

Modeling and Analysis of Non-Iterated Systems: An Approach based upon Series-Parallel Posets

Lubomir Ivanov
Department of Computer Science
Stevens Institute of Technology
Hoboken, NJ 07030
livanov@cs.stevens-tech.edu

Ramakrishna Nunna
Department of Electrical and
Computer Engineering
California State University Fresno
Fresno, CA 93740

Stephen Bloom
Department of Computer Science
Stevens Institute of Technology
Hoboken, NJ 07030

Verifying the correct operation of hardware systems is a complicated, multifaceted task, which can be performed on several different levels:

- System architecture level
- Implementation level (IC)
- Logic gates level
- Transistor level, etc.

Each of these levels requires a different approach for verifying the correctness of the system properties. At the logic gate level, for example, one can try and verify that two combinatorial or sequential circuits implement the same logic functions.

At the more abstract architectural level, the individual components of the system are treated as building blocks, and the verification process involves checking their interactions with each other according to a set of pre-specified rules (a hardware protocol).

In this paper, we present a system based upon Series Parallel Posets which can be used to model and analyze the behavior of non-iterated systems.

Series Parallel Posets:

A partially ordered set (poset) is a set with a reflexive, antisymmetric, and transitive relation defined on the set elements.

A Σ^* -labeled poset $P=(P, \leq, l)$ consists of a poset (P, \leq) , and an assignment of a nonempty word (i.e. a label) $l(v) \in \Sigma^*$ to each vertex v in P . We define two operations on labeled posets:

Given posets P and Q with $P \cap Q = \emptyset$,

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

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

where:

$$v \leq_{P \bullet Q} v' \iff v \leq_P v' \vee v \leq_Q v' \vee (v \in P \wedge v' \in Q)$$

$$v \leq_{P \otimes Q} v' \iff v \leq_P v' \vee v \leq_Q v'$$

A Series-Parallel Poset is then defined inductively as follows:

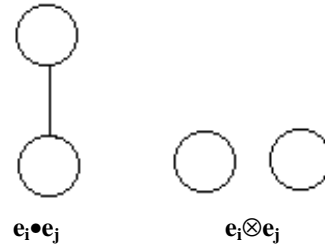
- The empty poset, I , is a SPP
- The singleton posets labeled σ , for each $\sigma \in \Sigma$

- If P and Q are series-parallel posets, so are $P \bullet Q$ and $P \otimes Q$

The set of all series parallel posets, denoted $SP(\Sigma^*)$, with concatenation and shuffle, forms a bimonoid, with identity the empty poset I . Bloom and Esik proved that $SP(\Sigma^*)$ is freely generated in the variety of all bimonoids by the set Σ^* .

For our purposes, the intuitive interpretation of series-parallel posets will be as follows:

Let each event, e_i , occurring in a system be represented by a singleton poset, e_i . If event e_i follows event e_j , then $e_i < e_j$, and can be represented as $e_i \bullet e_j$. If two events e_i and e_j occur independently, then they can form the series-parallel poset $e_i \otimes e_j$. Pictorially one can represent the two situations as follows:



Thus, with the help of SPPs, one can describe inductively an arbitrary arrangement of consecutive and independent events occurring in a combinatorial system.

By enriching the set $SP(\Sigma^*)$ with two more operations - union, $+$, and Kleene star, $*$ - we obtain 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 $L \in S$, then $K + L = \{P \mid P \in K \vee P \in L\}$
- If $K \in S$ and $L \in S$, then $K \bullet L = \{P \bullet Q \mid P \in K \wedge Q \in L\}$
- If $K \in S$ and $L \in S$, then $K \otimes L = \{P \otimes Q \mid P \in K \wedge Q \in L\}$
- If $K \in S$, then $K^* = I + K + K^2 + \dots = \sum_{i=0, \dots, \infty} K^i$

- \emptyset is the empty set of posets
- I is the empty poset

This new structure is powerful enough to describe the behaviors of sequential systems, with $*$ acting as iteration.

Thus, we define a *behavior* $B \in S$ of a system to be a set of series parallel posets generated from the singleton event posets with the help of the operations concatenation, \bullet , shuffle, \otimes , union, $+$, and Kleene star, $*$ as defined above.

Our goal is the formal verification of the operation of hardware systems, so modeling system behavior is only part of the task. We need to be able to express system properties, and verify that these properties are satisfied by the system.

The properties we wish to verify involve the occurrence of events and their mutual dependence or independence in time. Therefore, it makes sense to represent these properties as sets of series-parallel posets over an alphabet of system events as well.

To verify the correctness of these properties, we shall pose the verification questions as well-formed formulas over sets of series-parallel posets. We shall define a few predicates over series-parallel posets which will verify the proper “imbedding” of the specified property poset in the behavior poset of the system.

Before we give any formal definitions, let us discuss the issues on an informal level.

First, notice that there are two basic types of questions that we can ask:

- Is it possible for a set of events to occur in a particular sequence?
- Does a particular sequence of events always occur in the specified order?

Other, more complicated verification questions can be formulated from the ones above using the logical connectives AND, OR, and NOT. For example the question “Is it true that a particular sequence never occurs” can be restated as NOT(question a)).

Let P and Q be two sets of events. The events in each of these two sets may be arbitrarily organized, i.e. some could be independent, while others will depend on each other.

To say that the events in P are independent from the events in Q means that, at no time in the past has any event in the set Q triggered a chain of events leading to an occurrence of an event in P , and vice versa, no event in P is a predecessor of an event in Q .

To say that the events in P always precede the events in Q means that by the time any event in Q is about to occur, all events in P have already occurred. In other words, all events in P are predecessors of each event in Q .

There is, however, one other scenario that we must consider. What if the events in P occur before the events in Q sometimes but not always. This can occur for example when the event sets P and Q are independent or if not all, but only some events in P are necessary for the occurrence of an event in Q . Then we cannot claim that P always precedes Q . We shall refer to this situation as the *partial concatenation* of P and Q . The partial concatenation of two sets of related events P and Q implies that the occurrence of any event in Q must be preceded by an occurrence of one or more events from P .

Verification of Non-iterated Systems

In the case of non-iterated systems, the behavior is a single series-parallel poset. There are two normal forms for series-parallel posets:

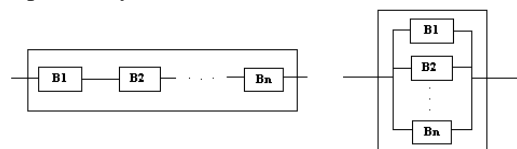
- Concatenation of series-parallel sub-posets:

$$B = B_1 \bullet B_2 \bullet \dots \bullet B_n$$

- Shuffle of series-parallel sub-posets:

$$B = B_1 \otimes B_2 \otimes \dots \otimes B_n$$

The first normal form represents a system whose components function in succession, whereas the second represents a system whose components work independently.



Each system component can itself be represented with a series-parallel poset.

Let us first define the following notions:

- **Immediate Predecessor** : A node a is an immediate predecessor of a node b in a spp B , if $a < b$ and for any $c \in B$, $a < c < b \Rightarrow a = c$.
- **Immediate Successor** : A node b is an immediate successor of a node a in a spp B if a is an immediate predecessor of b .
- **Predecessor**: A node a is a predecessor of a node b in B if $a < b$.
- **Successor** : A node b is a successor of a node a in B if a is a predecessor of b in B .

The set of all immediate predecessors of a node a in a series-parallel poset B will be denoted by

imm_pred_B(a). The set of all immediate successors of a node **a** will be denoted by **imm_succ_B(a)**. The set of all predecessors of a node **a** will be denoted by **pred_B(a)**. The set of all successors of a node **a** will be denoted by **succ_B(a)**.

Now, let **B** be a behavior series-parallel poset, and **P** be a property series-parallel poset. We shall use the following verification predicates:

- a) **AS(B, P)**, which we shall read as “Property **P** is always satisfied within the behavior **B**”, is a binary predicate which takes two series-parallel posets and verifies that the events in **P** always occur in the specified order within the system behavior, **B**.
- b) **SS(B, P)**, which we shall read as “Property **P** is sometimes satisfied within the behavior **B**”, is a binary predicate which takes two series-parallel posets and verifies that the events in **P** can sometimes occur in the specified order within the system behavior, **B**.

Let **S** be the set of singleton posets, **P(S)** - the powerset of **S**, and Σ - the alphabet of system event labels. We define the following two labeling functions:

- a) $l: S \rightarrow \Sigma, l(s) = \sigma, s \in S \text{ and } \sigma \in \Sigma$
- b) $L: P(S) \rightarrow P(\Sigma), L(\{s_i \mid 0 \leq i < n\}) = \{l(s_i) \mid 0 \leq i < n\}$

We also define the function predecessor, $pred_B(P)$, which takes the set of vertices of a series-parallel poset **P** and returns the set of predecessors of those vertices in a series-parallel poset **B**, or \emptyset if the set is empty.

Finally we define the function $set(P)$, which takes a series-parallel poset **P** and returns the set of vertices of that poset.

Now, we shall define one additional predicate **Independent_B(P, Q)** which takes two series-parallel posets and verifies their independence within the context of a series-parallel poset **B**. The formal definition of the predicate is:

Independent_B(P, Q) is TRUE iff

$$L(pred_B(set(P))) \cap L(set(Q)) = \emptyset \quad \wedge$$

$$L(pred_B(set(Q))) \cap L(set(P)) = \emptyset$$

Formally the two verification predicates are defined as follows:

- a) **AS(B, P)** is TRUE iff :
 - $|P| = 1 \wedge l(P) \in \Sigma_B$
 - $P = P_1 \bullet P_2 \bullet \dots \bullet P_n \wedge$
 $\forall i \in [n] [\text{AS}(B, P_i)] \text{ is TRUE} \quad \wedge$
 $\forall i \in [n-1] [\forall e \in P_{i+1} (L(pred_B(\{e\})) \cap L(set(P_i))) = L(set(P_i))]$

- $P = P_1 \otimes P_2 \otimes \dots \otimes P_n \quad \wedge$
 $\forall i \in [n] [\text{AS}(B, P_i)] \text{ is TRUE} \quad \wedge$
 $\forall i \in [n-1] [\text{Independent}_B(P_i, P_{i+1}) \text{ is TRUE}]$

b) **SS(B, P)** is TRUE iff :

- $|P| = 1 \wedge l(P) \in \Sigma_B$
- $P = P_1 \bullet P_2 \bullet \dots \bullet P_n \wedge$
 $\forall i \in [n] [\text{SS}(B, P_i)] \text{ is TRUE} \quad \wedge$
 $\forall i \in [n-1] [\forall e \in P_{i+1} (L(pred_B(\{e\})) \cap L(set(P_i))) \neq \emptyset \vee \text{Independent}_B(P_i, P_{i+1}) \text{ is TRUE}]$
- $P = P_1 \otimes P_2 \otimes \dots \otimes P_n \quad \wedge$
 $\forall i \in [n] [\text{SS}(B, P_i)] \text{ is TRUE} \quad \wedge$
 $\forall i \in [n-1] [\text{Independent}_B(P_i, P_{i+1}) \text{ is TRUE}]$

These definitions capture the intuitive interpretation of concatenation, partial concatenation and shuffle of two sets of events. They are the basis for our verification algorithm.

As we indicated earlier the verification questions about the proper embedding of a property poset within the behavior poset of system will be specified as well-formed formulas (wffs) over series parallel-posets (terms).

The two predicates defined above will be our atomic formulas. The wffs are obtained from the atomic formulas using the logical connectives \wedge, \vee, \neg .

Thus, the atomic formulas will verify the proper occurrence of the events specified in a property spp **P** within the behavior of a system defined as a spp **B**. The wffs will be used to verify combinations of several properties. For example, if **B** is the behavior of a system, and **P₁** and **P₂** are properties to be verified, as can specify the following verification question:
 $(\text{AS}(B, P_1) \wedge \neg \text{SS}(B, P_2)) \vee (\text{AS}(B, P_2) \wedge \neg \text{SS}(B, P_1))$
which verifies the mutual exclusion of the events in **P₁** and **P₂**.

A verification procedure consists of :

- Step 1: Identifying the normal form of the property spp **P**.
- Step 2: Verifying (recursively) each property sub-poset.
- Step 3: Verifying the pair-wise interconnections between successive property sub-posets.

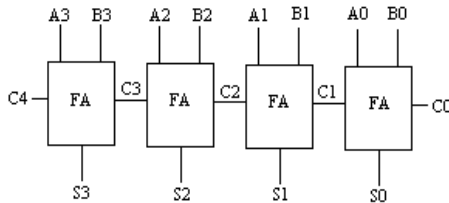
The above procedure always terminates within a finite number of steps. To see this, notice that every time the procedure is called recursively, the property poset it tries to verify is of lesser size than the original property poset that the calling function tries to verify. Since the original poset is of finite size, then within finitely many steps the recursive calls will try to verify property posets with one event only which can be verified

immediately. Thus, step 2 in the above procedure takes only a finite amount of time.

Step 3 will also take a finite amount of time, since each property poset contains only a finite number of sub-posets in one of the two normal forms, and the verification of the interconnections between posets involves the computation of an intersection of two finite sets, which takes a finite number of steps as well. Thus the procedure for verifying a property P within a behavior B always completes in a finite amount of time.

An illustrative example:

Consider a 4-bit binary adder shown below:

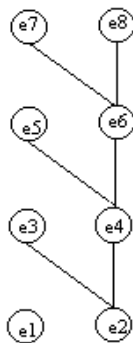


The events we want to represent are the following:

- e_1 - Full Adder 0 produces the sum bit S0
- e_2 - Full Adder 0 produces the carry out bit C1
- e_3 - Full Adder 1 produces the sum bit S1
- e_4 - Full Adder 1 produces the carry out bit C2
- e_5 - Full Adder 2 produces the sum bit S2
- e_6 - Full Adder 2 produces the carry out bit C3
- e_7 - Full Adder 3 produces the sum bit S3
- e_8 - Full Adder 3 produces the carry out bit C4

The relationship between the events are dictated by the structure of the circuit:

- e_3 and e_4 cannot occur before e_2 has occurred
 - e_5 and e_6 cannot occur before e_4 has occurred
 - e_7 and e_8 cannot occur before e_6 has occurred
- This leads to the following behavior series-parallel poset:



$$B = (e_1 \otimes (e_2 (e_3 \otimes (e_4 (e_5 \otimes (e_6 (e_7 \otimes e_8)))))))$$

We can now try to verify some properties:

a) "Are the sum bits S1 and S3 independent of each other?" :

$$\text{Independent}_B(e_3, e_7)?$$

$$L(\text{pred}(\{e_3\})) = \{e_2\} \cap \{e_7\} = \emptyset$$

$$L(\text{pred}(\{e_7\})) = \{e_2, e_4, e_6\} \cap \{e_3\} = \emptyset$$

$\Rightarrow \text{Independent}_B(e_3, e_7)$ is TRUE

b) "Is the sum bit S2 independent of the carry bit C2?"
 $\text{Independent}_B(e_2, e_5)?$

$$L(\text{pred}(\{e_5\})) = \{e_2, e_4\} \cap \{e_2\} = \{e_2\} \neq \emptyset$$

$\Rightarrow \text{Independent}_B(e_2, e_5)$ is FALSE

c) "Does C1 always precede S1?"

$$\text{AS_NI}(B, e_2 \bullet e_3) ?$$

Verify $\text{AS_NI}(B, e_2)$: TRUE

Verify $\text{AS_NI}(B, e_3)$: TRUE

Verify the interconnection $e_2 \bullet e_3$:

$$L(\text{pred}(\{e_3\})) = \{e_2\} \cap \{e_2\} = \{e_2\} \Rightarrow \text{verified}$$

$\Rightarrow \text{AS_NI}(B, e_2 \bullet e_3)$ is TRUE

Conclusion and Future Work

Series Parallel Posets have very powerful system level modeling and analysis capabilities. In this paper, we showed that they can be used to describe the behavior of non-iterated systems. By considering other operations on sets of series-parallel posets, we can obtain structures suitable to study the behavior of more complex systems.

Work is in progress on the development of similar algorithms for verifying *if-then-else* behavior of iterated and non-iterated systems, e.g. systems in which only one of several possible sets of events occurs depending on some condition.

The method described above is general enough to be applied to software and other systems as well. We are currently investigating the application of series-parallel poset to study the behavior of Neural Networks and Cellular Automata.

References:

1. L. Ivanov, "A Series-Parallel Poset Approach to Formal Verification of System Behavior", Ph.D. dissertation, Computer Science Department, Stevens Institute of Technology, May 1999
2. S. L. Bloom and Z. Esik, "Free Shuffle Algebras in Language Varieties", Theoretical Computer Science 163 (1996), 55-98, Elsevier
3. V. Pratt, "Modeling Concurrency with Partial Orders", International Journal of Parallel Programming, 15, 1, c. Nov. 1986