool Sigali. The result is a polynomial that describes the controller (that polynomial is implemented as a Binary Decision Diagram (BDD)). Finally, in a third stage, in order to visualize the new behavior of the controlled system, the obtained controller and some simulation processes are automatically included in the initial Signal program. It is then sufficient for the user to compile the resulting Signal program which generates executable code ready for simulation.

2. **Examples:**

   Different academic examples are used to illustrate the application of the tool. The first one, namely the cat and mouse example, has been first introduced by Ramadge and Wonham. The model used for this problem is an automaton. The second example deals with the control of a production cell. In this case, the model is a Petri net. Our goal here is not to describe in detail the specification in Signal of the models, but rather to show that our tool is generic and can be used to specify different kinds of systems, with various descriptions (automata, Petri nets, formal descriptions, etc).

References

[DLB94] B. Dutertre and M. Le Borgne. Control of polynomial dynamic systems : an example. Technical Report 2193, Institut National de Recherche en Informatique et en Automatique (INRIA), January 1994.

[Dut92] B. Dutertre. *Spécification et preuve de systèmes dynamiques.* PhD thesis, Université de Rennes I, September 1992.

[Fer84] J.-Cl. Fernandez. *ALDEBARAN: un Système de Vérification par Réduction de Processus Communicants.* Thèse de Doctorat, Univ. Joseph Fourier-Grenoble I, France, July 1984.

[GJ94] J. Gunnarsson and Plantin J. Control law synthesis for a discrete event system. Technical Report LiTH-ISY-R-, Linköping University, 1994.

[LBMR96] M. Le Borgne, H. Marchand, and E. Rutten. Formal verification of SIGNAL programs: Application to a power tranformar station controller. In *Proc. of the 5th Int. Conf. AMAST'96, Munich, Germany, LNCS 1101*, pages 270–285, July 1996.

[MBLL98] H. Marchand, P. Bournai, M. Le Borgne, and P. Le Guernic. A design environment for discrete-event controllers based on the signal language. In *1998 IEEE International Conf. On Systems, Man, And Cybernetics*, San Diego, California, USA, October 1998.

[MBS98] H Marchand, O. Boivineau, and Lafortune S. On the synthesis of optimal schedulers in discrete event control problems with multiple goals. In *1998 IEEE International Conf. On Systems, Man, And Cybernetics*, San Diego, California, USA, October 1998.

[ML98a] H. Marchand and M. Le Borgne. On the optimal control of polynomial dynamical systems over $\mathbf{Z}_p$. In *4th International Workshop on Discrete Event Systems*, pages 385–390, Cagliari, Italy, August 1998.

[ML98b] H. Marchand and M. Le Borgne. Partial order control of discrete event systems modeled as polynomial dynamical systems. In *1998 IEEE International Conference On Control Applications*, Trieste, Italie, September 1998.

[MRMC98] E. Marchand, E. Rutten, H. Marchand, and F. Chaumette. Specifying and verifying active vision-based robotic systems with the signal environment. *Int. Journal of Robotics Research*, 17(4):418–432, April 1998.

[Pnu77] A. Pnueli. The temporal logic of programs. In *18th Symp. on the Foundations of Computer Science*, Providence R.I., 1977. IEEE.

[VW86] M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Symposium on Logic in Computer Science*, June 1986.

[Boua98] A. Bouali. XEVE, an Esterel Verification Environment. In *Proceedings of the Tenth International Conference on Computer Aided Verification*, July 1998.