State-Expansion-Based Techniques for Synthesizing Concurrent Protocol Specifications in Distributed Systems

Jehad Al Dallal and Kassem Saleh
Department of Information Science
Kuwait University
P.O. Box 5969, Safat 13060
Kuwait
j.aldallal@ku.edu.kw

Abstract: Several methods have been proposed for synthesizing communication protocol specifications starting from given service specifications. Due to the inherently sequential nature of a finite state machine (FSM), some FSM-based protocol synthesis methods assume that primitives in the service specifications cannot be executed simultaneously. However, other synthesis methods were introduced to handle controlled primitive concurrency by imposing restrictions on the applicable FSM topologies. This paper proposes two alternative FSM-based protocol synthesis methods that eliminate the restrictions on concurrency imposed by earlier methods. The first method applies a sequential-based synthesis method to derive a sequential protocol specification (P-SPEC) from a service specification (S-SPEC) and then applies several state-expansion rules to re-model the resulting P-SPEC to consider the concurrency behavior specified in the S-SPEC. The second method re-models a concurrent S-SPEC into a sequential-like one by expanding its states and applies a sequential-based synthesis method to derive the concurrent P-SPEC. Thus, the paper’s main contribution is proposing synthesis methods that allow the protocol designers to model their service specifications with concurrency behaviors, using FSM-based models, and to derive, automatically, the corresponding protocol specifications for the concurrently executable protocol entities. The derived protocol specifications are guaranteed to be free of design errors; therefore, they do not require any further verification. The complexity of the two methods is discussed and their syntactic and semantic correctness are proven. As an example application, the synthesis method is used to derive the protocol specification of the H.323 call release standard used in Internet calls.

Keywords: distributed systems applications, concurrent protocol, protocol specification, protocol synthesis, service specification.

1. Introduction
A communication protocol is defined as an agreement governing the exchange of information between communicating entities that are typically distributed. A full protocol definition defines a precise format for valid messages (syntax), rules for the orderly and timely exchange of messages (grammar), and a vocabulary of valid messages that can be exchanged along with their meanings (semantics). The protocol design process involves the construction of interacting entities that provide a set of specified services for service users. A bad protocol design may include semantic and syntactic errors. Semantic design errors cause the provision of incorrect services to distributed protocol users. Syntactic design errors may lead to deadlocks in the protocol.

A communication system is most conveniently structured in layers. The Service Access Point (SAP) is the only logical place where a layer can communicate with its surrounding layers or service users. The communication between the layer and its surroundings is performed using Service Primitives (SPs). An interaction is described by the SP and the SAP at which it occurs.
From the user’s viewpoint and at a high level of abstraction, a service is a black box where only the interactions with the user, identified by the SPs, are visible. The specification of the service provided by the layer is defined by the ordering of the visible SPs and is called Service Specification (S-SPEC). At a lower level of abstraction, the service provided by the layer is performed using a number of cooperating protocol entities. The number of protocol entities is equal to the number of SAPs available at the layer. The protocol entities exchange protocol messages through a communication medium. The protocol specification (P-SPEC) prescribes the exchange of messages between the protocol entities. Figure 1 shows the two abstraction levels of a communication layer. Both S-SPEC and P-SPEC can be modeled using FSMs.

![Figure 1. The service and protocol concepts.](image)

In autonomous communication systems, such as the Internet and mobile communication systems (e.g., [1] and [2]), a user can initiate a service at any time. As a result, distributed users at different SAPs may issue simultaneous service primitives, and consequently, it is possible that two or more users will simultaneously issue service requests to each other. This situation leads to the so-called message collision. If the protocol specification is not designed to deal properly with this situation, an unspecified reception error will occur (i.e., the protocol would be at a state where it is unable to handle the message that may arrive).

The construction of a protocol specification from a given service specification is called protocol synthesis. The synthesis approach is used to construct or complete a partially specified protocol design such that the interactions between the constructed or completed protocol entities proceed without encountering any logical error and provide the specified service. In addition, the syntactic correctness of the synthesized protocol is often a direct byproduct of the synthesis method. Several protocol synthesis methods have appeared in the literature, such as those in [3, 4, 5, 6, 7, 8, 14]. The methods introduced in [4, 7] are either not based on the FSM model or only support sequential applications. The method introduced in [5] supports controlled non-sequential applications. The methods introduced in [6, 8, 14] support non-sequential applications that have restrictions on the service specification model topology or on the allowed ordering of service primitives in the service specification.

This paper introduces state-expansion-based automatic synthesis methods for deriving protocol specifications starting from either sequential or concurrent service specifications. Two alternative methods to synthesizing concurrent P-SPECs are proposed and compared. The first method applies the synthesis method introduced in [3] to derive the sequential P-SPEC. Then it applies several state-expansion rules to re-model the sequential P-SPEC to consider the concurrency behavior specified in the S-SPEC. The second method applies the state-expansion rules to re-model the concurrent-based
S-SPEC to a sequential-like S-SPEC. Then it applies the synthesis method introduced in [3] to derive the concurrent P-SPEC. Figure 2 visualizes the two alternative solutions for deriving the concurrent-based P-SPEC from the concurrent-based S-SPEC. Both synthesis methods support uncontrolled concurrent applications and are free of restrictions imposed on the service specification by the earlier methods. The introduced synthesis methods use an FSM-based model for modeling both service and protocol specifications. The paper shows a real application of the proposed synthesis method. The application considers the call release service of the H.323 standard used in Internet calls.

![Figure 2. Two alternative solutions for deriving the concurrent-based P-SPEC.](image)

The rest of the paper is organized as follows. Related research is overviewed in Section 2. Section 3 introduces the models used for service and protocol specifications. Section 4 introduces the synthesis method based on the re-modeling of the sequential protocol. Section 5 introduces the method based on the re-modeling of the S-SPEC and compares it to the first method. An application for the proposed synthesis method is illustrated in Section 6. Finally, Section 7 provides conclusions and discussion of future work.

### 2. Related Research

Two approaches are used in designing communication protocols: analysis and synthesis. In the analysis approach, a sequence of design, analysis, error detection, and correction is applied iteratively to produce error-free design. In the synthesis approach, the protocol design is constructed or completed in such a way that no further validation is needed. Some protocol synthesis methods start the derivation process from a complete service specification [3, 4, 5, 6, 7, 8, 9, 10, 14], and others do not [11, 12]. The protocol synthesis methods can be further classified according to the models used, which include finite state machines [3, 5, 6, 7, 8] and LOTOS-like [9, 10].

Some of the FSM-based service-oriented protocol synthesis methods consider the concurrency behavior of the protocol entities, including [5, 6, 8, 14]. In [5], a concurrent protocol synthesis method is introduced. The synthesis method extends the sequentially based synthesis method introduced by Saleh and Probert in [3]. After applying the three steps introduced in [3], the extended method added transitions to solve the concurrency problem by controlling it such that if a message collision occurs, either all parallel paths have to be re-executed or the parallel path that has the highest priority proceeds, and the events that are not yet executed in the other paths are cancelled. This solution is not practical, because it results in canceling or re-executing some events.

Kakuda et al. [6] introduced a concurrent protocol synthesis method with two restrictions. The first restriction is that the S-SPEC model must have a tree structure. The second is that if a collision...
occurs between two protocol entities, the parallel path that has the higher priority proceeds, and the events that have not been executed yet in the other path are cancelled.

In [8], a concurrent timed protocol synthesis method is introduced with three restrictions. The first restriction is that the S-SPEC model is assumed to be cyclic. The second restriction is that the required service is not allowed to have a specification that may cause a message collision. The last restriction is that if the S-SPEC has a state that has more than one outgoing transition, all the outgoing transitions from that state have to be associated with events that pass through the same SAP.

Finally, in [14], a concurrent protocol synthesis method is introduced. The synthesis method first uses a previously introduced method to synthesize sequentially based protocol specifications. Then the synthesis method re-models the derived protocol specifications by (1) partitioning the set of FSM states and transition involved in concurrency behaviors, (2) replicating the partitioned states and transitions according to a specific algorithm, and (3) adding transitions from and to the replicated states. In contrast with methods introduced earlier, the synthesis method solves the true concurrency problem such that if a message collision occurs, all parallel paths proceed without canceling any event. In addition, the synthesis method eliminates all the restrictions imposed by earlier methods. However, the synthesis method assumes that, in the FSM representing the S-SPEC, if there is a state at which concurrent events occur, the state has to have a unique corresponding state at which the concurrent events merge. Similarly, if there is a state at which concurrent events merge, the state has to have a unique corresponding state at which the concurrent events occur.

In this paper, the introduced synthesis methods solve the concurrency problem, such that if a message collision occurs, all parallel paths proceed without canceling or re-executing any event. In addition, the method introduced here eliminates the restrictions imposed by the methods introduced in [5], [8], and [14]. Therefore, the new methods apply to a wider range of applications.

3. Model Definition
Both service and protocol specifications are modeled using FSM-based models. An FSM consists of states and transitions between them. Due to its inherent sequential nature, the FSM is limited to modeling sequentially based systems. This paper addresses synthesizing protocol specifications from a service specification that includes concurrent behavior. Therefore, in this section, the FSM is extended to model the concurrent behaviors in the service specification. The extended model is called Extended Finite State Machine (EFSM). Each protocol entity is a sequentially based sub-system, and therefore, a traditional FSM can be used without extension to model the protocol entities. In this section, the models used are formally defined in the context of the layered communication system introduced in Section 1.

3.1. Service specification model
The service specification described in the EFSM defines sequences of primitives exchanged between users and processes through the service access points. A service specification includes a concurrent behavior if two or more service primitives pass through different SAPs simultaneously.

Definition 1: A service specification S-SPEC is modeled by an EFSM denoted by a tuple (Sₕ,Tₕ,σ), where:
1. Sₕ is a non-empty finite set of service states. Each state s∈Sₕ is a choice, fork, joint, or leaf state. A choice state is a state that has one or more outgoing transitions, and only one transition is executed at a time. A fork state, denoted by ||, is a state that has two or more outgoing transitions associated with service primitives that pass through different SAPs, and all these outgoing transitions are executed simultaneously. A joint state is a state that has two or more incoming
transitions and one or more outgoing transitions. Finally, a leaf state is a state that does not have any outgoing transitions.

2. $T_s$ is a finite set of transitions, such that each transition $t \in T_s$ is a 3-tuple $<\text{tail}(t), \text{head}(t), \text{SP}>$, where tail$(t)$ and head$(t)$ are, respectively, the tail and the head states of $t$, and SP is the service primitive that defines the service event, its type, and the index of the SAP through which the SP passes, which is denoted by SAP(SP).

3. $\sigma \in S_s$ is the initial service state.

A path $p$ in the ESFM is specified by the sequence of states $(s_i, s_j, ..., s_k)$ traversed through a set of successive transitions, where $s_i =$ successive$(s_{i-1})$. Parallel paths are specified by the sequence $(s_i, s_j, ..., s_k)$, where $s_i$ is a fork state and $s_k$ is either a joint or a leaf state.

Figure 3 shows an S-SPEC example. This example demonstrates a simple data transfer application, consisting of three entities: a server and two machines available at three SAPs. The service described in the S-SPEC is a server-controlled transfer of data between two machines, in which the server keeps a record of the exchanged data. The server and the two machines are processing concurrently. The service is initiated by the server user by issuing a $\text{Next}$ SP downward to the server machine. This service request results in two concurrent upward requests $\text{Next}$ SP at SAP2 and SAP3. The user at SAP2 issues a downward $\text{Data}$ SP. Similarly, the user at SAP3 issues a downward $\text{Data}$ SP. The downward $\text{Data}$ SP at SAP2 will result in two concurrent upward $\text{Data}$ SPs at SAP3 and SAP1 and similarly, the downward $\text{Data}$ SP at SAP3 will result in two concurrent upward $\text{Data}$ SP at SAP2 and SAP1. Then the service cycles back after receiving $\text{Next}$ SP downward from the server user. In this example, if one or both machines have nothing to exchange, they send NULL data message. In this example, $S_s=${$s_1, s_2, s_3, s_4, s_5, s_6, s_7$}, $T_s=$ {$<s_1,s_2, \downarrow\text{Next}_1>, <s_2,s_3, \uparrow\text{Next}_1>, <s_3,s_4, \downarrow\text{Data}_3>, <s_4,s_7, \uparrow\text{Data}_2>, <s_4,s_7, \uparrow\text{Data}_1>, <s_2,s_5, \uparrow\text{Next}_2>, <s_5,s_6, \downarrow\text{Data}_2>, <s_6,s_7, \uparrow\text{Data}_3>, <s_6,s_7, \uparrow\text{Data}_1>, <s_7,s_2, \downarrow\text{Next}_1>, <s_7,s_2, \downarrow\text{Next}_1>$}, and $\sigma=${$s_1$}. For the transition $t=$$<s_1,s_2, \downarrow\text{Next}_1>$, the states $s_1$ and $s_2$ are respectively the tail and head states of $t$, and the SP is $\downarrow\text{Next}_1$. For the SP, the event is Next, SAP(SP)=1, and the type is $\downarrow$. States $s_1, s_3,$ and $s_5$ are choice states, states $s_2, s_4,$ and $s_6$ are fork states, state $s_7$ is a joint state. Paths $(s_2,s_3,s_4,s_7)$ and $(s_2,s_4,s_6,s_7)$ are parallel paths. In addition, there are two parallel paths between each of the states $s_4$ and $s_7$, and $s_6$ and $s_7$. 

![Figure 3. A service specification example.](image-url)
3.2. Protocol specification model
The protocol specification consists of the specifications of the protocol entities that cooperate to provide the service described in the service specification.

**Definition 2:** The protocol entity specification PE-SPEC\(_i\) is modeled by an FSM denoted by a tuple \((S_{pi}, T_{pi}, \sigma_{pi})\), where:
1. \(S_{pi}\) is a non-empty finite set of states of protocol entity \(i\). Each state \(s_{pi} \in S_{pi}\) is an image of one or more S-SPEC states. A state \(s_{pi}\) can be an image of more than one S-SPEC state if two or more S-SPEC states are combined during the protocol synthesis process. A path \(p_{pi}\) in the PE-SPEC\(_i\) is an image of path \(p_{pi}\) in the S-SPEC if each state in \(p_{pi}\) is an image of one or more states in \(p_{pi}\) and each state in \(p_{pi}\) is a pre-image of a state in \(p_{pi}\). A parallel path in PE-SPEC is an image of a parallel path in S-SPEC.
2. \(T_{pi}\) is a finite set of transitions, such that each transition \(t \in T_{pi}\) is a 3-tuple \(\langle\text{tail}(t), \text{head}(t), E_i\rangle\), where \(\text{tail}(t)\) and \(\text{head}(t)\) are, respectively, the tail and the head states of \(t\), and \(E_i\) is a protocol event that can be either: (1) an SP that passes through SAP\(_i\), (2) an SP that passes through SAP\(_i\) and an event message \(E\) sent to PE\(_j\) denoted by \(!e_j\), or (3) an event message \(E\) received from PE\(_j\) denoted by \(?e_j\). The event of the second type is denoted by \(E/!e_j\).
3. \(\sigma_{pi} \in S_{pi}\) is the initial protocol state.

**Definition 3:** In a protocol entity, the parallel paths that have the same image of a fork state are denoted by \(R\), where \(R\) is written using the following BNF language rules:

\[
\begin{align*}
<R> & ::= ([<E>] \cdot[^*]<C>[^*][)] \\
<C> & ::= ||<E> \\
<E> & ::= <E_i> | (([<E>] <D> [)]) \\
<D> & ::= \langle\text{operator}\rangle <E> \\
<\text{operator}> & ::= . | + | ||
\end{align*}
\]

where \(E_i\) is a protocol event. Note that, in BNF, an optional symbol is enclosed in \([\ ]\), and \(or\) is denoted by \(|\). This formal language introduces three types of operators: ".", "+", and "\(\|\)" to represent the operations before, or, and and, respectively. Each of these operators has two operands. The compound term \(A.B\) means that \(A\) has to be executed before \(B\). The compound term \(A+B\) means that either \(A\) or \(B\) is to be executed. Finally, the compound term \(A||B\) means that both \(A\) and \(B\) have to be executed in any order. Figure 4 shows the mapping between the compound terms and the represented FSM structure. The precedence of the operators is (1) ".", (2) ",\(\|\)\", and (3) "+. Using the formal language, the parallel paths in PE-SPEC\(_1\), PE-SPEC\(_2\), and PE-SPEC\(_3\) given in Figure 5 are represented by ((?data\(_3\) . Data) \(||\) (?data\(_2\) . Data)), ((?data\(_3\) . Data/!data\(_1\)) \(||\) (Next . Data/!data\(_3\))), and ((?data\(_2\) . Data/!data\(_1\)) \(||\) (Next . Data/!data\(_2\)))\), respectively.

![Figure 4. Operator types.](image)
Figure 5. The PE-SPECs synthesized from the S-SPEC given in Figure 3.

The same language R can be used to represent the parallel paths outgoing from a fork state in the S-SPEC. For example, the parallel paths in the S-SPEC given in Figure 3 are represented by \(((↑\text{Next}_3 \cdot ↓\text{Data}_3 \cdot (↑\text{Data}_2 \parallel ↑\text{Data}_1)) \parallel (↑\text{Next}_2 \cdot ↓\text{Data}_2 \cdot (↑\text{Data}_3 \parallel ↑\text{Data}_1)))\).

4. Synthesis Method Based on the Re-Modeling of PE-SPEC

The first synthesis method we introduce in this section extends the sequentially based synthesis method introduced in [3] to consider the concurrency behavior of the service specification. In [3], the sequentially based PE-SPECs are obtained by projecting the S-SPEC onto each SAP. Then a set of transition synthesis rules are applied to the transitions of the projected S-SPECs to obtain the sequentially based PE-SPEC. Given the S-SPEC shown in Figure 3, the resulting sequentially based PE-SPECs are shown in Figure 5. In our extension, the states included in the parallel paths of the derived protocol entities are expanded to handle the concurrency behaviors. The algorithm used to expand these states is introduced, and its complexity is analyzed. The syntactic and semantic correctness of our extended synthesis method are provided in Appendix A.

4.1. The first extended synthesis algorithm

To solve the concurrency problem, each parallel path must be re-modeled to contain all possible sequences of events by considering the events in all other parallel paths. For example, either event \(?\text{data}_3\) or \(?\text{data}_2\) in PE-SPEC\(_1\) given in Figure 5 can be executed first. If \(?\text{data}_3\) is executed first, either event Data or \(?\text{data}_2\) is executed second. However, if \(?\text{data}_2\) is executed first, either event Data or \(?\text{data}_3\) is executed second and so on. In other words, the re-modeling problem can be solved by (1) modeling the parallel paths using the language R given in Section 3, (2) applying a set of state-expansion laws to eliminate the presence of the "||" operator from R that represents the parallel paths, and (3) modeling the resulting R using FSM. Figure 6 shows the synthesis algorithm.
4.2. The state-expansion laws

Having more than one outgoing transition from a state in an FSM means that one of these transitions can be executed. When having concurrent PE-SPECs, all such transitions have to be executed if the source state is an image of a fork state. Therefore, it is required to re-model the PE-SPEC to show that all these transitions can be executed. In Step 2.1 of the algorithm, the parallel paths are modeled using the language R. Using this language, three possible operators are used, including $\parallel$. This operator is not modeled in a typical FSM. Therefore, a set of state-expansion laws are introduced to eliminate the presence of "$\parallel$" operator from R that models the parallel path. The laws are listed and proven below and demonstrated in Figures 7 and 8. Note that in these state-expansion laws, un-bold letters represent events, and bold letters represent sequence of events separated by the "." operator.

(a) Interleaving Law 1

(b) Interleaving Law 2

S$_1$ is a fork state
S$_1$ is a choice state and S$_5$ and S$_{15}$ are fork states
**Interleaving Law 1:** $A \parallel B = A \cdot B + B \cdot A$

**Proof:** A protocol entity has a sequential behavior (i.e., no more than one event can be executed at a time). Two events $A$ and $B$ in two different parallel paths of a protocol entity are executed in sequence. Since any of the events can be executed before the other, two sequential interleavings are possible: either $A$ is executed before $B$ or $B$ is executed before $A$. Therefore, the rule is correct. ■

**Interleaving Law 2:** $(A \cdot B) \parallel (C \cdot D) = (A \cdot (B \parallel (C \cdot D))) + (C \cdot (D || (A \cdot B)))$

**Proof:** In this case there are two parallel paths. The first path contains a sequence of events starting with event $A$, and the other path contains a sequence of events starting with event $C$. When the protocol entity reaches the image of the fork state, either $A$ or $C$ is executed first. Note that $A$ and $B$ are in the same path, and therefore, events in $B$ cannot be executed before $A$. Similarly, events in $D$ cannot be executed before $C$. If $A$ is executed first, it is followed by any possible interleaving of events in $B$ with the events in the other parallel path. The possible interleaving of events in $A$ in a sequence of events $B$ is written as $(A \parallel B)$. Similarly, if $C$ is executed first, it is followed by any possible interleaving of events in $D$ with the events in the other parallel path. Therefore, the rule is correct. ■

**Interleaving Result 1:** $A \parallel (C \cdot D) = (A \cdot C \cdot D) + (C \cdot (A|| D))$

**Rationale:** This result is obtained from Interleaving Law 2 by deleting term $B$. ■

**Interleaving Law 3:** $(A + B) \parallel C = A \parallel C + B \parallel C$

**Proof:** In this case there are two possible paths, $A$ and $B$, and either of them is executed in parallel with path $C$. This means that the protocol entity either executes path $A$ or path $B$, in parallel with path $C$, which is similar to executing path $A$ in parallel with path $C$ or path $B$ in parallel with path $C$. ■

**Commutative Law 1:** $A \parallel B = B \parallel A$

**Proof:** The possible interleaving of sequence $A$ consisting of one or more events in a parallel path with sequence $B$ consisting of one or more events in another parallel path is the same as the possible interleaving of sequence $B$ consisting of one or more events in a parallel path in a sequence $A$ consisting of one or more events in another parallel path. Note that the operator "$\parallel"$ represents and, which is typically a commutative operation. ■

**Commutative Law 2:** $A + B = B + A$

**Proof:** The compound term $A + B$ means that either sequence $A$ of events or sequence $B$ of events can be executed, which is the same as $B + A$. Note that the operator "+" represents or, which is typically a commutative operation. ■

**Commutative Law 3:** $A \cdot B \neq B \cdot A$

**Proof:** The compound term $A \cdot B$ means that the sequence $A$ of events is executed first, then the sequence $B$ of events is executed, which is the opposite of $B \cdot A$. Note that operator "." represents before, which is typically not a commutative operation. ■
Figure 8. The commutative laws.

(a) Commutative Law 1

\[ A \parallel B = (A \parallel B) \parallel C = A \parallel (B \parallel C) \]

**Proof**: The compound term \((A \parallel B)\) results in finding a set of all interleaving of sequence \(A\) of events in sequence \(B\) of events. The compound term \((A \parallel B) \parallel C\) results in finding a set of all interleaving of \(s\) sequences of events in sequence \(B\) of events, which is similar to the sequences of events resulting from \(A \parallel (B \parallel C)\). ■

(b) Commutative Law 2

\[ A + B = B + A \]

**Proof**: The compound term \((A + B)\) means that \(A\), \(B\), or \(C\) has to be executed, which is the same as \(A + (B + C)\). ■

(c) Commutative Law 3

\[ A : B = B : A \]

**Proof**: The compound term \((A : B)\) means that \(A\) has to be executed before \(B\), and both \(A\) and \(B\) have to be executed before \(C\). This means that the order of execution is \(A\), then \(B\), and then \(C\), which is the same as the resulting meaning of \(A : (B : C)\). ■

Figure 9 shows an application of the state-expansion laws on \(R\) for each of the parallel paths given in Figure 5.
Figure 9. Applying the state-expansion laws on R for the parallel paths given in Figure 5.

After applying the state-expansion laws, the resulting R is modeled using states and transitions in a straightforward way according to Definition 3. Figure 10 shows the resulting PE-SPECs for the sample example used in this paper after applying Step 2 of the synthesis method given in Figure 6. The resulting PE-SPECs can have redundant states and transitions; therefore, an FSM minimization technique [13] is applied to obtain the reduced PE-SPECs. Figure 11 shows the resulting PE-SPECs for the sample example used in this paper after applying Step 3 of the synthesis method given in Figure 6.

For parallel paths in PE-SPEC₁, assuming A: ?data, B: Data, C: ?data, D: Data, 
\[ R = (A.B||C.D) \]
\[ = (A.(B||C.D)+C.(D||A.B)) \quad \text{Using Interleaving Law 2} \]
\[ = (A.(B.C.D+D.B)+C.(D.A.B+A.(D.B+B.D))) \quad \text{Using Interleaving Result 1} \]
\[ = (A.(B.C.D+C.(B.D+D.B))+C.(D.A.B+A.(D.B+B.D))) \quad \text{Using Interleaving Law 1} \]

The same result applies for PE-SPEC₂, assuming A: ?data, B: Data!/data₁, C: Next, D: Data!/data₂, and PE-SPEC₃, assuming A: ?data, B: Data!/data₁, C: Next, D: Data!/data₂.
4.3. Algorithm and complexity analysis

A formal recursive algorithm given in Figure 12 can be used to implement Step 2 of the synthesis method given in Figure 6. The algorithm applies the state-expansion laws listed and discussed in Section 4.2. Using this algorithm, each parallel path is remodeled into several corresponding paths (e.g., the parallel paths in PE-SPEC1 shown in Figure 5 are remodeled into six paths as shown in Figure 10). Each path includes a possible trace for all transitions in the parallel paths such that the order of the transitions in each parallel path is preserved. Formally, given \( m \) parallel paths, the number of transitions in each of the resulting paths equals \( \sum_{i=1}^{m} n_i \), where \( n_i \) is the number of transitions in the \( i \)th parallel path.

```
//evaluate E_1||E_2
Procedure InterLaw1(E_11,E_12){
    Print ((E_11,E_12)+(E_12,E_11))
}

//evaluate E_1||E_2, where E_2=E_13,E_1y
Procedure InterResult1(E_11,E_12){
    Print ((E_11,E_12,E_1y)+(E_12,InterLaw(E_11,E_1y)))
}

//evaluate E_1||E_2, where E_1=E_13,E_1x and E_2=E_1y,E_2
Procedure InterLaw2(\(E_{x1},E_{y2}\)){
    Print (E_13,(InterLaw(E_1x,E_1y))+(E_1y,InterLaw(E_1x,E_1y)))
}

//evaluate E_1||E_2
Procedure InterLaw(E_1x,E_2y){
    if x=1 and y=1 then  InterLaw1(E_11,E_12)
    else if x=1 then          InterResult1(E_11,E_12)
    else if y=1 then          InterResult1(E_12,E_2x)
    else                     InterLaw2(E_1x,E_2y)
}

//evaluate E_1||E_2||…||E_n
Procedure AssLaw1(E_1x, E_2y, …,E_n){
    if n=2 then   InterLaw(E_1x,E_2y)
    else          InterLaw3(InterLaw(E_1x,E_2y),E_3,…E_n))
}

//evaluate (A_{u1}+…+A_{hn})|E_{x1}|…|E_{xn}
Procedure InterLaw3(A_{u1}…A_{hn},E_{x1}…E_{xn}) {
    if n=1 then
        InterLaw(A_{u1},E_{x1})+ …+InterLaw(A_{hn},E_{x1})
    else
        InterLaw3(InterLaw3(A_{u1}…A_{hn},E_{x1}),E_{x2}…E_{xn})
}
```

Figure 12. Parallel paths transformation algorithm

The algorithm given in Figure 12 remolds \( m \) parallel paths into corresponding paths such that these paths cover all possible traces of the transitions in the parallel paths. The number of these paths, referenced here as \( P_m \), is equal to the number of different combinations of transitions in the parallel paths such that the order of the transitions in each parallel path is preserved. In the case of two
parallel paths, this number is calculated using the following formula [15]:

\[ P_2 = C(n_1+n_2, n_2) = \binom{n_1 + n_2}{n_2} = \frac{(n_1 + n_2)!}{(n_1)!(n_2)!} \quad (1) \]

For example, there are two parallel paths specified in PE-SPEC shown in Figure 5. Each parallel path consists of two transitions. Therefore, the number of paths obtained by applying the algorithm given in Figure 12 equals \( C(2+2, 2) = 6 \), as shown in Figure 10.

Generally, given \( m \) parallel paths, the number of paths \( P_m \), produced using the algorithm given in Figure 13, is formally calculated as follows:

\[ P_m = \frac{(\sum_{i=1}^{m} n_i)!}{\prod_{i=1}^{m} (n_i!)} \quad (2) \]

**Proof:** To prove this formula using mathematical induction, both basic and inductive steps have to be proven as follows.

**Basic Step:** The minimum number of parallel paths is two. In this case, using Formula 2,

\[ P_2 = \frac{(\sum_{i=1}^{2} n_i)!}{\prod_{i=1}^{2} (n_i!)} = \frac{(n_1 + n_2)!}{(n_1)!(n_2)!} = \binom{n_1 + n_2}{n_2}, \]

which is equal to the result obtained using Formula 1 and proven in [15].

**Inductive Step:** In the inductive step, assuming that \( P_m \) is correct, we have to prove that \( P_{m+1} \) is also correct. As discussed earlier in this section, given \( m \) parallel paths, each of the \( P_m \) paths produced using the algorithm given in Figure 12 consists of all transitions in all \( m \) paths, which equals \( \sum_{i=1}^{m} n_i \).

Using Interleaving Law 3, when adding parallel path \( m+1 \), this additional path is executed in parallel with each path \( j \) among the \( P_m \) paths. Using Formula 1, when path \( m+1 \) is executed in parallel with any path \( j \), the number of resulting paths equals \( C(\sum_{i=1}^{m} n_j + n_{m+1}, n_{m+1}) \). As a result, considering all \( P_m \) paths, the total number of produced paths is calculated as follows:

\[ P_{m+1} = P_m \times C(\sum_{i=1}^{m} n_j + n_{m+1}, n_{m+1}) \]

\[ = \frac{(\sum_{i=1}^{m} n_j)!}{\prod_{i=1}^{m} (n_i)!} \times \frac{(\sum_{i=1}^{m} n_j + n_{m+1})!}{\prod_{i=1}^{m} (n_i)! + n_{m+1}} = \frac{(\sum_{i=1}^{m} n_j)!}{\prod_{i=1}^{m} (n_i)!} \times \frac{(\sum_{i=1}^{m} n_j)!}{\prod_{i=1}^{m} (n_i)!} \times \frac{(\sum_{i=1}^{m+1} n_j)!}{\prod_{i=1}^{m+1} (n_i)!} \times \frac{(\sum_{i=1}^{m+1} n_j)!}{\prod_{i=1}^{m+1} (n_i)!} \]

\[ = \frac{(\sum_{i=1}^{m} n_j)!}{\prod_{i=1}^{m} (n_i)!} \times \frac{(\sum_{i=1}^{m+1} n_j)!}{\prod_{i=1}^{m+1} (n_i)!} \]

which is equal to \( P_{m+1} \) found using Formula 2. ■

The complexity of the algorithm given in Figure 12 is bounded by the number of produced transitions. The number of produced transitions is itself bounded by the multiplication result of the
number of resulting paths and the number of transitions in each path (i.e., \( P_m \times \sum_{i=1}^{m} n_i \)).

5. Synthesis Method Based on the Re-modeling of S-SPEC

In the previous section, to solve the concurrency problem, each parallel path in a PE-SPEC is re-modeled to contain all possible sequences of events considering the events in all other parallel paths. The same concept can be applied for the S-SPEC before synthesizing the PE-SPECs from it. This section introduces our second synthesis method that starts by re-modeling the S-SPEC. Then, we compare this method with the one introduced in Section 4.

5.1. The second extended synthesis algorithm

When the state-expansion laws introduced in Section 4 are applied for the parallel paths in the S-SPEC, the resulting S-SPEC models all possible combinations of events resulting from the concurrency behavior. The sequential-like re-modeled S-SPEC can have redundant states and transitions; therefore, an FSM minimization technique [13] is applied to obtain the reduced S-SPEC. The sequential-based synthesis method introduced in [3] can then be applied on the reduced S-SPEC to derive the PE-SPECs. The synthesized PE-SPECs guarantee the capture of the concurrency behavior modeled in the S-SPEC. This is due to the fact that the PE-SPECs are derived from the reduced S-SPEC that models all possible combinations of events resulting from the concurrency behavior. Figure 13 shows the synthesis algorithm.

![Figure 13. The re-modeling S-SPEC-based synthesis algorithm.](image)

5.2. Comparison

Applying the state-expansion rules complicates the topology of the FSM by adding more states and transitions to model the concurrency behavior. Therefore, applying the state-expansion rules on the S-SPEC complicates the application of the synthesis method introduced in [3] since we will be dealing with more states and transitions. In addition, the number and length (i.e., in terms of the number of transitions) of the parallel paths in an S-SPEC are greater than or equal to number and length of the parallel paths in each PE-SPEC. This means that applying the state-expansion rules on the S-SPEC is more difficult than applying them on each of the PE-SPECs. From these two perspectives, the first extended synthesis algorithm introduced in Section 4 is better than the second one introduced in this section.

On the other hand, applying the state-expansion rules on the S-SPEC (i.e., the second algorithm) is better than applying them on the PE-SPECs (i.e., the first algorithm) in the sense that, typically, the total number of parallel paths in the S-SPEC is less than the total number of parallel paths in all of the
distributed PE-SPECs. This means that, typically, the number of times that the state-expansion rules have to be applied to re-model the S-SPEC is less than those that are used to re-model all the PE-SPECs. In addition, in the second algorithm, the FSM minimization process has to be applied only on the S-SPEC, while in the second algorithm, the minimization process has to be applied on every PE-SPEC that includes parallel paths. Therefore, from these two perspectives, the second algorithm introduced in this section is better than the first one introduced in Section 4. Consequently, the preference of one of the introduced algorithms over the other depends on the complexity of the parallel paths in the S-SPEC and PE-SPECs.

For example, consider the S-SPEC that includes three parallel paths, each of them having four transitions. When we apply the second extended synthesis algorithm introduced in this section, the number of transitions produced using the algorithm given in Figure 12 is bounded by:

\[ P_3 \times \sum_{i=1}^{3} n_i = \frac{(4 + 4 + 4)!}{4! \cdot 4!} \times (4 + 4 + 4) = 415800. \]

This means that the synthesis method introduced in [3] has to be applied on this huge number of transitions in the re-modeled S-SPEC to produce the PE-SPECs, in which parallelism is considered. However, when applying the first extended synthesis method introduced in Section 4, the synthesis method introduced in [3] would be applied on 12 transitions (i.e., the summation of the number of transitions included in the parallel paths in S-SPEC). Assume that the application of the synthesis method introduced in [3] on the original S-SPEC results in synthesizing three PE-SPECs such that each of them includes three parallel paths with two transitions each. In this case, in each PE-SPEC, the number of transitions produced by the algorithm given in Figure 12 is bounded by:

\[ P_3 \times \sum_{i=1}^{3} n_i = \frac{(2 + 2 + 2)!}{2! \cdot 2!} \times (2 + 2 + 2) = 720. \]

As a result, the total number of transitions produced in the three PE-SPECs is bounded by 720×3=2160. Note that this number is much smaller than the corresponding number obtained using the second extended synthesis method. This indicates that, in this case, applying the first extended synthesis method is better than applying the second one from the number of transitions point of view. Note that the above complexity analysis is limited to the transitions included in the parallel paths.

### 6. Application

In this section, we demonstrate the application of the synthesis method to the specification of the call release service of the H.323 standard [16] used for the transmission of real-time audio, video, and data communications over packet-based networks. The service includes a gatekeeper G and two endpoints, denoted as P1 and P2, all processing concurrently. The gatekeeper is a central point for all calls within its zone and provides call control services for registered H.323 endpoints. In H.323 standard, the call release is initiated by the endpoint P2 who sends an end session command request ESCR downwards to P1. Then P1 responds to P2 by sending an end session command confirmation ESCC. P2 completes the call release by sending release complete message RC to P1. After that, concurrently, each of the endpoints sends a disconnect request message DRQ to the gatekeeper, which in turn issues a disconnect confirmation message DCF corresponding to each disconnect request. Figure 14 shows the service specification and the corresponding protocol specification derived from the S-SPEC using the proposed synthesis method.
7. Conclusions and Future Work

In this paper, two alternative approaches for the synthesis of concurrent protocol specifications from service specifications are introduced. Both the service and protocol specifications are modeled using FSM-based models. The service specification FSM-based model is extended to model concurrency behaviors. In the first approach, the synthesis method first uses a previously introduced method to synthesize sequential based protocol specifications. Then, the synthesis method applies several state-expansion rules to re-model the derived protocol specifications. Finally, the resulting PE-SPECs are minimized. In the second approach, first the S-SPEC is re-modeled using the state-expansion rules, and then the resulting S-SPEC is minimized. Finally, a sequential based synthesis method is applied to derive the concurrent PE-SPECs. Table 1 summarizes the differences between the proposed method and the existing ones in terms of dealing with the message collision problem and applicability to different S-SPEC topologies. In contrast with methods introduced earlier, the synthesis methods introduced here solve the true concurrency problem, such that if a message collision occurs, all parallel paths proceed without canceling any event. In addition, the synthesis methods introduced here eliminate all the restrictions imposed by the earlier methods. As a result, this paper’s main contribution is its proposal of synthesis methods that are applicable to FSM-based service specifications for autonomous communication systems. The synthesis methods automatically derive error-free FSM-based specifications for concurrent protocol entities. These synthesis methods offer an advantage over the existing methods; they allow the application of our methods to a wider range of concurrent applications.

Figure 14: The S-SPEC and the synthesized PE-SPECs of the call release service of the H.323 standard.
Table 1: Comparison between the proposed and the existing methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Solution for message collision problem</th>
<th>Applicability to SSPEC topologies</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Cyclic</td>
</tr>
<tr>
<td>Saleh and Probert, 1991 [5]</td>
<td>Canceling or re-executing some events</td>
<td>Yes</td>
</tr>
<tr>
<td>Kakuda et al. 1994 [6]</td>
<td>Cancelling events in parallel paths with low priority</td>
<td>No</td>
</tr>
<tr>
<td>Park and Miller, 1997 [8]</td>
<td>Not handling messages causing collision problems</td>
<td>Yes</td>
</tr>
<tr>
<td>Al Dallal and Saleh, 2008 [14]</td>
<td>Executing all events in all parallel paths</td>
<td>Yes</td>
</tr>
<tr>
<td>State-expansion method</td>
<td>Executing all events in all parallel paths</td>
<td>Yes</td>
</tr>
</tbody>
</table>

The preference of one of the introduced approaches over the other depends on the complexity of the parallel paths in the S-SPEC and PE-SPECs. Generally, it is easier to apply the re-modeling S-SPEC technique, especially when the parallel paths in the S-SPEC are not as complex as those in the parallel paths in the PE-SPECs.

The proposed synthesis methods have several limitations, such as not considering communication channel delays and timing requirements that could be provided in the service specifications. These two issues are subjects for future work. In addition, the number of states and transitions in the derived protocol specifications exponentially increases as the number of parallel paths and the number of states in the parallel paths increase. Therefore, the proposed methods are applicable to the service specifications of a relatively low number of parallel paths and states. To solve this problem, we plan to propose a synthesis method for the service specifications modeled in the Unified Modeling Language (UML) statecharts. Such a model allows the representation of specifications that include concurrency behaviors and nested hierarchical states. Allowing nested hierarchical states should considerably reduce the complexity of the derived protocol specifications, in terms of the numbers of states and transitions.

Acknowledgment
The authors would like to acknowledge the support of this work by Kuwait University Research Grant WI03/10.

Appendix A. Proof of correctness
Proving the correctness of the synthesis method requires proving that the synthesis method is syntactically and semantically correct. This proof is necessary to eliminate the need for further validation of the derived protocol specifications. In [3], it is proven that the protocol entities derived using the first step of the synthesis method are syntactically and semantically correct. Step 2 of the synthesis method given in Figure 6 re-models only the paths between the images of the fork and the joint or leaf states. Therefore, before executing outgoing transitions from the images of the fork state, all the protocol entities are proven to be syntactically and semantically correct. In Lemma 2, it is proven that no protocol entity leaves the image of the joint state unless all protocol entities reach the image of the joint state. After reaching the image of the joint state in all protocol entities, the execution of the protocol entities is also proven to be syntactically and semantically correct [3].
Therefore, it must only be proven that the execution of the re-modeled paths between the images of the fork and the joint or leaf states is syntactically and semantically correct.

A.1. Syntactic correctness
Proving the syntactic correctness of the synthesis method requires proving that the derived protocol specifications are free of syntactic design errors, including unspecified reception, deadlock, and livelock.

**Lemma 1.** Step 2 of the algorithm given in Figure 6 prevents the occurrence of unspecified message reception when parallel paths outgoing from a fork state are executed.
**Proof:** In a PE, when any outgoing transition from the fork state is executed, the execution of the path \( p \) that contains the transition begins. However, the PE has to behave as if the execution of the other parallel paths is also starting. As a result, at any state in \( p \), the execution of any yet to be executed transitions, associated with receiving events, in any other parallel path must be allowed. Otherwise, an unspecified reception error occurs. This problem is dealt with using the interleaving laws proven in Section 4. Using the interleaving laws, the state space is expanded such that all possible traces are represented. As a result, the interleaving laws guarantee that each state in \( p \) is a source state for any yet to be executed transition, associated with a receiving event, in any other parallel path. This prevents the occurrence of unspecified message reception at any state in \( p \). The same applies for any state in any parallel path, which means that the interleaving laws applied at Step 2 of the algorithm given in Figure 6 prevents unspecified message receptions when the parallel paths outgoing from a fork state are executed. ■

**Lemma 2.** Step 2 of the algorithm given in Figure 6 ensures that no protocol entity leaves the image of the joint state unless all the protocol entities reach the image of the joint state.
**Proof:** Using the interleaving laws proven in Section 4, each parallel path in each PE is remodeled such that, in the resulting model, each path includes all the transitions in all the other parallel paths. Therefore, the image of the joint state cannot be reached unless all transitions in all parallel paths are executed. This is applied for each PE, which ensures that, in any PE, the image of the joint state is not left unless all transitions in all parallel paths in all PEs are executed. In other words, in any PE, the image of the joint state is not left unless all PEs reach the corresponding images of the joint state. ■

**Lemma 3.** The protocol entities derived using the synthesis algorithm introduced in this paper are free of unspecified reception errors.
**Proof:** Step 2 of the synthesis method given in Figure 6 re-models only the paths between the images of the fork and the joint or leaf states. Therefore, before executing outgoing transitions from the images of the fork state, all the protocol entities are proven to be free of unspecified reception errors [3]. According to Lemma 1, the execution of the re-modeled paths between the images of the fork and the joint or leaf states is free of unspecified reception errors. According to Lemma 2, applying Step 2 of the synthesis algorithm does not affect the execution of transitions after reaching the image of the joint state. After reaching the image of the joint state in all protocol entities, the execution of the protocol entities is also proven to be free of unspecified reception errors [3]. Therefore, the protocol entities derived using the synthesis algorithm introduced in this paper are free of unspecified reception errors. ■

**Lemma 4.** The protocol entities derived using the synthesis algorithm introduced in this paper are free of deadlock errors.
**Proof:** Deadlock errors occur when the protocol is at a non-final state, all channels are empty, and no transmission transition is specified. In other words, deadlock occurs when the protocol is at a state in
which all its outgoing transitions are associated with receiving events and these events are for
messages that are not to be sent by any other protocol entity. This case cannot occur in the extended
paths because each of the original parallel paths (before being modeled) is synthesized using an
algorithm proven in [3] to be free of deadlock errors. This means that each transition associated with
a receiving event in the original parallel paths has a corresponding transition associated with a
sending event, which eliminates the possibility of having a deadlock error. Step 2 of the algorithm
given in Figure 6 remolds the parallel path by interleaving each parallel path by all transitions in all
other parallel paths. Therefore, this step does not introduce any additional transition not represented
before in the original paths. In other words, Step 2 does not add an additional transition, associated
with a receiving event, that does not have a corresponding transition associated with a sending event.
This means that the new transitions added when remodeling the parallel paths do not cause deadlock
errors. The rest of the protocol specification, before the image of the fork state and after the image of
the joint state, is proven in [3] to be free of deadlock errors. As a result, the protocol entities derived
using the synthesis method introduced in this paper are free of deadlock errors.

**Lemma 5.** The protocol entities derived using the synthesis algorithm introduced in this paper are
free of livelock errors.

**Proof:** A livelock error occurs when the protocol entities exchange messages that are meaningless for
the provision of the desired service. There are two types of messages exchanged within the parallel
paths. The first type is a message derived directly from the service specification using the algorithm
introduced by Saleh and Probert (1991a), and therefore these messages are meaningful for the
provision of the desired service. The other type is a message added in Step 2 of the synthesis
algorithm given in Figure 6. All these messages are associated with transitions represented in other
paths. These other paths are derived directly from the service specification using the algorithm
introduced in Saleh and Probert (1991a), and therefore the messages associated with these paths are
also meaningful for the provision of the desired service. As a result, the protocol entities derived
using the synthesis method introduced in this paper are free of livelock errors.

**Theorem 1.** The protocol entities derived using the synthesis algorithm introduced in this paper are
syntactically correct.

**Proof:** Since the protocol entities derived using the synthesis algorithm introduced in this paper are
free of unspecified reception (Lemma 3), deadlock (Lemma 4), and livelock errors (Lemma 5), the
protocol entities are syntactically correct.

**A.2. Semantic correctness**

Proving the semantic correctness of the synthesis method requires proving that the interactions
among the derived protocol entities through a reliable underlying FIFO communication medium
provide the service specified in the S-SPEC. In other words, it must be proven that all possible
orderings of the service primitives, noticed when the protocol entities are executed, are consistent
with the ordering of the service primitives in the S-SPEC.

**Theorem 2.** The protocol entities derived using the synthesis algorithm introduced in this paper are
semantically correct.

**Proof:** The protocol entities derived in Step 1 of the synthesis algorithm given in Figure 6 are proven
to be semantically correct [3]. Step 2 of the algorithm remolds the parallel paths by interleaving each
of the parallel paths with the transitions of the other parallel paths without changing the ordering of
the transitions within each parallel path. Therefore, the order of the service primitives in all of the
remodeled paths is maintained. As a result, the protocol entities derived using the synthesis algorithm
introduced in this paper are semantically correct.
References:
Authors’ Biographies

Dr. Jehad Al Dallal received his PhD degree in Computer Science from University of Alberta in Canada and was granted the best PhD research award. He is currently working at Kuwait University, department of Information Science as an Associate Professor. Dr. Al Dallal completed several research projects in the areas of software testing, software metrics, and communication protocols. In addition, he published more than 50 papers in conference proceedings and ACM, IEEE, IET, Elsevier, Wiley, and other journals. Dr. Al Dallal was involved in developing more than 20 software systems. He also served as a technical committee member of several international conferences and an associate editor for several referred journals.

Dr. Kassem A. Saleh is currently a professor in Information Sciences at Kuwait University. He received his BS, MS, and PhD in computer science from the University of Ottawa in Canada. Dr. Saleh worked as a computer systems specialist at Mediatel, Bell Canada, from 1985 to 1991, and was on the faculty of Concordia University during 1991-1992, Kuwait University from 1992 to 2000, and American University of Sharjah from 2000 to 2007. Dr. Saleh is also a Certified Information Systems Security Professional (CISSP). The Journal of Systems and Software has ranked Dr. Saleh among the top scholars in the field of systems and software engineering in seven of its annual assessments published from 1996 to 2003. His research interests include software engineering, requirements engineering, and information security. Dr. Saleh has published more than 60 refereed journal papers and has presented numerous tutorials and lectures at international conferences and universities worldwide. Dr. Saleh recently published his first textbook on Software Engineering. Dr. Saleh is currently editor in-chief of the Journal of Software.