

# Modeling and Verification of Iterated Systems and Protocols

**Lubomir Ivanov**

Department of Computer Science  
Iona College  
715 North Avenue  
New Rochelle, NY 10801  
[livanov@iona.edu](mailto:livanov@iona.edu)

**Ramakrishna Nunna**

Department of Electrical and Computer  
Engineering  
California State University Fresno, MS 94  
Fresno, CA 93740  
[rnunna@csufresno.edu](mailto:rnunna@csufresno.edu)

## ABSTRACT

*The high complexity of modern hardware systems necessitates the use of formal methods for checking the satisfaction of desired properties and the absence of design flaws. Some powerful methods such as model checking and the  $\omega$ -automata approach have found wide acceptance, but suffer from the "state explosion" problem. To avoid this issue we recently proposed a new formal verification method based on series-parallel posets. The technique is applicable to verifying the proper sequencing of events occurring in non-iterated, as well as globally-iterated/locally-non-iterated systems. In this paper we extend the series-parallel poset verification to handle the much broader class of general iterated systems. This allows us to model and verify the behavior of systems involving feedback on multiple levels, as well as the behavior of communication, interconnect, and cache coherence protocols. The verification algorithms retain a low-order polynomial space- and time complexity.*

## INTRODUCTION

The ever-increasing complexity of hardware systems and the protocols which govern their behavior has greatly increased the likelihood of design errors, and, at the same time, limited the usefulness of the classical simulation and testing methods for uncovering design faults. As a result the field of formal verification has rapidly developed into a major research effort to find more accurate and reliable methods for proving the correctness of hardware designs. An excellent overview of the field of formal verification can be found in [1]. Some powerful formal verification methods such as *Symbolic Model Checking* [1] and  *$\omega$ -Automata Verification* [2] have become extremely popular, and have led to the development of industrial-level verification tools (SMV, FormalCheck, etc.). Unfortunately, the higher expressiveness of a method usually leads to a higher space- and time complexity of the associated verification algorithms. Model Checking and several other methods are plagued by the "state explosion" problem – the fact that explicitly representing the system model requires an exponential number of states in the number of system variables. To avoid "state explosion" and other related issues, a number of new verification methods have emerged, which, while relatively less powerful and general, guarantee a significantly improved efficiency. Among these, several methods have been based on using partial orders to describe the dependence or independence of sets of events occurring in a hardware system [3, 4, 5, 6, 7]. The main appeal of

using partial orders in modeling and verifying system behavior is in avoiding the study of all possible interleavings of events occurring during a run of the system. In addition, partial order models are usually very clear and intuitive, and the verification algorithms can be fully automated. In some cases, partial order methods reduce the complexity of verifying properties of asynchronous circuits from exponential to polynomial. In [8] we introduced a new formal verification method for proving timing properties of non-iterated systems. The method is based on the inductively defined notion of series-parallel posets. The associated verification algorithms are characterized by a low-order polynomial time- and space complexity. The method was expanded [9] to allow the modeling and verification of globally-iterated/locally-non-iterated systems. In [10], we introduced a reduction methodology, which further improved the complexity of the verification algorithms.

In this paper we explore the issues of verifying the correct sequencing of events occurring in an iterated system. We begin with a brief introduction to series-parallel posets. Next we discuss the issues of event dependence and independence, and briefly mention some prior results related to verifying properties of non-iterated- as well as globally iterated/locally non-iterated systems. The core of the paper presents the iterated systems verification methodology and the associated verification algorithm. The complexity of the algorithm is briefly considered. Finally, the new method is placed in the context of other related work, and some strengths and weaknesses are discussed. The modeling and verification capabilities of our approach have been demonstrated by the verification of a handshaking protocol and the PCI bus protocol [12], and the MESI cache coherence protocol [13].

## SERIES PARALLEL POSETS

A *partially ordered set (poset)* is a set with a reflexive, antisymmetric, and transitive relation defined on the set elements. A  $\Sigma^*$ -labeled poset  $P=(P, \leq, l)$  consists of a poset  $(P, \leq)$ , and an assignment of a nonempty word (a label)  $l(v) \in \Sigma^*$  to each vertex  $v$  in  $P$ . Given posets  $P$  and  $Q$  with  $P \cap Q = \emptyset$ , we define two operations on labeled posets:

**Concatenation ( $\bullet$ ):**  $P \bullet Q := (P \cup Q, \leq_{P \bullet Q})$

**Shuffle ( $\otimes$ ):**  $P \otimes Q := (P \cup Q, \leq_{P \otimes Q})$ ,

where:  $v \leq_{P \bullet Q} v' \Leftrightarrow v \leq_P v' \vee v \leq_Q v' \vee (v \in P \wedge v' \in Q)$   
 $v \leq_{P \otimes Q} v' \Leftrightarrow v \leq_P v' \vee v \leq_Q v'$

A *Series-Parallel Poset (SPP)* over  $\Sigma$  is defined inductively:

- The empty poset,  $I$ , is a SPP
- For each  $\sigma \in \Sigma$ , the singleton labeled  $\sigma$  is a SPP
- If  $P$  and  $Q$  are SPPs, so are  $P \bullet Q$  and  $P \otimes Q$

The set of all series parallel posets formed from  $I$  and the singletons and closed under concatenation ( $\bullet$ ) and shuffle ( $\otimes$ ) forms a bimonoid denoted  $SP(\Sigma^*)$ . Further details can be found in [11]. For our purposes, the alphabet,  $\Sigma$ , will consist of all distinct events occurring during a run of the system under consideration. Let each event,  $e_i$ , occurring in a system be represented by a singleton poset,  $e_i$ . Then, the fact that event  $e_i$  precedes event  $e_j$  is represented by  $e_i \bullet e_j$ . The independence of events  $e_i$  and  $e_j$  is represented by  $e_i \otimes e_j$ . This extends naturally to sets of events.

*When are two sets of events dependent or independent?*

Two sets of events,  $P$  and  $Q$ , are *independent* if no event in  $P$  triggers a chain of events leading to the occurrence of an event in  $Q$  and v.v., i.e.  $P$  and  $Q$  are independent if the set of predecessors of  $P$  does not involve any event from  $Q$  and v.v.

A set of events  $P$  *always precedes* a set of events  $Q$  if all events in  $P$  occur before any event in  $Q$  does, i.e. if each event in  $Q$  has all events in  $P$  as predecessors.

A set of events  $P$  *partially precedes* a set of events  $Q$  if  $P$  sometimes occurs before  $Q$ . This so when each event in  $Q$  has at least one predecessor from  $P$ , or when  $P$  and  $Q$  are independent.

Let us illustrate the definitions with an example. Consider the following series-parallel poset:

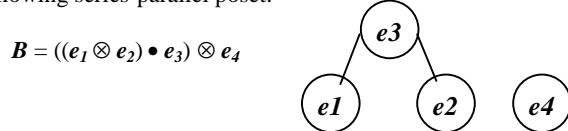


Fig.1 A Simple Series Parallel Poset Example

Consider now the sets of events  $P_1 = \{e_1, e_4\}$ , and  $P_2 = \{e_3\}$ . Clearly,  $P_1$  and  $P_2$  are not independent since  $e_1$  must occur before  $e_3$  does.  $P_1$  does not always precede  $P_2$  since one possible sequence of events is  $e_1, e_2, e_3, e_4$ . But  $P_1$  may sometimes precede  $P_2$  since another possible sequence is, for example,  $e_1, e_2, e_4, e_3$ .

Interpreting series-parallel posets as descriptions of the dependence or independence of sets of events allows us to model the *behavior* of a system in terms of the sequences of events occurring during its operation. In [8] we presented an approach to modeling the behavior and properties of non-iterated systems with series-parallel posets. A *non-iterated system* is one, in whose events are distinct and not repeated.<sup>1</sup>

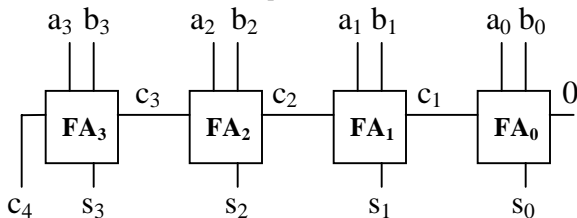


Fig.2 A Non-Iterated System: 4-bit Binary Adder

<sup>1</sup> Not all non-iterated systems can be expressed with series-parallel posets. See the section on Contributions and Limitations.

We presented an algorithm, which can be used to verify that a particular property is always satisfied or sometimes satisfied within a given behavior. In [9], the methodology was further expanded to deal with *globally iterated/locally non-iterated systems*. These are systems, which consist of non-iterated sub-systems operating either in series or in parallel, but such that the global system output is fed back for another iteration.

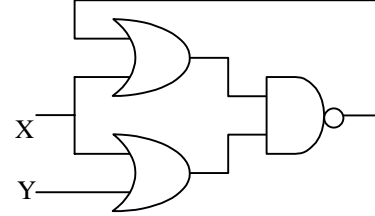


Fig.3 A Simple Globally-Iterated/Locally Non-Iterated System

In both cases, the verification algorithms have a low-order polynomial time- and space complexity, which is further improved with the introduction of the behavior reduction methodology presented in [10].

## MODELING OF ITERATED SYSTEMS

An *iterated system* is one in which some or all events are repeated. Thus, an iterated system consists of a number of components, which function in series or independently so that each component is either an iterated- or a non-iterated system. A wide variety of systems can be considered iterated:

- Communication-, Interconnect, or Cache Protocols
- Asynchronous Sequential Circuits
- Feedback Control Systems, etc.

Concrete examples of iterated systems to which we have applied our methodology are the *Peripheral Component Interconnect (PCI)* bus protocol, which allows 32-bit or 64-bit high-speed data transfers between devices, and the *Modified/Exclusive/Shared/Invalid (MESI)* cache coherence protocol used to synchronize the operation of cache controllers in shared-memory MIMD systems, as well as to maintain the consistency between the level-1 and level-2 caches of the Intel Pentium® microprocessor [12, 13]. In this paper for the sake of illustration, we present a simple example of an iterated system at the logic gate level:

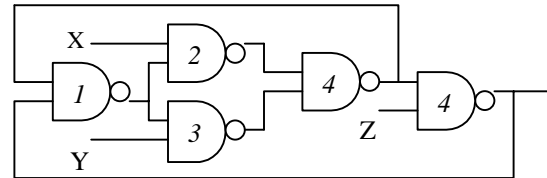


Fig.4 A Simple Iterated System (gate level)

The notion of a series-parallel poset is not sufficient to describe the behavior of a system with iteration. Hence, we introduce a new structure - the star shuffle semiring  $S = (S, +, \bullet, \otimes, *, \theta, I)$  of series-parallel posets, defined as follows:

- $S$  - the set of finite subsets of  $SP(\Sigma^*)$ , closed under the semiring operations
- If  $K \in S$  and  $L \in S$ ,  $K+L = \{P \mid P \in K \vee P \in L\} \in S$
- If  $K \in S$  and  $L \in S$ ,  $K \bullet L = \{P \bullet Q \mid P \in K \wedge Q \in L\} \in S$
- If  $K \in S$  and  $L \in S$ ,  $K \otimes L = \{P \otimes Q \mid P \in K \wedge Q \in L\} \in S$

- If  $K \in S$ , then  $K^* = I + K + K \bullet K + \dots = \sum_{i=0, \dots, \infty} K^i \in S$ , where  $K^i = K \bullet K \bullet \dots \bullet K$ ,  $i$  times.
- $\emptyset$  is the empty set of posets
- $I$  is the empty poset

We define the *behavior*,  $B$ , of an iterated system to be an element of the star-shuffle semiring  $S$ , i.e.  $B \in S$ . Thus, the behavior of an iterated system is a set of series-parallel posets. For example, if we denote the event “gate  $i$  produces a valid output” by  $e_i$ , then the behavior of the system in Fig.4 is given by the expression:

$$B = ((e_1 \bullet (e_2 \otimes e_3) \bullet e_4)^* \bullet e_5)^*$$

We can represent the *verification properties* as sets of series-parallel posets as well, i.e.  $P \in S$ . Unlike behaviors, however, properties will usually be defined over a subset of the alphabet  $\Sigma$ , since we are most often interested in the mutual dependence or independence of a relatively small subset of system events. For example the property “Gates 2 and 3 produce valid outputs independent of each other, but gate 4 depends on both gates 2 and 3” is given by the expression

$$P = (e_2 \otimes e_3) \bullet e_4.$$

The *verification questions* are specified as *predicates* over sets of series-parallel posets. These predicates are:

- $SS(B, P)$  is a binary predicate, which we shall read as “The property  $P$  is sometimes satisfied within the behavior,  $B$ ”. The predicate takes a behavior and a property and verifies that  $P$  can sometimes be traced within the behavior,  $B$ , of the system.
- $AS(B, P)$  is a binary predicate, which we shall read as “The property  $P$  is always satisfied within the behavior,  $B$ ”. The predicate takes a behavior and a property and verifies that  $P$  can always be traced within  $B$ .

The behavior can be in one of four standard forms:

- Concatenation:  $B = B_1 \bullet B_2 \bullet \dots \bullet B_n$
- Shuffle:  $B = B_1 \otimes B_2 \otimes \dots \otimes B_n$
- Plus:  $B = B_1 + B_2 + \dots + B_n$
- Star:  $B = B_1^*$

The corresponding property forms are:

- Concatenation:  $P = P_1 \bullet P_2 \bullet \dots \bullet P_m$
- Shuffle:  $P = P_1 \otimes P_2 \otimes \dots \otimes P_m$
- Plus:  $P = P_1 + P_2 + \dots + P_m$
- Star:  $P = P_1^*$

### Reduction of the Behavior

In [10] we introduced the notion of a *reduction* of the behavior of a non-iterated system. The reduction is prompted by the fact that, while the behavior of a system may involve hundreds of thousands of distinct events, in most cases the verification of a property involves testing for the mutual dependence or independence of only a few events. We shall introduce a similar reduction for the case of general iterated systems. To perform the reduction, we need a *projection function*  $Pr(B, set(P))$ , which takes a behavior,  $B$ , and a set of events, and returns a reduced behavior,  $B'$ , with respect to the events in the set. The projection function is a mapping  $Pr: S \times \wp(\Sigma) \rightarrow S$ , where  $S$  is the set of all behaviors over an alphabet  $\Sigma$ , and  $\wp(\Sigma)$  is the set of all subsets over  $\Sigma$ .

Function  $Pr$  is defined recursively as follows:

$$\begin{aligned} Pr(A, \emptyset) &= I \\ Pr(e_i, E) &= \begin{cases} e_i & \text{if } e_i \in E \\ I & \text{otherwise} \end{cases} \\ Pr(A \bullet B, E) &= Pr(A, E) \bullet Pr(B, E) \\ Pr(A \otimes B, E) &= Pr(A, E) \otimes Pr(B, E) \\ Pr(A + B, E) &= Pr(A, E) + Pr(B, E) \\ Pr(A^*, E) &= (Pr(A, E))^* \end{aligned}$$

where  $A, B \in S$ , and  $E \in \wp(\Sigma)$ . The effect of the projection function is to substitute  $I$  in place of all events *not in the set*  $E$ . Notice that the projection function does not modify the ordering of events in the behavior. It merely “compresses” the behavior to include only the events whose temporal ordering we want to check with respect to the verification property.

As an example consider the reduction of the behavior,  $B$ , below with respect to the events in the property,  $P$ :

*Behavior:*

$$B = (((e_1 \otimes e_2)^* \bullet (e_3^* \otimes e_4) \bullet e_5^*)^* \otimes (e_6 \bullet e_7)^*)^*$$

*Property:*

“Event  $e_1$  is repeated independently of  $e_7$ 's repetitions.”

$$P = e_1^* \otimes e_7^*$$

*Reduced Behavior:*

$$B' = Pr(B, \{e_1, e_7\}) = (e_1^* \otimes e_7^*)^*$$

In our further discussions, we shall assume that the verification predicates are defined over the reduced behavior, rather than the complete system behavior. This improves the efficiency (time- and space complexity) of the verification algorithm, and simplifies reasoning about events and their mutual dependence and independence under iteration.

### The Verification Predicates

Based on a number of theorems, corollaries, and lemmas, which examine the satisfaction of all forms of properties with respect to all forms of behaviors, we derive the following definitions of the two verification predicates  $SS(B, P)$  and  $AS(B, P)$  for iterated systems:<sup>2</sup>

$SS(B, P)$  iff

- $P = e \wedge (B = e \vee B = e^*)$
- $P = P_1^* \wedge B = B_1 \otimes B_2 \otimes \dots \otimes B_n \wedge SS(B, P_1) \wedge \forall i \in [n] B_i = B_{i1}^*$
- $P = P_1^* \wedge B = B_1^* \wedge SS(B, P_1)$
- $P = P_1 \otimes P_2 \otimes \dots \otimes P_m \wedge B = B_1^* \wedge SS(B_1, P)$
- $P = P_1 \otimes P_2 \otimes \dots \otimes P_m \wedge B = B_1 \otimes B_2 \otimes \dots \otimes B_n \wedge ((P \in SP(\Sigma^*) \wedge SS\_NI(NI(B), P)) \vee (P \notin SP(\Sigma^*) \wedge \forall i \in [m] SS(B, P_i)) \wedge \forall i \in [m-1] \mathbf{Independent}_B(P_i, P_{i+1}) \wedge \forall i \in [m] (P_i = (P_{i1})^* \rightarrow \forall e \in P_i L(set(lisc(B, e))) \subseteq L(set(P_i))))$
- $P = P_1 \bullet P_2 \bullet \dots \bullet P_m \wedge (B = B_1 \bullet B_2 \bullet \dots \bullet B_n \vee B = B_1 \otimes B_2 \otimes \dots \otimes B_n \vee B = B_1^*) \wedge ((P \in SP(\Sigma^*) \wedge SS\_NI(NI(B), P)) \vee (P \notin SP(\Sigma^*) \wedge \forall i \in [m] SS(B, P_i) \wedge \forall i \in [m] (P_i = (P_{i1})^* \rightarrow \forall e \in P_i L(set(lisc(B, e))) \subseteq L(set(P_i)))) \wedge \forall i \in [m-1] (\forall e \in P_{i+1} L(pred_B(\{e\})) \cap L(set(P_i)) \neq \emptyset \vee \mathbf{Independent}_B(P_i, P_{i+1}))))$
- $B = B_1 + B_2 + \dots + B_n \wedge \exists i \in [n] SS(B_i, P)$
- $P = P_1 + P_2 + \dots + P_n \wedge \exists i \in [n] SS(B, P_i)$

<sup>2</sup> There are over 30 theorems, corollaries, and lemmas, and we cannot present them within the specified paper length limits.

$AS(\mathbf{B}, P)$  iff

- $P = e \wedge B = e$
- $P = P_1^* \wedge B = B_1 \otimes B_2 \otimes \dots \otimes B_n \wedge AS(\mathbf{B}, P_1) \wedge \forall i \in [n] B_i = B_{i1}^*$
- $P = P_1^* \wedge B = B_1^* \wedge AS(\mathbf{B}_1, P_1)$
- $P = P_1 \otimes P_2 \otimes \dots \otimes P_m \wedge B = B_1^* \wedge AS(\mathbf{B}_1, P)$
- $P = P_1 \otimes P_2 \otimes \dots \otimes P_m \wedge B = B_1 \otimes B_2 \otimes \dots \otimes B_n \wedge P \notin SP(\Sigma^*) \wedge \forall i \in [m] AS(\mathbf{B}, P_i) \wedge \forall i \in [m-1] \text{Independent}_B(P_i, P_{i+1}) \wedge \forall i \in [m] (P_i = (P_{i1})^* \rightarrow \forall e \in P_i, L(\text{set}(\text{lisc}(\mathbf{B}, e))) \subseteq L(\text{set}(P_i)))$
- $P = P_1 \bullet P_2 \bullet \dots \bullet P_m \wedge (B = B_1 \bullet B_2 \bullet \dots \bullet B_n \vee B = B_1^*) \wedge P \notin SP(\Sigma^*) \wedge \forall i \in [m] SS(\mathbf{B}, P_i) \wedge \forall i \in [m] (P_i = (P_{i1})^* \rightarrow \forall e \in P_i, L(\text{set}(\text{lisc}(\mathbf{B}, e))) \subseteq L(\text{set}(P_i))) \wedge \forall i \in [m-1] (\forall e \in P_{i+1}, L(\text{pred}_B(\{e\})) \cap L(\text{set}(P_i)) = L(\text{set}(P_i)))$
- $B = B_1 + B_2 + \dots + B_n \wedge \forall i \in [n] AS(\mathbf{B}_i, P)$
- $P = P_1 + P_2 + \dots + P_n \wedge \exists i \in [n] AS(\mathbf{B}, P_i)$

In these definitions we used a number of functions and auxiliary predicates:

- Labeling functions:
  - $l: S \rightarrow \Sigma, l(s) = \sigma, s \in S$  and  $\sigma \in \Sigma$
  - $L: \wp(S) \rightarrow \wp(\Sigma), L(\{s_i \mid 0 \leq i < n\}) = \{l(s_i) \mid 0 \leq i < n\}$ ,
 where  $S$  is the set of singleton posets,  $\wp(S)$  the powerset of  $S$ , and  $\Sigma$  the alphabet of system events

- Function  $\text{set}(\mathbf{B})$ :
$$\begin{aligned} \text{set}(\mathbf{B}) &= \emptyset && \text{if } \mathbf{B} = \mathbf{I} \\ \text{set}(\mathbf{B}) &= \{e\} && \text{if } \mathbf{B} = e \\ \text{set}(\mathbf{B}) &= \text{set}(\mathbf{B}_1) && \text{if } \mathbf{B} = \mathbf{B}_1^* \\ \text{set}(\mathbf{B}) &= \text{set}(\mathbf{B}_1) \cup \dots \cup \text{set}(\mathbf{B}_n) && \text{if } (\mathbf{B} = \mathbf{B}_1 \bullet \dots \bullet \mathbf{B}_n \vee \\ & && \mathbf{B} = \mathbf{B}_1 \otimes \dots \otimes \mathbf{B}_n \vee \mathbf{B} = \mathbf{B}_1 + \dots + \mathbf{B}_n) \end{aligned}$$
 Function  $\text{set}(\mathbf{B})$  takes a series-parallel poset expression,  $\mathbf{B}$ , and returns the set of its vertices.

- Function  $\text{pred}_B(\mathbf{P})$ :
$$\begin{aligned} \text{pred}_B(\mathbf{P}) &= \emptyset && \text{if } \mathbf{B} = e \vee \mathbf{B} = \mathbf{I} \vee (\mathbf{B} = \mathbf{B}_1 \bullet \dots \bullet \mathbf{B}_n \wedge e \in \text{set}(\mathbf{B}_1)) \\ \text{pred}_B(\mathbf{P}) &= \cup_{k=1, \dots, (i-1)} \text{set}(\mathbf{B}_k) && \text{if } \mathbf{B} = \mathbf{B}_1 \bullet \dots \bullet \mathbf{B}_n \wedge e \in \text{set}(\mathbf{B}_i), i > 1 \\ \text{pred}_B(\mathbf{P}) &= \text{pred}_{B_i}(\mathbf{P}) && \text{if } (\mathbf{B} = \mathbf{B}_1 \otimes \dots \otimes \mathbf{B}_n \vee \mathbf{B} = \mathbf{B}_1 + \dots + \mathbf{B}_n) \wedge \\ & && e \in \text{set}(\mathbf{B}_i) \end{aligned}$$
 Function predecessor,  $\text{pred}_B(\mathbf{P})$  takes the set of vertices of a series-parallel poset expression  $\mathbf{P}$  and returns the set of predecessors of those vertices in a series-parallel poset expression,  $\mathbf{B}$ , or  $\emptyset$  if the predecessor set is empty.

- Predicate  $\text{Independent}_B(\mathbf{P}, \mathbf{Q}) = \text{TRUE}$  iff
$$\begin{aligned} L(\text{pred}_B(\text{set}(\mathbf{P}))) \cap L(\text{set}(\mathbf{Q})) &= \emptyset \\ \wedge L(\text{pred}_B(\text{set}(\mathbf{Q}))) \cap L(\text{set}(\mathbf{P})) &= \emptyset \end{aligned}$$
 The predicate verifies that two sets of series-parallel posets of events,  $\mathbf{P}$  and  $\mathbf{Q}$ , are independent within the context of a behavior,  $\mathbf{B}$ .

- Function  $NI(\mathbf{B})$  (Non-Iterated):
$$\begin{aligned} NI(\mathbf{I}) &= \mathbf{I} \\ NI(e) &= e \\ NI(\mathbf{B}^*) &= \mathbf{B} \\ NI(\mathbf{B}_1 \bullet \mathbf{B}_2) &= NI(\mathbf{B}_1) \bullet NI(\mathbf{B}_2) \\ NI(\mathbf{B}_1 \otimes \mathbf{B}_2) &= NI(\mathbf{B}_1) \otimes NI(\mathbf{B}_2) \\ NI(\mathbf{B}_1 + \mathbf{B}_2) &= NI(\mathbf{B}_1) + NI(\mathbf{B}_2) \end{aligned}$$
 Function  $NI(\mathbf{B})$  allows us to verify a non-iterated property within a single iteration of an iterated system with behavior  $\mathbf{B}$  by reducing an iterated behavior to a non-iterated one.

- Function  $\text{lisc}(\mathbf{B}, e)$  (least iterated sub-component):
$$\begin{aligned} \text{lisc}(\mathbf{B}, e) &= e && \text{if } \mathbf{B} = e \\ \text{lisc}(\mathbf{B}, e) &= \text{lisc}(\mathbf{B}_i, e) && \text{if } (\mathbf{B} = \mathbf{B}_1 \bullet \dots \bullet \mathbf{B}_n \vee \\ & && \mathbf{B} = \mathbf{B}_1 \otimes \dots \otimes \mathbf{B}_n \vee \mathbf{B} = \mathbf{B}_1 + \dots + \mathbf{B}_n) \wedge e \in \text{set}(\mathbf{B}_i) \\ \text{lisc}(\mathbf{B}, e) &= \mathbf{B} && \text{if } \mathbf{B} = \mathbf{B}_1^* \wedge Pr(\mathbf{B}_1) = Pr(NI(\mathbf{B}_1)) \\ \text{lisc}(\mathbf{B}, e) &= \text{lisc}(\mathbf{B}_1, e) && \text{if } \mathbf{B} = \mathbf{B}_1^* \wedge Pr(\mathbf{B}_1) \neq Pr(NI(\mathbf{B}_1)) \end{aligned}$$
 Function  $\text{lisc}(\mathbf{B}, e)$  returns the least iterated sub-component of an iterated behavior  $\mathbf{B}$ , which contains the event  $e$ , e.g.
$$\text{lisc}(((e_1^* \otimes e_2)^* \bullet e_5^*)^*, e_2) = (e_1^* \otimes e_2)^*$$

The  $AS(\mathbf{B}, P)$ ,  $SS(\mathbf{B}, P)$ , and  $\text{Independent}_B(P, Q)$  predicates serve as the basis of our verification algorithm. The above definitions have been implemented as a software package, which allows the automatic verification of properties related to the sequencing of events occurring in an iterated system.<sup>3</sup>

The analysis of the time- and space requirements of the implemented algorithm shows that the worst-case time complexity is  $O(n+m^3)$ , and the average case time complexity is  $O(n+m^2)$ , where  $n$  is the number of events in the behavior (before the reduction is performed), and  $m$  is the number of events in the property we are attempting to verify. The space complexity after the reduction is  $O(m)$ .

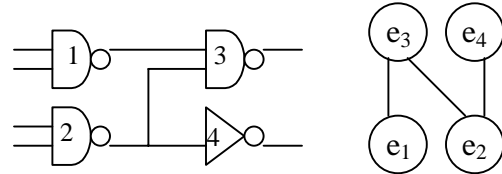
## CONTRIBUTIONS AND LIMITATIONS

Relative to other formal modeling and verification methods, which are aimed at proving safety and liveness properties, we are interested mainly in verifying the proper *sequencing* of events as given by the system specification. The advantage is the very low complexity of the verification algorithms.

The issues of event sequencing and timing has been studied for a long time by many researchers [14, 15, 16, 17, 18, 19]. Closest to our work is that of V.Pratt [4]. There are however important differences. The main stress in [4] is on modeling system behavior - communication channels, in particular - with the help of an extensive collection of operations.

Our technique uses a far smaller collection of operations (concatenation, shuffle, and Kleene star), but models not only system behaviors but properties as well. The emphasis of our work is on verification. In that respect reduced collection of operations simplifies the analysis, and contributes to the efficiency of the algorithms.

Though it has major advantages, one important shortcoming of our technique is the inability to model “N”-type dependencies among the events occurring in a system. These are encountered quite often in real systems and significantly limit the general applicability of our algorithm. Consider the simple example below:



<sup>3</sup> For efficiency, the implementation of some of the functions does not follow their formal inductive definitions above.

If  $e_i$  represent the event “gate  $i$  produces a valid output”, then the event dependency diagram has the “N”-shape described on the right. This type of dependency cannot be modeled only with shuffle and concatenation. Currently we are working on extending our verification methodology to deal with this type of event dependencies as well.

## CONCLUSIONS

In this paper, we outlined a new approach to modeling and formal verification of general iterated systems based on the inductively defined notion of series-parallel posets. The method is applicable to verifying properties of hardware systems involving feedback on multiple levels, as well as various communication-, interconnect-, and cache coherence protocols. The low time- and space complexity of the verification algorithms makes them readily applicable to larger real-world systems.

We have used the implemented software package, based on the presented theory, to verify (among other things) the behavior of the PCI interconnect bus protocol and the MESI cache coherence protocol. In addition we have already submitted a paper in which the above methodology has been applied to modeling and verification of microcoded control for a RISC microprocessor. Work on verifying properties of other large-scale commercial designs is currently underway.

In addition, recent research efforts include extending the series-parallel poset technique to modeling and verifying the behavior of parallel systems and protocols. Finally, a concentrated effort is aimed at overcoming one of the deficiencies of our technique – the inability to model systems with “N”-type event dependence.

## REFERENCES:

- [1] K. McMillan, “*Symbolic Model Checking*”, Kluwer Academic Publishing, 1993
- [2] R.Kurshan, “*Computer Aided Verification of Coordinating Processes: The Automata-Theoretic Approach*”, Princeton Series in Computer Science, Princeton, 1994
- [3] M.Nielsen, G.Plotkin, and G.Winskel, “*Petri nets, event structures, and domains*”, part I, Theoretical Computer Science, 1981
- [4] V.Pratt, “*Modeling Concurrency with Partial Orders*”, International Journal of Parallel Programming, 1986
- [5] P. Godefroid, “*Partial Order Methods for the Verification of Concurrent Systems: an Approach to the State Explosion Problem*”, Doctoral Dissertation, University of Liege, 1995
- [6] R. Nalumasu, G. Gopalakrishnan, “*A New Partial Order Reduction Algorithm for Concurrent System Verification*”, IFIP, 1996
- [7] D. Peled, “*Combining Partial Order Reductions with On-the-Fly Model Checking*”, Journal of Formal Methods in Systems Design, 8 (1), 1996
- [8] L.Ivanov, R.Nunna, S.Bloom, “*Modeling and Analysis of Non-Iterated Systems: An Approach Based on Series-Parallel Posets*”, Proceedings of ISCAS'99, Orlando, FL, 1999
- [9] L.Ivanov, R.Nunna, “*Formal Verification with Series-Parallel Posets of Globally-Iterated Locally-Non-Iterated Systems*”, Proceedings of the MWSCAS'99, Las Cruces, NM, 1999
- [10] L.Ivanov, R.Nunna, “*Formal Verification: A New Partial Order Approach*”, Proc. of ASIC/SOC'99, Washington DC, 1999
- [11] S. Bloom, Z. Esik, “*Free Shuffle Algebras in Language Varieties*”, Theoretical Computer Science 163 (1996) 55-98, Elsevier
- [12] L. Ivanov, R. Nunna, “*Modeling and Verification of an Interconnect Bus Protocol*”, Proc. of MWSCAS'00, Lansing, MI, 2000
- [13] L. Ivanov, R. Nunna, “*Modeling and Verification of Cache Coherence Protocols*”, To appear in the proceedings of ISCAS'01, Sydney, Australia, 2001
- [14] D.Dill, “*Trace Theory for Automatic Hierarchical Verification of Speed-Independent Circuits*”, Technical Report 88-119, CS Dept., CMU, 1988
- [15] J.van de Snepscheut, “*Trace Theory and VLSI Design*”, Ph.D. Thesis, CS Dept., Eindhoven University of Technology, 1993
- [16] B.Moszkowski, “*A Temporal Logic for Multilevel Reasoning about Hardware*”, Computer, Feb. 1985
- [17] J.Halpern, Z.Manna, B.Moszkowski, “*A Hardware Semantics based on Temporal Intervals*”, Proc. 19<sup>th</sup> International Colloquium. On Automata, Lang., and Programming, Springer, Vol.54, 1983
- [18] B.Moszkowski, Z.Manna, “*Reasoning in Interval Temporal Logic in : Logic of Programs*”, Springer Lecture Notes, Vol 164, 1983
- [19] J.Allen, “*An Interval-based Representation of Temporal Knowledge*”, Proc. of 7<sup>th</sup> International Joint Conference on AI, Vancouver, Canada, 1981