Combining simulation and guided traversal for the verification of concurrent systems *

Enric Pastor and Marco A. Peña
Department of Computer Architecture
Technical University of Catalonia
08034 Barcelona, Spain
{enric, marcoa}@ac.upc.es

Abstract

We present a hybrid methodology that combines simulation and symbolic traversal in order to improve invariant checking. The methodology concentrates on concurrent systems, whose peculiarities are not fully exploited by other existing techniques for hybrid verification. Our approach exploits the information obtained from simulations to improve the knowledge of the state space, effectively guiding symbolic traversal.

1. Introduction

In recent years mixed approaches have been introduced combining the benefits of simulation and symbolic traversal techniques, thus coining the term *hybrid verification*. Instead of ensuring the complete exploration of the state space, hybrid verification pretends to provide efficient mechanisms to identify significantly large portions of the space space with a reduced computational complexity. Hybrid verification has been traditionally useful when the size of the system under analysis is too large to be fully verified by conventional means. In this cases, hybrid verification provides the designer with positive feedback to improve the reliability of the system in terms of the failures discovered.

We present a hybrid verification strategy tailored for asynchronous concurrent systems (including asynchronous circuits [6, 2], network protocols, distributed systems [5, 3], etc.). Even though existing hybrid approaches may be extended to asynchronous concurrent systems, to the best of our knowledge none of them exploits their peculiarities, i.e. the interleaved execution of concurrent events.

We propose a two-step mechanism based on a combination of simulation and reachability analysis (see Figure 1). In a first step, simulation provides an initial depth-first view of the states in the system. In order to guarantee a good coverage of the state space, simulation detects those states where the system chooses between excluding execution sequences (*i.e. branching sequences*). Then, each one of the possible excluding sequences will be further explored.

* Funded by ACiD-WG (ESPRIT 21949) and the Ministry of Education of Spain (TIC2001-2476-C03).

Then, symbolic traversal is applied to improve the state coverage. The information about the ordering in which events are fired, obtained by simulation, is used to guide the way in which traversal is applied [6]. Reachability analysis is performed for each one of the sequences generated by the simulation phase, accumulating the obtained states.

2. Simulating Transition Systems

A *transition system* (TS) [1] is composed of a non-empty set of states $\mathcal{S}$, a non-empty alphabet of events Σ, a transition relation $T \subseteq \mathcal{S} \times \Sigma \times \mathcal{S}$, and a set of initial states S_{in}. A TS is a formalism oriented to modeling asynchronous concurrent systems that emphasizes the execution of abstract events rather than how they are encoded.

We introduce a novel simulation approach for transition systems that automatically provides a good state space coverage. At each explored state the causality between firable events is analyzed to identify conflicts between them. Conflict detection allows to identify execution sequences that exclude each other. Simulation chooses a particular ordering among all possible interleaved executions of concurrent events, limiting its effectiveness. Section 3 shows that event interleaving due to concurrency can be explored efficiently by symbolic traversal once the information from a simulation sequence is available.

Figure 2 illustrates different conflict situations between events: (a) shows three events e_1, e_2, e_3 that are mutually concurrent; (b) shows a symmetric conflict between e_1 and e_2; and (c) shows an asymmetric conflict in which e_2 disables e_1 but not the contrary (event e_3 remains concurrent to e_1 and e_2).

Symmetric conflicts are associated to states in which the system takes a decision. Each branch may involve completely different sets of events and thus produce distinct/disjoint sets of states. Different sequences are generated for each branch in order to achieve a better coverage of the state space. Asymmetric conflicts can be associated to *disablings*, in which the firing of one event (the *disabler*) prevents the firing of a second event (the *disabled*).

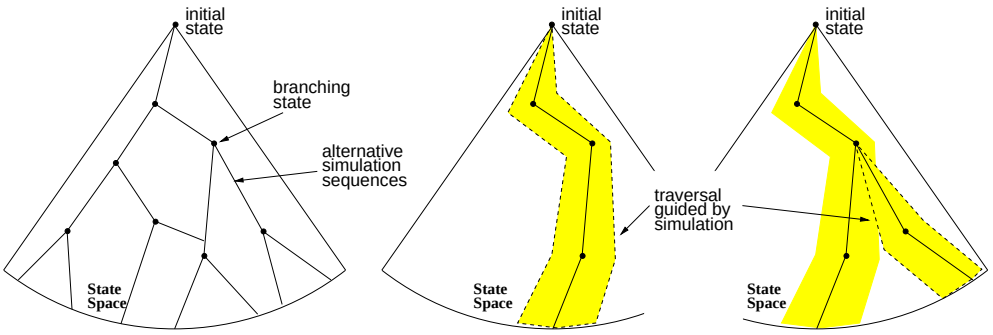

Figure 1. Two-step scheme: simulation plus guided-traversal.

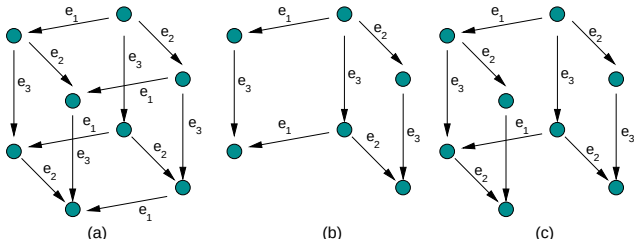

Figure 2. Concurrent and conflict structures.

3. Guided traversal

The sequences generated by the simulation, together with the *enabling* information at each state, allows to analyze the causality relations between events. Such causality is exploited to improve the traversal by using chaining [6].

Causal event structures describe all possible sequential and concurrent executions of a set of events. A *causal event structure* [4] (CES) is a tuple $\langle \Sigma, \prec \rangle$ where $\Sigma = \{\mathsf{e}_1, \ldots, \mathsf{e}_n\}$ is a finite set of *events* and $\prec \subseteq \Sigma \times \Sigma$ is a strict partial order (irreflexive and transitive) over Σ called the *causality relation*. Given a CES the following relations can be defined where **co** is called the *concurrency relation*:

$$\begin{aligned} \mathbf{id} &\stackrel{def}{=} \{(\mathsf{e},\mathsf{e}) \mid \mathsf{e} \in \Sigma\} \\ \succ &\stackrel{def}{=} \{(\mathsf{e}_1,\mathsf{e}_2) \mid (\mathsf{e}_2,\mathsf{e}_1) \in \prec\} \\ \mathbf{co} &\stackrel{def}{=} \Sigma \times \Sigma - \{\prec \cup \succ \cup \mathbf{id}\} \end{aligned}$$

Provided a sequence of events, we can recover a partial order showing the causality relationships among those events, *i.e.* a CES. A *topological order* of the events of a CES is a sequence $\mathsf{e}_1 \cdots \mathsf{e}_n \in \Sigma^*$ $(n = |\Sigma|)$, such that all e_i are distinct and $\forall\, 1 \leq i, j \leq n \;:\; \mathsf{e}_i \prec \mathsf{e}_j \;\Rightarrow\; i < j$.

Firing the events following such topological order guarantees that when an event is fired all its causal predecessors have been already fired. Given an event e ready to fire, if all the events concurrent to e have been already fired, most states in which e is enabled will be already reached. A traversal algorithm in which events are fired following the topological order derived from the causality relations, guarantees a maximum effectiveness.

Given a set of simulation sequences, we propose the following three-step traversal algorithm: (1) Generate the CES for each sequence. (2) Find a topological order for the events in the CES. (3) Execute a symbolic traversal from the initial state by applying each event exactly once. Events will be applied following the topological order extracted from the CES. The states generated after the image computation of one event will be immediately applied as domain for the image computation of the successor event, thus *chaining* the effect.

4. Conclusions

We believe that the incremental analysis of the state space of a system is the key for the success of traversal algorithms. We have proposed an two-step hybrid traversal strategy that combines simulation and guided-traversal. Simulation provides information about the causality between events. Information about good chaining order can be extracted from it. The combination of both allows to guide traversal and minimize BDD sizes and execution times.

References

[1] A. Arnold. *Finite Transition Systems*. Prentice-Hall, 1994.

[2] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers. *IEICE Transactions on Information and Systems*, E80-D(3):315–325, 1997.

[3] A. S. Miner and G. Ciardo. Efficient reachability set generation and storage using decision diagrams. In *International Conference on Application and Theory of Petri Nets*, pages 6–25, 1999.

[4] M. Nielsen, G. Plotkin, and G. Winskel. Petri Nets, Event Structures and Domains. *Theoretical Computer Science*, 13:85–108, 1981.

[5] E. Pastor, J. Cortadella, and O. Roig. Symbolic analysis of bounded petri nets. *IEEE Transactions on Computers*, 50(5):432–448, 2001.

[6] O. Roig, J. Cortadella, and E. Pastor. Verification of asynchronous circuits by bdd-based model checking of petri nets. In *International Conference on Application and Theory of Petri Nets*, pages 374–391, 1995.