PROTOCOL FEATURE INTERACTIONS

Thomas F. LaPorta, David Lee, Yow-Jian Lin
and Mihalis Yannakakis
Bell Laboratories, Lucent Technologies

Abstract: Feature interaction is one of the most intriguing problems in research and development of telecommunications systems. Features may interact and result in undesirable system behavior. Today the analysis of interactions is conducted in an ad hoc manner by subject matter experts. This leads to time-consuming feature design and testing without any interaction-free guarantee.

We present a formal approach for the detection of feature interactions during feature design, specification, implementation, and integration. We use automata to model switch behaviors and services rendered to users. Given a protocol system specification, we propose detection algorithms and prove their correctness and optimality; they require minimal checking to conclude whether a given system is interaction-free. To aid with the introduction of new features to an existing protocol system, we describe efficient algorithms for the incremental validation of interaction-free conditions. Moreover, given a set of design and integration requirements, we obtain necessary and sufficient conditions to prove that an interaction-free system can be built based on the given requirements. We illustrate various concepts and techniques using telephony examples.

1 INTRODUCTION

Feature interaction is a major roadblock to the development and evolution of telecommunications systems. Typical telecommunication switches in a system support hundreds of features [1], combinations of which are marketed as services. Though some feature interactions are desirable by design, others could arise unexpectedly, resulting in user annoyance or adverse system operation. Given a complex switching system implementation, it is formidable to check for feature interactions. Hence, much effort is devoted during feature design and integration phases to ensure that unintented feature interactions do not exist. Currently subject matter experts conduct such analysis in an ad hoc manner. The task has been difficult due to large number of features on a system and complicated interaction scenarios among them. This leads to time-consuming feature design and testing, yet lack of interaction-free guarantee.

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35394-4_29

S. Budkowski et al. (eds.), *Formal Description Techniques and Protocol Specification, Testing and Verification*

The analysis of feature interactions is a matter of determining if users' expectation of feature behaviors is consistent with that exhibited by a system. Since various combinations of features can be simultaneously in use, an interaction-free system means that the behavior of a feature or feature set must be independent of the presence or absence of any other features. Two questions follow in investigating feature interactions: (i) Given a design specification of a system that offers many features, is the system design interaction-free? (ii) Given a set of feature requirements based on users' expectations, is it possible to design an interaction-free system that meets those requirements?

To validate in an effective manner that a system is interaction-free, a major challenge is to minimize the number of feature sets to be analyzed. Similarly, in adding a new feature sets to an existing interaction-free system, the challenge is to avoid analyzing those subsets of features of which the interaction-free status has not changed.

Approaches to solving the feature-interaction problem has proliferated over the past several years [2, 3, 4]. Many of these efforts apply formal techniques, which range from languages for describing composition and alternation of features [15], to state transitions or process algebra based modeling in conjunction with detection techniques such as reachability analysis or model checking (e.g., see [5, 6, 12, 14]). Nevertheless, they mainly focus on detection mechanisms; the complexity of the above mentioned challenges remains an open issue.

This paper addresses the challenges in the context of the two raised questions. We present a formal approach to specify features from both user and network perspectives, and define conditions that guarantee a system to be free from undesirable feature interactions. Regarding the first question, we study efficient algorithms for feature interaction detection. We derive a necessary and sufficient set of combinations that need to be tested, and show that there is no need to check every possible combination of features. Furthermore, we prove that our algorithm is optimal in a sense that the number of tests conducted is minimal. Depending on how feature sets are grouped, this can significantly reduce the number of tests performed. Concerning the second question, we derive necessary and sufficient conditions for the design of an interaction-free system that satisfies feature requirements. We present an efficient algorithm and show that it is optimal. In an evolving telecommunications system, the system is augmented with new features, and the new features may interact with each other or with existing features. We thus study conditions for interactions due to introducing new features and discuss algorithms for their testing.

The remainder of the paper is organized as follows. In the next section we introduce the framework and present the basic definitions of protocol systems, features and their interaction. In Section 3 we address the issues in validating an interaction-free system. In Section 4 we illustrate our methods on a telephony example. In Section 5 we discuss the design and integration of interaction-free systems. Section 6 deals with evolving systems, followed by our concluding remarks in Section 7. Due to the space limit, we omit all the proofs and intermediate results and refer the readers to [10].

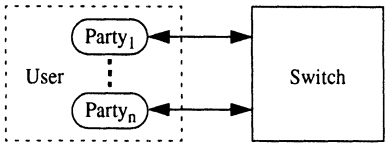

Figure 1 A reference model of a protocol system

2 PROTOCOL SYSTEMS

A communication system provides a set $\mathbf{F}$ of *services*, or *features* to its users. The feature set $\mathbf{F}$ includes the *basic feature* F_0 and a set of *value-added features* F_i, $i = 1, 2, \cdots, N$. Communication between the system and the users takes place via two sets of matching *service primitives*: $S_\mathbf{F} = \{f_1, f_2, \cdots\}$ and $\bar{S}_\mathbf{F} = \{\bar{f}_1, \bar{f}_2, \cdots\}$. The sets $S_\mathbf{F}$ and $\bar{S}_\mathbf{F}$ represent service primitives from the user's point of view and the system's (e.g., switch's or network's) point of view, respectively. Each feature F_i has two associated subsets of matching service primitives $S_{F_i} \subseteq S_\mathbf{F}$ and $\bar{S}_{F_i} \subseteq \bar{S}_\mathbf{F}$ with the property that a primitive f is in S_{F_i} if and only if its matching primitive $\bar{f}$ is in $\bar{S}_{F_i}$. We assume that $S_\mathbf{F} = \cup_i S_{F_i}$ and $\bar{S}_\mathbf{F} = \cup_i \bar{S}_{F_i}$. The switch may have additional *non-service primitives* which are used to communicate with other network entities.

We assume that for each element f in $S_\mathbf{F}$ there is one and only one matching element in $\bar{S}_\mathbf{F}$ and vice versa. The matching relation between the service primitives is for the corresponding *send* and *receive* events, denoted by $!x$ and $?x$ respectively, for a message x between a user and the switch in synchronous communication [7]. The matching relation can be extended from the service primitives to strings and languages over them. Two strings $f_{p_1} f_{p_2} \cdots f_{p_k} \in S_\mathbf{F}^*$ and $\bar{f}_{q_1} \bar{f}_{q_2} \cdots \bar{f}_{q_k} \in \bar{S}_\mathbf{F}^*$ *match* if f_{p_i} and $\bar{f}_{q_i}$ match for $i = 1, \cdots, k$. For a string x over $S_\mathbf{F}$ or $\bar{S}_\mathbf{F}$, we denote by $match(x)$ the string that matches x. For a language L over the alphabet $S_\mathbf{F}$ or $\bar{S}_\mathbf{F}$, let $match(L) = \{match(x) : x \in L\}$. Note that since the matching relation among the service primitives is one-to-one and onto, if $x' = match(x)$ then also $x = match(x')$, and, consequently, if $L' = match(L)$, then also $L = match(L')$.

Based on the network design and user demands, each user is able to use a subset of features in $\mathbf{F}$, including at least the basic feature F_0. Let $\mathcal{F} = \{F : F_0 \in F \subseteq \mathbf{F}\}$ be a class of all the possible feature sets offered by the system (this need not be the complete powerset) for a group of communicating parties; all sets include the basic feature F_0. Figure 1 depicts the relationship between network and users. In our general framework, the term "user" means a group of communicating parties (shown as a dash box in Figure 1); the features set of a user therefore refers to the combinations of features used by *all* communicating parties. We also discuss a special setting in which a feature set is the set of features available to each communicating party.

We model the behavior of each feature set F by a *user automaton* A_F with input symbols S_F. We model the switch behavior by a *switch automaton* $\bar{A}$ whose input symbols consists of the service primitives of all the features $\bar{S}_\mathbf{F}$

and possibly other non-service primitives. The automata have an initial "idle" state. All the states of the user and switch automata are considered accepting. We use $L(A)$ to denote the language accepted by an automaton A; in the case of the switch automaton, the transitions on non-service primitives are treated as τ moves, i.e., $L(\bar{A})$ is a language over the alphabet $\bar{S}_{\mathbf{F}}$.

Definition 1 A *protocol system* P on feature sets class $\mathcal{F}$ contains a switch automaton $\bar{A}$ and a set of user automata $\{A_F : F \in \mathcal{F}\}$ (one for each feature set F in $\mathcal{F}$) which communicate with $\bar{A}$ by matching service primitives.[1] □

In order to identify each transition in the switch automaton on a per feature basis, we also tag each transition with the corresponding feature(s) of which the execution traverses the transition. Multiple features may share a transition, i.e., the execution of these features may traverse the same transition labeled with either a shared primitive or a non-service primitive. In this case the transition is tagged by all the features that share the transition. A transition in the switch automaton is F_i-related if it is tagged with F_i. A transition in the switch automaton is F-related if it is F_i-related for some $F_i \in F$. Note that the feature tags are not part of the alphabet for the switch automaton.

Consider the switch behavior with respect to a feature set F in $\mathcal{F}$. If no other features are present, then the switch will traverse only F-related transitions; that is, the other transitions are as if they are missing. This gives rise to the following definition of a restriction operator.

Definition 2 Let $\bar{A}$ be an automaton whose transitions are tagged by the features. The *restriction* of $\bar{A}$ on a set of features F, denoted $\rho_F(\bar{A})$, is the automaton obtained from $\bar{A}$ by deleting all transitions that are not F-related. The remaining transitions retain in their tags only the features from F. □

To the parties that jointly use feature set F, the user automaton A_F should be consistent with the view $\rho_F(\bar{A})$ supported by the switch. Formally,

Definition 3 A protocol system $P = (\bar{A}, \{A_F\}_{\mathcal{F}})$ on feature set class $\mathcal{F}$ is *consistent* if $L(A_F) = match(L(\rho_F(\bar{A})))$ for all $F \in \mathcal{F}$. □

Note that both the switch and user automata can be nondeterministic. There are various notions of equivalence for nondeterministic automata, including language (trace) equivalence, observational equivalence etc. [13]. In this paper we will use language equivalence, mainly for reasons of simplicity; a similar development can be carried out using other equivalences.

Since the matching of service primitives of the user and switch automata is unique, given the switch automaton of a consistent protocol system, the user automata are well defined and unique in terms of language acceptance.

Example 1 Figure 2 shows a switch automaton and a user automaton as part of a protocol system from telephony. We label transitions in a format [!,?]X_Y,

[1] For clarity we omit specification P of the underlining protocol system if there are no ambiguities.

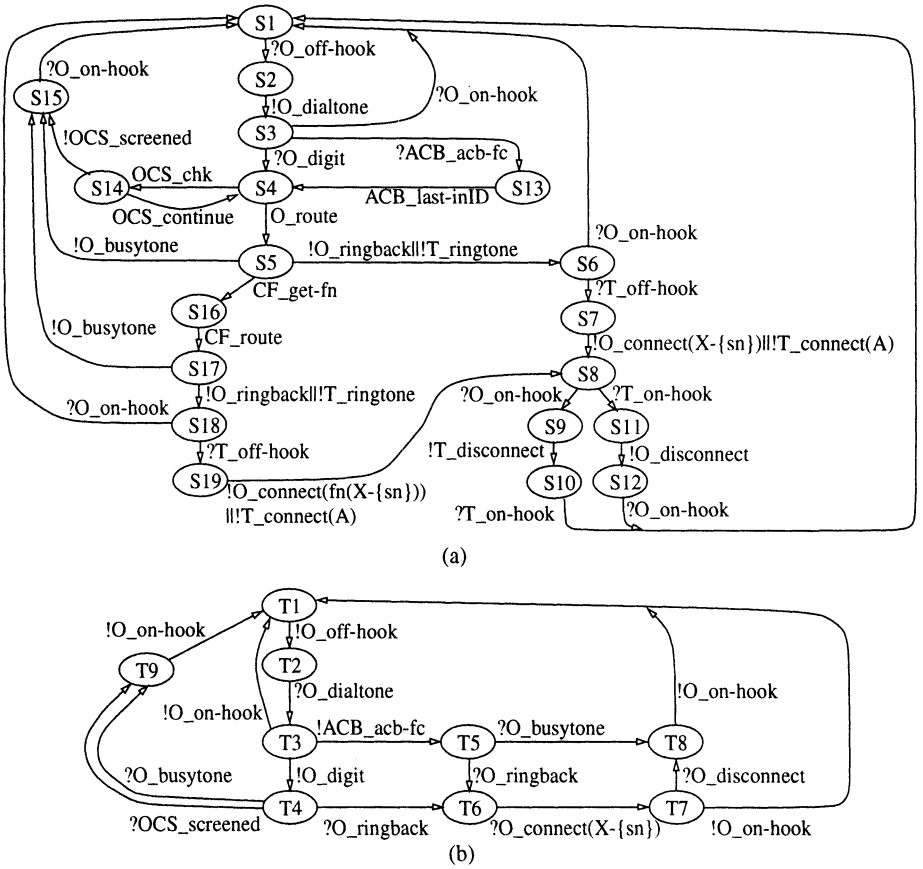

Figure 2 An example of (a) a switch automaton and (b) a user automaton

where X corresponds to a feature tag, and Y is a primitive. The labels preceded with either "!" or "?" are service primitives, and the ones without are non-service primitives. The switch is for calls involving two parties, the caller and the callee. Correspondingly, the basic feature set has two components: the originating POTS (Plain Old Telephone System), denoted O in the figure, and the terminating POTS, denoted T. The switch provides three valued-added features: Originating Call Screening (OCS), Automatic CallBack (ACB), and Call Forwarding (CF). The user automaton represents expected caller behaviors. □

Assume a set of operative features that includes a feature set F and possibly an additional set E. Consider the switch behavior as perceived by an observer who tracks only the communications involving F. We can classify transitions of the switch automaton into three categories: F-related transitions; E-related but not F-related transitions; and the rest (neither F- nor E-related) transitions. The switch now traverses transitions in the first and second categories but not the ones in the third category, as if features in $\mathbf{F} - (F \cup E)$ are not available. Consequently, we can delete the transitions in the third category from

the switch automaton without affecting the apparent behavior of the switch. Furthermore, the associated semantics of transitions in the second category are irrelevant to F; hence from F's point of view we can change these transitions to τ moves. This gives rise to the following projection operator.

Definition 4 Let $\bar{A}$ be an automaton whose transitions are tagged by the features. The *projection* of $\bar{A}$ on a set of features F, denoted $\pi_F(\bar{A})$, is the automaton obtained from $\bar{A}$ by changing to τ moves all transitions that are not F-related (the τ transitions retain their tags).

For a pair of feature sets F and E, the *projection* of the automaton $\bar{A}$ on F w.r.t. (or, in the context of) E, denoted by $\pi_{F|E}(\bar{A})$, is the automaton $\pi_F(\rho_{F\cup E}(\bar{A}))$, i.e., it is the automaton obtained from $\bar{A}$ by changing the transitions as follows: (i) All F-related transitions remain unchanged; (ii) All transitions that are E- but not F-related become τ moves; and (iii) All the other transitions (those are not $(F \cup E)$-related) are deleted. □

The restriction operator ρ_F corresponds to the situation where only feature set F is available. The projection operator π_F corresponds to the situation where one is observing the system behavior only from F's viewpoint, though other features may be available and active. The operator $\pi_{F|E}$ combines the two cases.

The restriction operator ρ_F is similar to that of *protocol pruning* [11] where the transitions labeled with service primitives not in the feature set F are deleted. Furthermore, only the strongly connected component with the initial state is maintained for the protocol operations. The projection operator is different from *protocol projection* [9], which is intended for protocols with several distinguishable services. The goal there is to construct image protocols for each service that preserve safety and progress properties. In [11] the projection is done by aggregating states into blocks (and message values into blocks).

We also define our restriction and projection operators in languages.

Definition 5 Given a string $x \in S_{\mathbf{F}}^*$ ($\bar{x} \in \bar{S}_{\mathbf{F}}^*$) and a subset of features $F \in \mathcal{F}$, the *projection* of x ($\bar{x}$) onto F, denoted by $\pi_F(x)$ ($\pi_F(\bar{x})$), is obtained by deleting all the symbols in x ($\bar{x}$) that are not in S_F ($\bar{S}_F$). The definition can be extended to languages in a straightforward way: $\pi_F(L) = \{\pi_F(x) : x \in L\}$. The *restriction* of L on F, denoted $\rho_F(L)$, is the set of all strings of L that contain only symbols in $\bar{S}_F$ (or in S_F for a language on the user side). The projection of L on F w.r.t. E is $\pi_{F|E}(L) = \pi_F(\rho_{F\cup E}(L))$. □

Example 2 Figure 3 depicts the projection $\pi_{\{O\}|\{T\}}(\bar{A})$ of the switch automaton in Figure 2(a). It is the basic POTS as seen from the caller. For simplicity we have removed τ transitions and redundant states. □

We now define interactions between feature sets. The definitions of independent feature sets and interaction-free protocol systems follow.

Definition 6 A feature set $F \in \mathcal{F}$ is *independent* of (or *free of interactions* from) another feature set E, denoted as $ind(F, E)$, if $L(\pi_{F|E}(\bar{A})) = L(\rho_F(\bar{A}))$. Otherwise, we say that F has interactions from E. □

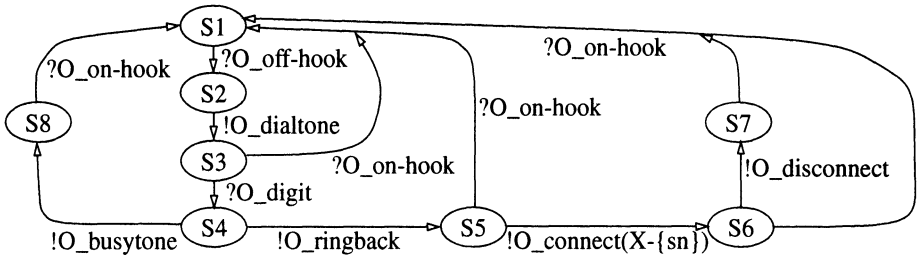

Figure 3 The projection $\pi_{\{O\}|\{T\}}(\bar{A})$ of the switch automaton in Figure 2(a)

Note that this relation is not symmetric; $ind(F, E)$ does not imply that $ind(E, F)$. Moreover, interactions between E and F need not concern us unless $(F \cup E)$ belongs to some feature set offered by the protocol system.

Definition 7 A feature set $F \in \mathcal{F}$ is *independent* (or *interaction-free*) in $\mathcal{F}$ if $ind(F, E)$ holds $\forall E$ such that $(F \cup E) \subseteq H$ for some $H \in \mathcal{F}$. A protocol system P with feature set class $\mathcal{F}$ is *interaction-free* if every set $F \in \mathcal{F}$ is interaction-free in $\mathcal{F}$. □

For a protocol system P, the design and engineering of the system is on the switch automaton $\bar{A}$. On the other hand, the services provided from the user's point of view are specified by the user automata. Hence, analyzing feature interactions means validating that P is consistent and interaction-free. In the next section we investigate the detection of feature interactions.

3 FEATURE INTERACTION DETECTION

Consider a protocol system $P = (\bar{A}, \{A_F\}_{\mathcal{F}})$ on a class $\mathcal{F}$ of feature sets. Checking for the consistency of P is straightforward from the definition: we need to check for each feature set $F \in \mathcal{F}$ that $L(A_F) = match(L(\rho_F(\bar{A})))$. The interesting part is checking for interactions.

The straightforward way to check for interaction-free is to check $ind(F, E)$ for every pair of feature sets, F, E with $(F \cup E) \in \mathcal{F}$. We seek ways to avoid checking all possible pairs. As for the basic operation of checking interactions of some pairs of feature sets, from Definition 6, the computation is reduced to checking equivalence of two automata, which has been well studied in automata theory [8]. In general, it is an expensive operation and we want to minimize the number of such operations in feature interaction detection process. For convenience, we assume that each such operation has a unit cost, and we want to design feature interaction detection algorithms that minimize the total cost.

Intuitively, if there are no interactions from a feature set then there are no interactions from its subsets as well.

Lemma 1 Suppose that F, E, E' are sets of features with $E' \subseteq E$. If $ind(F, E)$ then also $ind(F, E')$. The independence relation ind is transitive on compara-

ble sets: for feature sets $F \subset K \subset L$, if $ind(F,K)$ and $ind(K,L)$ then also $ind(F,L)$. □

We proceed now to study how to check whether a given protocol is interaction-free. We first consider a protocol with two parties, the originating and the terminating party. The same development can be extended along the same lines to the case of multiple parties. Assume we are given a switch automaton $\bar{A}$, and that the features are partitioned into originating and terminating features, there is a class $\mathcal{F}_\mathcal{O}$ of originating feature sets and a class $\mathcal{F}_\mathcal{T}$ of terminating feature sets. A feature set in $\mathcal{F}_\mathcal{O}$ can pair with a feature set in $\mathcal{F}_\mathcal{T}$, and vice versa. There is a basic originating feature set F_0 included in all the sets of $\mathcal{F}_\mathcal{O}$, and a basic terminating feature set E_0 included in all the terminating feature sets. We wish to determine whether there are any feature interaction between the two parties, i.e. whether the observed behavior by one party for some feature set depends on the feature set of the other party.

Formally, a feature set $F \in \mathcal{F}_\mathcal{O}$ is independent of a feature set $E \in \mathcal{F}_\mathcal{T}$, denoted $ind(F,E)$, if $\pi_{F|E}(\bar{A}) = \pi_{F|E_0}(\bar{A})$, i.e., the originating side cannot tell the difference between the terminating side using E or its basic feature set E_0 (note that it has to use at least E_0). Similarly, E is independent of F if $\pi_{E|F}(\bar{A}) = \pi_{E|F_0}(\bar{A})$. An originating or terminating feature set is called *externally independent* if it is independent of every feature set of the other party. (Note that it may still have interactions from features of the same side). The protocol system is externally independent if all the feature sets are.

The definition of external independence asks to check two language equalities for each pair of feature sets F, E with $F \in \mathcal{F}_\mathcal{O}$ and $E \in \mathcal{F}_\mathcal{T}$. We will use the properties from the last subsection to reduce this number.

Definition 8 The *maximal subclass* of a class of feature sets $\mathcal{F}$, denoted by $\mathcal{F}^*$, is a subset of $\mathcal{F}$ such that (i) No feature set in $\mathcal{F}^*$ is a proper subset of any other one in $\mathcal{F}$; and (ii) For each feature set $G \in \mathcal{F}$ there is a feature set $F \in \mathcal{F}^*$ such that $G \subseteq F$. □

For a given class of feature sets $\mathcal{F}$, there is a unique maximal subclass $\mathcal{F}^*$; it consists of the set-theoretically maximal sets of $\mathcal{F}$. It can be easily obtained by eliminating all the elements in $\mathcal{F}$ that is a proper subset of another element. Therefore, for the two disjoint feature set classes for the two parties, $\mathcal{F}_\mathcal{O}$ and $\mathcal{F}_\mathcal{T}$, we have the two maximal subclasses, $\mathcal{F}_\mathcal{O}{}^*$ and $\mathcal{F}_\mathcal{T}{}^*$. The feature interaction procedure is summarized in Algorithm 1. Line 2 and 5 call a subroutine $ind(\cdot,\cdot)$, which checks for independence of two feature sets, as in Definition 6. It constructs two automata from the two given feature sets and checks whether they are equivalent or accept the same language with a unit cost.

The following two theorems show the correctness of Algorithm 1, i.e., checking for interaction-free with respect to the maximal feature sets is sufficient.

Theorem 1 A feature set $F \in \mathcal{F}_\mathcal{O}$ is externally independent if and only if $\forall E \in \mathcal{F}_\mathcal{T}{}^*, ind(F,E)$, i.e., F is independent of every maximal feature set E of $\mathcal{F}_\mathcal{T}$. Similarly, A feature set $H \in \mathcal{F}_\mathcal{T}$ is externally independent if and only if $\forall G \in \mathcal{F}_\mathcal{O}{}^*, ind(H,G)$. □

Algorithm 1 (Feature Interaction Detection - Two Parties)

Input: Feature sets $\mathcal{F}_{\mathcal{O}}, \mathcal{F}_{\mathcal{T}}, \mathcal{F}_{\mathcal{O}}{}^*, \mathcal{F}_{\mathcal{T}}{}^*$.

```
1   for (each pair (F ∈ F_O, E ∈ F_T*)) do
2       if (ind(F, E) = FALSE)
3           return "feature interaction detected";
4   for (each pair (H ∈ F_T, G ∈ F_O*)) do
5       if (ind(H, G) = FALSE)
6           return "feature interaction detected";
7   return "feature interaction free";
```

Thus, for each $F \in \mathcal{F}_{\mathcal{O}}$ ($H \in \mathcal{F}_{\mathcal{T}}$) the cardinality of $\mathcal{F}_{\mathcal{T}}{}^*$ ($\mathcal{F}_{\mathcal{O}}{}^*$) determines the number of feature sets that we must validate against in order to claim that F (H) is independent in $\mathcal{F}$. In the worst case, it is the total number of value-added features in $\mathcal{F}_{\mathcal{O}}$ ($\mathcal{F}_{\mathcal{T}}$); in other words, every feature set in $\mathcal{F}_{\mathcal{O}}$ ($\mathcal{F}_{\mathcal{T}}$) consists of exactly one value-added feature and the basic feature. In the best case, the cardinality of $\mathcal{F}_{\mathcal{O}}$ ($\mathcal{F}_{\mathcal{T}}$) is one, that is, there is a feature set in $\mathcal{F}_{\mathcal{O}}$ ($\mathcal{F}_{\mathcal{T}}$) that contains every originating (terminating) value-added features.

From Algorithm 1, the pairs of feature sets to be checked for independence is $|\mathcal{F}_{\mathcal{O}}| \times |\mathcal{F}_{\mathcal{T}}{}^*| + |\mathcal{F}_{\mathcal{T}}| \times |\mathcal{F}_{\mathcal{O}}{}^*|$ whereas a brute-force checking requires $2|\mathcal{F}_{\mathcal{O}}| \times |\mathcal{F}_{\mathcal{T}}|$:

Corollary 1 Given a class of feature set $\mathcal{F}$. Let n and m be the cardinality of $\mathcal{F}_{\mathcal{O}}{}^*$ and $\mathcal{F}_{\mathcal{T}}{}^*$, and N and M be the cardinality of $\mathcal{F}_{\mathcal{O}}$ and $\mathcal{F}_{\mathcal{T}}$, respectively. To check that every feature set is externally independent, Algorithm 1 reduces the number of tests from $2NM$ to $nM + Nm$. □

Can we further reduce the number of tests for external independence? The following theorem shows that Algorithm 1 is optimal in a sense that it conducts all the necessary checking and, therefore, there is no guarantee of external independence if less tests are performed.

Theorem 2 Let S be a set of pairs (F, E) with $F \in \mathcal{F}_{\mathcal{O}}$, $E \in \mathcal{F}_{\mathcal{T}}$. If S does not contain $(\mathcal{F}_{\mathcal{O}} \times \mathcal{F}_{\mathcal{T}}{}^*) \cup (\mathcal{F}_{\mathcal{O}}{}^* \times \mathcal{F}_{\mathcal{T}})$, then there is a protocol system P such that $\forall (F, E) \in S$, $ind(F, E)$ holds, but P is not externally independent. □

An important special case is that there is a feature set that includes all the features: $E = \mathbf{F}_T$ and $G = \mathbf{F}_O$. In this case, $m = n = 1$, and we only need to perform a linear number of tests:

Corollary 2 Suppose that there exists a feature set $E \in \mathcal{F}_{\mathcal{T}}$ and $G \in \mathcal{F}_{\mathcal{O}}$ that include all the features, i.e., $E = \mathbf{F}_T$ and $G = \mathbf{F}_O$. Then all the feature sets are externally independent if $ind(F, E)$ for all $F \in \mathcal{F}_O$ and $ind(H, G)$ for all $H \in \mathcal{F}_T$. The number of tests is thus reduced to $M + N$ where and N and M be the cardinality of $\mathcal{F}_{\mathcal{O}}$ and $\mathcal{F}_{\mathcal{T}}$, respectively. □

We come back now to the general framework. We have a communication protocol P, given by the switch automaton $\bar{A}$, with a class $\mathcal{F}$ of feature sets. We

Algorithm 2 (Feature Interaction Detection - General Case)

Input: Feature sets $\mathcal{F}$

```
1   for (each F ∈ 𝓕) do
2       construct interaction graph D_F;
3       for (each weakly connected component of D_F) do
4           find a sink node G;
5           if (ind(F,G) = FALSE)
6               return "feature interaction detected";
7   return "feature interaction free";
```

wish to check if P is interaction-free, without checking all the pairs of feature sets of $\mathcal{F}$.

First note that Lemma 1 implies that we need only perform independence tests $ind(F, E)$, where E is a maximal set of $\mathcal{F}$. For, if E is a proper subset of another set G of $\mathcal{F}$, then we can test instead $ind(F, G)$; if it holds, then also $ind(F, E)$. Recall also that independence of a feature set F requires that $ind(F, E)$ for all sets E such that $F \cup E$ is contained in some feature set G of $\mathcal{F}$. Thus, we only need to test pairs (F, G) of feature sets of $\mathcal{F}$ such that $F \subset G$ and G is a maximal set of $\mathcal{F}$. Do we need to test all these pairs? Not necessarily. We investigate below the pairs that need to be checked. Although the choice of these pairs is not necessarily unique in this case as we shall see, the number of pairs is uniquely determined.

Consider the partial order induced by set containment on the class $\mathcal{F}$ of feature sets. That is, we form a directed graph D with the feature sets as nodes and with an arc $F \to G$ if $F \subset G$. By 'connectivity' below we mean weak connectivity, i.e., the directions of the arcs are ignored and the graph is treated as undirected.

For a node (feature set) F, let D_F be the *interaction graph*, a subgraph induced by all the proper descendants of F, i.e. all nodes corresponding to sets that properly contain F. Let c_F be the number of connected components of D_F. We shall show that $\sum_{F \in \mathcal{F}} c_F$ tests are necessary and sufficient to verify that the protocol is interaction-free.

For each feature set $F \in \mathcal{F}$ and each connected component of D_F, we select a sink of the component G (i.e., G is in the component and is a maximal set of $\mathcal{F}$) to form a pair (F, G). Let S be a set of all such pairs. Note that a component may have several sinks so there is a choice for which one we pick for G to test against F. That is, S is not uniquely determined, although its cardinality is determined, namely $\sum_F c_F$. The procedure is summarized in Algorithm 2 and following theorem proves its correctness:

Theorem 3 If $ind(F, E)$ for all pairs in S, then P is interaction-free. □

We can show again that Algorithm 2 is optimal if we access the switch automaton through independence tests; if we omit any of the tests in Algorithm

2 then there is no guarantee of interaction-fee. We can assume without loss of generality that we test pairs (F, E) with $F \subset E$ feature sets of $\mathcal{F}$.

Theorem 4 Let R be a set of pairs of sets of features. Suppose that there is a feature set $F \in \mathcal{F}$ and a connected component C of D_F such that R does not contain any pair (F, E) with E in the connected component C. Then there is a protocol system P that passes all the independence tests in R but the protocol is not interaction-free. □

An important special case is that there is a feature set that includes all the features: $E = \mathbf{F}$, and it is the sink node of all connected components of the interaction graphs of all feature sets. We only need to check each feature set F against E: K

Corollary 3 Suppose that there exists a feature set $E \in \mathcal{F}$ that include all the features, i.e., $E = \mathbf{F}$. Then all the feature sets are independent if $ind(F, E)$ for all $F \in \mathcal{F}$. The number of tests is thus reduced to N where N is the cardinality of $\mathcal{F}$. □

4 EXAMPLE

Consider the switch automaton $\bar{A}$ given in Figure 2 and assume that a caller can use any combination of the two originating features, OCS and ACB, whereas a callee has the choice of using the terminating feature CF. The feature set class $\mathcal{F}$ in the setting of Section 3 is $\mathcal{F} = \{\{O,\ OCS,\ ACB\},\ \{O,\ OCS\},\ \{O,\ ACB\},\ \{T,\ CF\}\}$. The following illustrates how a pair of feature sets can fail the independence checking.

Assume that a user a has the feature set $\{O, OCS, ACB\}$ and another user b has the feature set $\{T, CF\}$. Assume further that a has c on the screening list for OCS, and b has c as the forwarding number for CF. Figure 4(a) and (b) depict the projections $\pi_{\{O,OCS,ACB\}|\{T,CF\}}(\bar{A})$ and $\pi_{\{O,OCS,ACB\}|\{T\}}(\bar{A})$ respectively of the switch automaton in Figure 2(a). The two automata accept the same set of language except that the automaton in (a) also accepts a string ...O_connect(c)... through states S9 and S10 but the automaton in (b) does not. The string corresponds to the case that user a attempts to call b, who in turn forwards the call to c; hence a is connected to c even though c is on a's screening list.

Based on our results in Section 3, Table 1(a) summarizes the independence checks we need to perform before claiming that the feature set is externally interaction-free. In addition, we need to check that OCS and ACB are independent of each other. That is, we need to check for $L(\pi_{\{O,OCS\}|\{ACB,T\}}(\bar{A})) = L(\pi_{\{O,OCS\}|\{T\}}(\bar{A}))$ as well as $L(\pi_{\{O,ACB\}|\{OCS,T\}}(\bar{A})) = L(\pi_{\{O,ACB\}|\{T\}}(\bar{A}))$. According to the general framework in Section 3.3, on the other hand, the context for each independence check will be different. Note that in this setting the feature set class $\mathcal{F}$ =$\{\{OCS,\ O,\ T\},\ \{ACB,\ O,\ T\},\ \{OCS,\ ACB,\ O,\ T\},\ \{O,\ CF,\ T\},\ \{OCS,\ O,\ CF,\ T\},\ \{ACB,\ O,\ CF,\ T\},\ \{OCS,\ ACB,\ O,\ CF,\ T\}\}$. Table 1(b) lists the six checks we derived from the interaction graph depicted

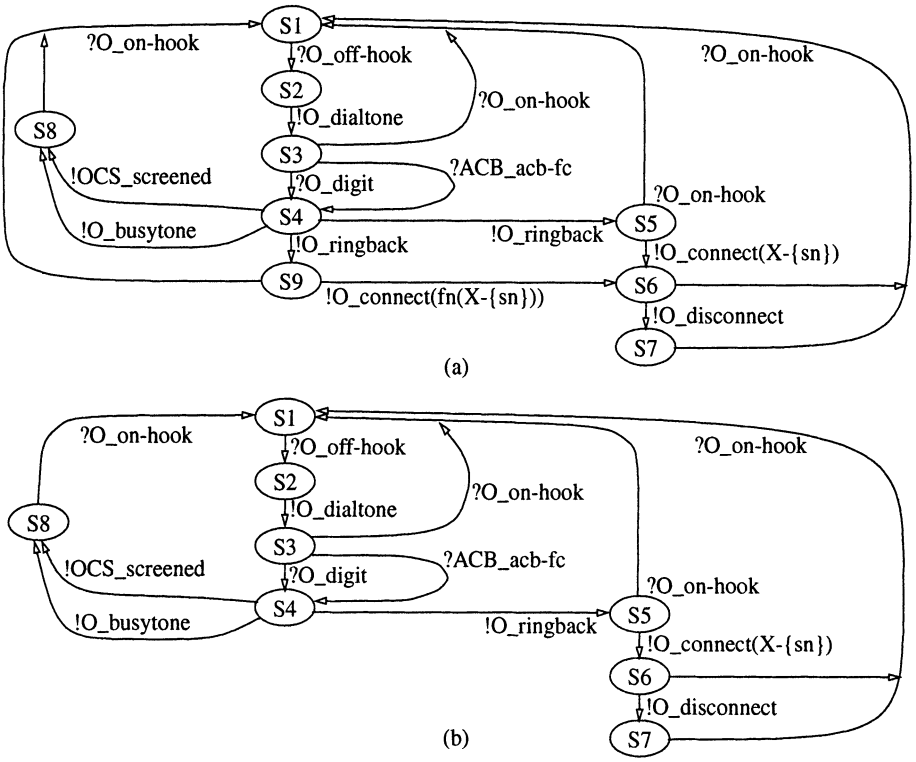

Figure 4 Projections (a) $\pi_{\{O,OCS,ACB\}|\{T,CF\}}(\bar{A})$ and (b) $\pi_{\{O,OCS,ACB\}|\{T\}}(\bar{A})$ of the switch automaton in Figure 2(a)

(a)

1	ind({O,OCS,ACB}, {T,CF})
2	ind({O,OCS}, {T,CF})
3	ind({O,ACB}, {T,CF})
4	ind({T,CF}, {O,OCS,ACB})

(b)

1	ind({OCS,ACB,O,T}, {OCS,ACB,O,CF,T})
2	ind({OCS,O,T}, {OCS,ACB,O,CF,T})
3	ind({ACB,O,T}, {OCS,ACB,O,CF,T})
4	ind({O,CF,T}, {OCS,ACB,O,CF,T})
5	ind({OCS,O,CF,T}, {OCS,ACB,O,CF,T})
6	ind({ACB,O,CF,T}, {OCS,ACB,O,CF,T})

Table 1 A summary of independence checks

in Figure 5. There is a one-to-one correspondence in the semantics of these two sets of checks.

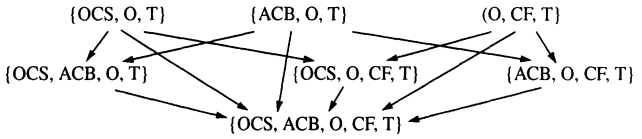

Figure 5 The interaction graph for our example

5 DESIGN AND INTEGRATION OF INTERACTION-FREE PROTOCOL SYSTEMS

In the previous development we have assumed that given a switch automaton and the user automata for the various subsets of features what the users will *actually see* in their interactions with the switch: this is basically what the consistency condition says - the language of a user automaton can be derived from that of the switch automaton. On the other hand, when new features are created, specifications for the features are first created before the features get incorporated in the switch. It is desirable to analyze feature interactions at this stage of design, and indeed typically during the generation of requirements for new features, system engineers analyze and list interactions with other features. In this point of view, we consider user automata (or languages) for various subsets of features to be specifications of what the users are *supposed to see* for different feature sets - i.e. what is desired. We wish to analyze the interactions of the features on the basis of these specifications.

A similar situation arises if we have different systems that need to be integrated. In this case the automata for different features represent the user views for different protocols that were constructed separately and we need to determine whether it is possible to combine them in a consistent manner without interactions.

We first describe a specification system that is typically constructed from design requirements. We want to check whether it is possible to build a protocol system, an implementation from the specification, that meets the design requirements and is interaction-free.

Definition 9 A *specification system* C for a set of features $\mathbf{F} = \{F_i : i = 0, \cdots, r\}$ is a finite set of user automata $\{A_F : F_0 \in F \subseteq \mathbf{F}\}$, where each automaton A_F has input alphabet S_F. The family $\mathcal{F}_C$ of feature sets for which an automaton is specified may or may not be the complete power set of $\mathbf{F} = \{F_i : i = 0, \cdots, r\}$ of all subsets that include the basic feature F_0. [2] □

Definition 10 Let $C = \{A_F : F \in \mathcal{F}_C\}$ be a specification system over a set of features $\mathbf{F} = \{F_i : i = 0, \cdots, r\}$. The system C is *interaction-free* if there is a consistent, interaction-free protocol $P = (\bar{A}, \{A_F : F \in \mathcal{F}_C\})$ whose user automata are as specified by the system C; we say that the protocol P *implements* $C = \{A_F : F \in \mathcal{F}_C\}$. □

[2] This corresponds to the case that features can be obtained only in certain combinations.

Although we did our analysis before based on the switch automaton, we can carry out a similar analysis equally well based on the user automata. That is, we can extend our methodology and algorithms to apply in the setting as well.

Theorem 5 A specification system $\mathcal{C} = \{A_F : F \in \mathcal{F}_\mathcal{C}\}$ is interaction-free if and only if $L(A_F) = \pi_F(L(A_E)) \subseteq L(A_E)$ for all feature sets $F \subset E$ of $\mathcal{F}_\mathcal{C}$. □

We can show along the same lines as Theorem 3 that it is sufficient to verify the equality $L(A_F) = \pi_F(L(A_E))$ only for the pairs (F, E) of the set S of Theorem 3; the equality then is implied for all other pairs $F \subset E$. Furthermore, clearly the containment $L(A_F) \subseteq L(A_E)$ needs to be checked only for the pairs (F, E) in the Hasse diagram (transitive reduction) of the partial order that is induced by the feature sets in $\mathcal{F}$, i.e., for the pairs $F \subset E$ in $\mathcal{F}$ such that there is no $G \in \mathcal{F}$ with $F \subset G \subset E$.

6 INCREMENTAL VALIDATION

Suppose that we have a consistent, interaction-free protocol system P with switch automaton $\bar{A}$ and user automata A_F, $F \in \mathcal{F}$. There are two possible ways of extending P: adding new features, i.e., modifying the switch automaton $\bar{A}$; or adding new feature sets to $\mathcal{F}$ only. In the following we discuss a general scenario of adding both a new feature and new feature sets.

Consider the introduction of a new feature F_{N+1} to protocol system P. It results in a new protocol system Q, where Q consists of: (i) A new switch automaton $\bar{B}$; (ii) A new class of feature sets $\mathcal{G} = \mathcal{F} \cup \{G_i : F_{N+1} \in G_i\}$; [3] (iii) A user automaton A_{G_i} for each new feature set G_i; and (iv) The old user automaton A_F for every old feature set $F \in \mathcal{F}$. We would like the new protocol to provide the same service for each one of the old features, hence the condition that the user automata for the old features remain the same in Q. To verify that Q is consistent, we need to check again for every old and new feature F that $L(A_F) = match(L(\rho_F(\bar{B}))$. Note that we have to verify again consistency for the old features since the switch automaton has changed to an arbitrary new automaton $\bar{B}$.

For feature interaction detection, we do not need to check from scratch; we can use the fact that the old protocol was interaction-free to avoid checking the old features. The procedure is summarized in Algorithm 3.

Theorem 6 Suppose that a consistent and interaction-free protocol system P of feature set $\mathcal{F}$ is incremented with new feature sets $\mathcal{F} \subset \mathcal{G}$, resulting in a new consistent protocol system Q. Algorithm 3 determines correctly whether the incremented protocol system Q is interaction-free. □

[3] We assume that all feature sets in $\mathcal{F}$ remain in $\mathcal{G}$. Adding the new feature F_{N+1} results in adding a new set of feature sets, each of which contains F_{N+1}.

Algorithm 3 (Feature Interaction Detection - Incremental Validation)

Input: Two feature sets classes $\mathcal{F}$ (old) and $\mathcal{G}$ (incremented); $\mathcal{G} - \mathcal{F}$ is the set of new feature sets.

```
1   for (each F ∈ F) do
2       construct interaction graph D_F;
3       for (each weakly connected component of D_F) do
4           if (it does not contain any old feature set node)
5               find a sink node G;
6               if (ind(F,G) = FALSE)
7                   return "feature interaction detected";
8   for (each new feature set F in G − F) do
9       construct interaction graph D_F;
10          for (each weakly connected component of D_F) do
11              find a sink node G;
12              if (ind(F,G) = FALSE)
13                  return "feature interaction detected";
14  return "feature interaction free";
```

7 CONCLUDING REMARKS

We have proposed a formal framework for a study of feature interaction problem. We investigated conditions and algorithms for testing features for freedom from interactions, with an eye towards minimizing the number of tests that need to be performed.

Certain features behave differently by design when other features are present. For example, the Caller ID feature will not display the ID of any caller who subscribes to the Caller ID Blocking feature. As a result, a feature set containing these features will not be independent based on our current definition. One way to address this issue is to modify our definition of independence: a feature set F is independent in $\mathcal{F}$ if the behavior of F is independent of the presence of any feature that F has no desired interactions with. Formally, let D be the set of features such that F and D have desirable interactions. $ind(F, E)$ if $L(\pi_{F|E}(\bar{A})) = L(\pi_{F|(E\cap D)}(\bar{A}))$.

In the algorithms we invoked a subroutine call $ind(\cdot, \cdot)$ to check for interactions. Also we need consistency checking. Formally, they require manipulations on the switch automaton, which is complex and often not available. Instead of explicitly constructing the switch automaton and performing projection and restriction operations for checking, we can apply conformance testing to the switch automaton, which is a "black-box" as follows. From the user automata with features of interest, we expect the structure and hence the languages accepted by the switch automaton for the interaction and consistency checking. Therefore, we can apply a conformance test to the switch automaton to confirm its structure to the expectation and conclude the consistency and the presence or absence of feature interactions.

This work is an attempt to use formal methods to model and analyze protocol feature interactions. We use automata to model both the switch and user behavior. They model the control portions of protocols well, but they are not powerful enough to succinctly model the data portions where variable values determine the system behavior and parameters are passed among system entities in messages. Extended, communicating, and parameterized finite state machines can be use to model more properly the real protocol systems. On the other hand, timers play an important role in determining protocol behaviors, and their presence further complicates the analysis of feature interactions. A lot of issues remain open in the research of feature interaction and much effort is needed before formal techniques can have a real impact on practical systems.

References

[1] 5ESS Switch Feature Handbook, AT&T, September, 1987.

[2] IEEE Communications Magazine, Feature Topic: Managing Feature Interactions in Telecommunications Systems, Ed. N.D. Griffeth, Y-J. Lin, Vol. 31, No. 8, Aug., 1993.

[3] IEEE Computer, Special Issue: Telecommunications - How many Features Can You Add?, Ed. N. D. Griffeth, Y-J. Lin, Vol. 26, No. 8, Aug., 1993.

[4] Proceedings of International Workshop on Feature Interactions in Telecommunications Networks, 1994-1997.

[5] P. K. Au and J. M. Atlee, Evaluation of a State-based Model of Feature Interactions, in [4], pp. 153-167, 1997.

[6] O. C. Dahl and E. Najm, Specification and Detection of IN Service Interference Using LOTOS, in *FORTE VI*, pp. 53–69, 1993.

[7] C. A. R. Hoare, Communicating Sequential Processes, *Prentice-Hall*, New York 1985.

[8] J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation,* Addison-Wesley, 1979.

[9] S. S. Lam and A. U. Shankar, Protocol verification via projections, *IEEE Trans. on Software Engineering,* vol. SE-10, no. 4, pp. 325-342, 1984.

[10] T. La Porta, D. Lee, Y.-J. Lin, and M. Yannakakis, Protocol Feature Interactions, *Tech. Mem. Bell Labs., no. 113410-980422/03*, 1998.

[11] D. Lee, A. N. Netravali and K. Sabnani, Protocol pruning, *Proceedings of The IEEE,* Vol. 83, No. 10, pp. 1357–1372, 1995.

[12] F. J. Lin and Y.-J. Lin, A Building Block Approach to Detecting and Resolving Feature Interactions, in [4], pp. 86–119, 1994.

[13] R. Milner, *Communication and Concurrency,* Prentice Hall, 1989.

[14] M. Thomas, Modelling and Analysing User Views of Telecommunications Services, in [4], pp. 168–182, 1997.

[15] K. J. Turner, An Architectural Foundation for Relating Features, in [4] pp. 226–241, 1997.