Project 22703: SYRF

Deliverable 3.1.1

Symbolic abstraction of automata

1 Approach

The basic paradigm for modularity in reactive formalisms consists in designing the full system/program as a collection of interacting parallel components. With *synchronous* reactive systems providing common clocks and signal broadcast, the addition of a new parallel module preserve to a large extend the previous behaviour of (the rest of) the system. This is not always entirely true, but one may get a formal diagnostic of disturbance by formally comparing two finite state machines: the one representing the system prior to the addition of the later module, and the one with the module (but abstracted away on new interface signals possibly brought in by this extension).

This formal check makes special sense for instance in telephony systems, where modules implement new features, and the problem of transparent addition amounts to *feature interactions*. But the method has a broader scope, and we have found it widely applicable in a lot of cases, with very little demand on the user concerning the provision of correctness properties (here the submodule is itself its spec).

This method uses an algorithm for checking (finite state) machine equivalence, in a context where possibly previous signal hiding introduced non-determinism (while originally synchronous systems exhibit deterministic behaviours as a function of their current state and input events). We call this equivalence *symbolic bisimulation*, where the adjective *symbolic* is meant to recall here that, at the concrete representation level, input events are represented symbolically as propositional predicates on signals, while the state structure itself can be either implicit or explicit. Negated litterals retain the fact that transitions may be triggered by the *absence* of a signal.

2 INRIA Sophia-Antipolis contribution

2.1 General description

In presence of symbolic input predicates on transitions the bisimulation checking algorithm must be refined: transitions are not searched for a 1-1 match, but instead a collection of predicates borne on a collection of transitions (from a given state to a set of yet equivalent states, with identical output events), may recover a single predicates from another state. This added complexity is easily coped with in the classical partitioning algorithm, but not in other variations such as Paije-Tarjan or on-the-fly methods.

The definition and full motivation for this work are provided in a research report. We implemented the algorithm twice: first, for an explicit automaton representation in a tool called FC2SYMBMIN, which works on Mealy type FSM in a speficic representation format named `fc2`; second, for an implcit state space representation, in a tool named BLIFFC2 which starts from a format for netlist description named `blif`. Both kinds of descriptions can be produced by the Esterel compiler.

We found out that in most cases FC2SYMBMIN performed remarkably well in the (factoring) minimisation of transition numbers, and also in minimisation of states. Further signal hiding improve the reduction ratio. We applied these techniques to a climatic chamber example provided by an industrial partner inside the SYRF project. We provide the theoretical foundations and applications of abstraction on a Steam-Boiler problem in a separate research report, also obtainable from URL http://www.inria.fr/rapports/sophia/RR-3285.html . We briefly sketch informally the basic definitions and results in the next section.

2.2 Mealy machines and their symbolic representation

Let *In* and *Out* be two finite sets of respectively input and output signals. A Mealy FSM is a finite automata labeled in $2^{In} \times 2^{Out}$. The natural interpretation is that, supposing that exactly these input signals are simultaneously occuring together, then the corresponding output signals are emitted in instantaneous reaction while the transition is performed.

We call input event any set of input (meant to be presented to the machine as occuring simultaneously. The FSM is said to be *deterministic* if for each state and input event there is at most one fireable transition. It is said to be *complete* if for each state and input event there is at least one fireable transition. Mealy machines used as models of digital circuits or synchronous languages are usually complete and deterministic, so that there is a completely defined behaviour for any incoming event in any state. The notions of bisimulations and trace equivalences are straightforward.

Now this basic definition is generally too concrete, because often many possible input events shall trigger essentially the same behaviour (for instance: perform this transition as soon as a given signal is present, no matter what the other inputs are). The distinction is generally overlooked, and Mealy machine representations are generally based on predicates for input event selection. This hides the fact that new transitions now each stands for a set of transitions in the former sense (a predicate can be fulfilled by a set of events). To avoid confusion we call *reactive automata* these machines labelled in $2^{2^{In}} \times 2^{Out}$.

To each reactive automaton one can associate a Mealy FSM, by expanding transitions along all possible successful event valuations of the predicates. Of course the association is *not* generally one-to-one. A reactive automaton is said to be deterministic or complete if the underlying Mealy FSM is, and similarly two reactive automata are bisimilar when their corresponding Mealy FSMs are. But the true algorithmic problem is to compute these relations at the level of reactive automata and input predicates, instead of returning to fully expanded level.

2.3 Equivalence checking

We shall not recall the traditional coarsest partition fixpoint algorithm for bisimulation. Informally, it is adapted to our case by noticing that instead of matching transitions pairwise (with identical labels, leading to states so-far still equivalent, in order to establish which source states will remain so), one must collect the full union of transitions with identical outputs and leading to states so-far still equivalent, in order to partition source states: those with identical unions of input predicates will remain paired for this iteration step.

The fact that the split provides possibly more than two classes, and that it is necessary to collect *all* transitions leading to (recursively) equivalent states tends to put a penalty on Paije-Tarjan and local/on-the fly methods. The need for checking equality between dispunctions of predicates calls for their canonical representation, provided by BDDs.

Our FC2SYMBMIN tool performs minimisation based on these ideas, possibly with a previous signal hiding abstraction. This hiding operation usually introduces nondeterminism (which we deal with), and erases many distinctions between states so that reduction becomes drastic when only a few relevant signals are kept. We found one very specific framework in which this methodology was particularly fruitful: consider a synchronous process, and augment it with a new reactive module (adding a new service for instance); how did this new module interfere/interact with the rest of the system ? Some answer can be found by comparing the original system with the augmented one, once they have been abstracted away from signals which do not occur commonly at their interface.

Our experiments showed large savings in states, but even larger in transitions due to the merging and folding into simpler predicates. In the coming years we plan to extend our work to compositional verification approach along these lines.

3 INRIA Rennes contribution

3.1 Symbolic representation in the SIGNAL environment

The multi-clock data-flow synchronous language SIGNAL manipulates *signals*, and its constructs can be used in an equational style to specify the relations between signals. Systems of equations on signals are built using a *parallel composition* as well as a *events hiding* constructs, thus defining *processes*.

The equational nature SIGNAL makes that it relies on a formal model in terms of *polynomial dynamical equations systems (PDS)*, which can be seen as an equational representation of an automaton.

The theory of PDS uses classical tools in algebraic geometry such as *ideals, varieties* and *morphisms*. This is the basis of the verification tool SIGALI, developed in the EP-ATR group. The techniques used in consist in manipulating the system of equations instead of the sets of solution, which avoids the enumeration of the state space. Each set of states is uniquely characterized by a polynomial and the operations on sets can be equivalently performed on the associated polynomials.

Therefore, a wide spectre of properties, such as liveness, invariance, reachability and attractivity can be checked. The PDS abstraction is automatically performed by the

SIGNAL compiler and the resulting PDS is then used as a formal basis for verification of SIGNAL programs. Many algorithms for computing predicates states are available.

We want here to adapt these original techniques of the SIGNAL environment to the case of symbolic bisimulation (see sections above for full motivation of this work), and we are developing the algorithm for the symbolic bisimulation computation in the SIGNAL environment.

First, we introduce the models, *implicit labeled transition systems*(ILTS). The class of ITLS can be provided with many operations. Among them, the *parallel composition* and the *events hiding* offer powerful constructs to design more complex systems from simple ones. We prove the congruence theorem stating that symbolic bisimulation is a congruence w.r.t. abovementioned constructs.

We propose a behavioral equivalence relation over sets of ILTS, which definition is strongly inspired from de Simone's symbolic bisimulation over reactive automata. It can in fact be established that provided the ILTS are encoded into reactive automata, the two definition coincide.

Next, polynomial dynamical systems are presented and naturally equipped with an operational semantics in terms of implicit labeled transition system. The algebraic geometry of polynomials makes it possible to perform implicit model-checking techniques over the underlying transition system. The techniques are already implemented in the tool SIGALI.

In our work, we mainly focus on the notion of *greatest invariant* since it is used for the computation of the symbolic bisimulation : take a labeled transition system with set of states Q, and $V \subseteq Q$. We aim to compute the set of states $\mathcal{GI}(V)$ called *the greatest invariant contained in* V defined by $\{x \in Q | x \in V \wedge \forall x \xrightarrow{y} x', x' \in V\}$. This can be done implicitly by the tool SIGALI in the PDS $S(n,m) =< E(X,Y,X') >$, provided we have a canonical generator g representing V, i.e. $Sol(g) = V$. SIGALI computes a canonical generator $\hat{g}$ of $\mathcal{GI}(V)$. From this formal setting, the algorithm computing the symbolic bisimulation is derived: polynomials techniques with the SIGALI tool are used to describe it. This way, one can build the quotient ILTS (reduced PDS) to improve the SIGNAL environment and its verification tool.

In a separate draft research report, entitled "Symbolic methods in the SIGNAL environment", by S. Pinchinat and O. Kouchnarenko, we provide the theoretical background, definitions and practical results.