Detecting Feature Interactions: how many components do we need?

Muffy Calder and Alice Miller

Department of Computing Science
University of Glasgow
Glasgow, Scotland.
muffy,alice@dcs.gla.ac.uk

Abstract. Features are a structuring mechanism for *additional* functionality, usually in response to changing requirements. When several features are invoked at the same time, by the same, or different components, the features may not interwork. This is known as *feature interaction*. We employ a property-based approach to feature interaction detection: this involves checking the validity (or not) of a temporal property against a given system model. We use the logic LTL for temporal properties and the model-checker Spin to prove properties.
To gain any real insight into feature interactions, it is important to be able to infer properties for networks of *any* size, regardless of the underlying communication structure. We present an inference mechanism based on abstraction. The key idea is to model-check a system consisting of a constant number (m) of components together with an *abstract* component representing any number of other (possibly featured) components. The approach is applied to two systems with communication which is peer to peer and client server. We outline a proof of correctness in both cases.
The techniques developed here are motivated by feature interaction analysis, but they are also applicable to reasoning about networks of other types of components with suitable notions of data abstraction.

1 Introduction

Features are a structuring mechanism not dissimilar from objects and agents. For example, services are built up from a number of feature components. However, a major philosophical difference is that feature components are usually *additional* to a core body of software, often they are a response to new, or changing requirements. Typically, features are added incrementally, at various stages in the life cycle, usually by different developers, and often cutting across object boundaries. So when deployed, while each feature functions well on its own, they may not *interwork*. Namely, when several features are added to a service, or services are composed with each other there may be behavioural incompatibilities or modifications. This is known as the *feature interaction* problem [3, 13, 21, 4, 1]. While

feature interactions are not necessarily undesirable (the addition of any feature to the base system is an interaction!), we need at least to know of their existence, before determining desirability or resolution. This is known as *detection*, the subject of this paper. Although traditionally applied to telecommunications systems, the concept of feature interaction can equally apply to any distributed system in which features are offered.

We consider modelling features and analysing feature interactions in two different paradigms: a *telecommunications* system and an *email* system. Both systems consist of a network of basic components with different sets of features enabled. But the two paradigms represent different communication structures. The former is peer to peer, whereas the latter is client server.

In the first case, our model is based on the specification described more fully elsewhere [6] and follows the IN (*Intelligent Networks*) distributed functional plane [19]. In the second case, our model is based on a specification also described in more detail elsewhere [8], and is derived from Hall's email model [17].

To gain any real insight into the feature interaction problem, it is important to be able to infer properties for networks of *any* size, regardless of the underlying communication structure. But this is an example of the *parameterised model checking problem* (PMCP) which is, in general undecidable [2]. However, in some subclasses of systems the PMCP is decidable.

Our goal is to develop, in both paradigms,

- an interaction analysis which is *fully automated*, based on model-checking, and
- techniques to *infer* results about systems consisting of *any* number of components.

In the next section we introduce the concept of feature interaction, giving examples in the context of a telephone system and an email system. We explain the role of configurations and the novelty of our approach. In section 3 we describe the two network architectures that we will be considering. In sections 4 and 5 we give an overview of a basic service and feature behaviour in a telephony system and an email system respectively and in section 6 we provide a brief summary of the Promela implementation in each case.

In section 7 we give a brief overview of model checking and the model-checker Spin. We describe how model checking is used to perform feature interaction analysis on small, fixed size models of our examples and give the results. In section 8 we define PMCP and describe our solution, an abstraction technique. We apply it to our two example systems and give an outline of a proof of correctness. In section 9 we discuss our approach in the context of feature interaction analysis. Conclusions are in section 10.

We note that we have presented our basic Promela models and discussed feature interaction analysis for both paradigms elsewhere [6, 8]. In addition we have discussed our generalisation approach for the telecommunications example in [7] and for the email example in [8]. However, we have not compared the results for the two different communications architectures or provided any proof of the generalisation results in any previous publication.

The techniques presented here are motivated by feature interaction analysis, and illustrated in that context. However, they are, in principle, applicable to reasoning about about networks of other types of components such as objects and agents. The only requirement is for suitable data abstractions and characterisations of the of observable behaviour of sets of components.

2 Feature Interaction

2.1 Feature Interactions in Telephony

Feature interaction detection in telecommunications has been the topic of intense research over the last decade [5]. In a telephone system, control of the progress of calls is provided by a (software) service at an exchange (a *stored program control* exchange). This software must respond to events such as handset on or off hook, as well as sending control signals to devices and lines such as ringing tone or line engaged. A *feature* is additional functionality, for example, a *call forwarding capability*, or *ring back when free*; a user is said to *subscribe* to or *invoke* a feature.

An example of an interaction is the following. Suppose a user subscribes to *call waiting* (CW) and *call forward when busy* (CFB) and is engaged in a call. What happens when there is a further incoming call? If the call is forwarded, then the CW feature is clearly compromised. If the subscriber receives the call waiting signal, then the CFB is compromised. In either case, the subscriber will not have his/her expectations met. This is an example of a single component (SC) interaction – the conflicting features are subscribed to by a single user. More subtle interactions can occur when more than one user/subscriber are involved, these are referred to as multiple component (MC) interactions. Consider when user A subscribes to *originating call screening* (OCS), with user C on the screening list, and user B subscribes to CFB to user C. If A calls B, and the call is forwarded to C, as prescribed by B's CFB, then A's OCS is compromised. If the call is not forwarded, then we have the converse. These kind of interactions can be particularly difficult to detect (and resolve), since different features are activated at different stages of a call.

2.2 Feature Interactions in Email

The problem of feature interaction in email was first suggested and investigated by Hall [17], who presented a systematic methodology based on simulation and formal test coverage. An example of an interaction is the obvious (desirable) interaction between an encryption feature and a decryption feature. However, a more subtle interaction arises between a filtering feature and an autoresponse feature. Suppose that a client A filters messages from client B, and that client A also has the autoresponse feature. If a message from B arrives at A, will the message be simply discarded, or will an automatic response be sent back to B? In either event, one feature is compromised.

2.3 Configurations for Feature Interaction Detection

Feature interaction detection involves examining scenarios, e.g. component A with features f_1 and f_2 performs some action which affects components B and C with features f_3 and f_4 respectively. But in order to detect scenarios, one has to first determine the *configuration*: the number of components, and the enabled features for each component.

Nearly all analysis in the literature vary only one aspect of the configuration: the enabled features. They do not vary the number of components, but rather obtain results for a specific system of *fixed* size, i.e. for a system consisting of a *fixed* number of components. The results are (informally) assumed to hold for the general case, but there is no proof of such a generalisation.

In this paper, we first summarise feature interaction results obtained by model checking for systems of fixed size in both the telephone example and the email example. Second, we expand on the generalisation approach suggested in [7] and [8], to generalise our feature interaction results to networks of arbitrary size.

Our analysis is *pairwise*, known as 2-way interaction analysis. While at first sight this may appear limiting, empirical evidence suggests that 3-way interactions that are not detectable as a 2-way interaction are exceedingly rare [22].

2.4 Property based Approach

A property based approach to feature interaction detection assumes a formal model of the entire system and a given set of properties (usually temporal) associated with the features. Two features are said to *interact* if a property that holds for the system when only one of the features is present, does not hold when both features are present. For features f_1 and f_2 we define feature interaction as follows:

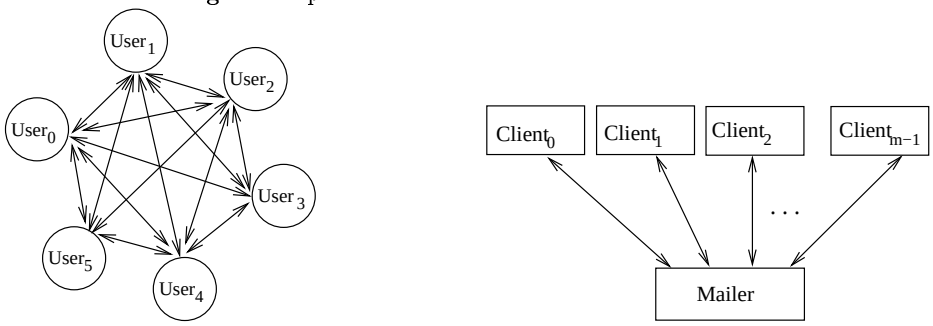
Definition 1. *Let $\mathcal{M}$ be the model of a system of N components in which neither f_1 or f_2 are present and $\mathcal{M}(f_1)$, $\mathcal{M}(f_2)$ and $\mathcal{M}(f_1 \cap f_2)$ models in which only f_1, only f_2 and both f_1 and f_2 have been added respectively. If ϕ_1 and ϕ_2 are properties that define f_1 and f_2 respectively then f_1 and f_2 are said to interact if $\mathcal{M}(f_1) \models \phi_1$ but $\mathcal{M}(f_1 \cap f_2) \not\models \phi_1$; or $\mathcal{M}(f_2) \models \phi_2$ but $\mathcal{M}(f_1 \cap f_2) \not\models \phi_2$.*

Note that this definition is relatively high-level, it does not contain details of the configuration. Thus it does not distinguish between SC and MC interactions. When we report on results later (section 7) we will make this distinction. Note also that this analysis will only reveal interactions that exist with respect to the particular properties ϕ_1 and ϕ_2. For complete analysis it may be necessary to perform analysis for a suite of properties for each feature, or to conjoin properties.

3 Communication within Telephone and Email Systems

The two communication mechanisms we consider are illustrated in Figure 1.

Fig. 1. Telephone and email network communication



(a) Peer to peer (b) Client server

In peer to peer communication every member of a network can communicate with every other member of the network.

An example of such a network is an (unfeatured) basic telecommunications system. In our system, communication between call components (*Users*) takes place via channels. There is one channel associated with each user. Each channel has capacity for at most one message: a pair consisting of a channel name (the other party in the call) and a status bit (the status of the connection). Figure 1 (a) illustrates communication channels within a telephone system with 6 User components.

The email system is an example of a system which uses client server communication. This system consists of a number of *clients* and one server, in this case the *mailer* component. Figure 1 (b) illustrates an email system for m Client components. In our email system, each client has a unique mail address. Clients send mail messages, addressed to other clients (or themselves) to the mailer; the mailer delivers mail messages to clients. Communication between client and server is asynchronous. Therefore, mail messages are not necessarily received by clients in the (global) order in which they were sent, but local temporal ordering is maintained, i.e. if client A sends messages 1 and 2 to client B, in that order, then client B will always receive message 1 before message 2 (though it may receive other messages in between).

4 Basic Telephone System and Features

Figure 2 is a high-level, abstract automaton for the basic call service behaviour (note the full implementation is somewhat more complicated, for example, some states (e.g. *unobtainable*) have been omitted). States to the left of the idle state represent *terminating* behaviour, states to the right represent *originating* behaviour. Transitions between states are triggered by *user-initiated* events at the terminal device, such as (handset) on and (handset) off, or by *communication* events on shared channels. We have excluded some trivial behaviour from the

automaton. For example, it is possible to perform a dial event (with no effect) from most states. Note also that while the state preidle is an important detail of the implementation (where local and global variables are reset), it does not play a part in the observable behaviour of a call component.

Originating and terminating automata can affect each other's behaviour through communication via (shared) channels. In the automaton, the channels are referred to as c, for the channel associated with that component, and p, for the channel associated with the partner component. In the originating side of the automaton, p is chosen *non-deterministically*. Otherwise, p is determined by the nature of incoming messages. We use the notation $c!x, y$ to denote *write* the value (x, y) to the channel c, $c!!x, y$ to denote *overwrite* the channel c with (x, y), $c? < x, y >$ to denote *poll* or non-destructively read value (x, y) from channel c, and $c?x, y$ to denote *destructively read* value (x, y) from channel c. When the value may be arbitrary, we use variables x and y; otherwise we use the actual constants required, e.g. 0,1, p, etc. If a read or write statement is written in italics it implies a *condition*. The appropriate action should be taken if and when the relevant channel is not empty/full. When there are two transitions from a particular state, one of which has a condition labelling it (e.g. from *calling*) the italicised transition should be taken if the condition holds.

A call component is not connected to, or attempting to connect to, any other call component when its associated communication channel is empty, When a communication channel is not empty, then the associated call component is *engaged in a call*, but not necessarily connected to another user. The interpretation of messages is described more comprehensively in [7].

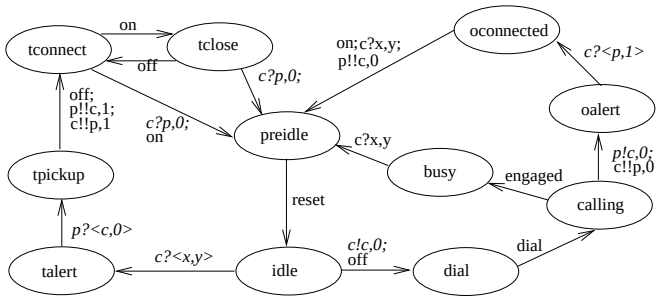


Fig. 2. Basic Call - States and Events

4.1 Features of the Telephone System

Consider a set of 7 features to be added to the basic call, with the following properties.

CFU – call forward unconditional Assume that $User[j]$ forwards to $User[k]$. If $User[i]$ rings $User[j]$ then a connection between $User[i]$ and $User[k]$ will be attempted before $User[i]$ hangs up.
CFB – call forward when busy Assume that $User[j]$ forwards to $User[k]$. If $User[i]$ calls $User[j]$ when $User[j]$ is busy then a connection between $User[i]$ and $User[k]$ will be attempted before $User[i]$ hangs up.
OCS – originating call screening Assume that $User[i]$ has $User[j]$ on its screening list, $i \neq j$. No connection from $User[i]$ to $User[j]$ is possible.
ODS – originating dial screening Assume that $User[i]$ has $User[j]$ on its screening list, $i \neq j$. $User[i]$ may not dial $User[j]$.
TCS – terminating call screening Assume that $User[i]$ has $User[j]$ on its screening list, $i \neq j$. No connection from $User[j]$ to $User[i]$ is possible.
RBWF – ring back when free Assume that $User[i]$ has RBWF. If $User[i]$ has requested a ringback to $User[j]$, $i \neq j$, (and not subsequently requested a ringback to another user) and subsequently $User[i]$ is idle when $User[i]$ and $User[j]$ are both free (and they are still free when $User[i]$ is no longer idle) then $User[i]$ will hear the ringback tone.
OCO – originating calls only Assume that $User[j]$ has OCO. No connection from $User[i]$ to $User[j]$ is possible.
TCO – terminating calls only Assume that $User[j]$ has OCO. No connection from $User[j]$ to $User[i]$ is possible.
RWF – return when free Assume that $User[j]$ has RWF. If $User[i]$ calls $User[j]$ when $User[j]$ is busy ($i \neq j$), then $User[i]$ will hear the *ret_alert* tone and $retnum[i]$ will be set to j (before $User[i]$ returns to the idle state).

The logic we use to formalise these properties is LTL – linear temporal logic. This logic has temporal operators [] (always), $\langle\rangle$ (eventually), X (next) and U (weak until), the only path operator is (implicit) universal quantification. Conjunction is denoted by $\wedge$. We do not give full details of all the LTL here, but give one example the formula for RBWF:

$$[]\neg((p \wedge q \wedge r \wedge s) \wedge ((p \wedge q \wedge r \wedge s)U((p \wedge (\neg q) \wedge r) \wedge ((\neg t)Uq))))$$

where $p = (rgbknum[i] == j)$, $s = (len(chan\_name[i]) == 0)$, $q = (User[i]@idle)$, $r = (len(chan\_name[j]) == 0)$, and $t = (network\_event[i] == ringbackev)$.

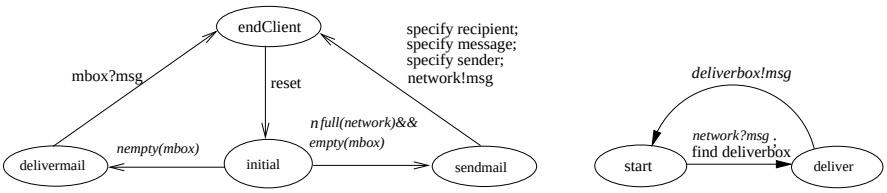
The propositions *p,q* etc. refer to the state of internal variables, these should be self-explanatory.

5 Basic Email Service and Features

High level, abstract automata for the client and mailer components are given in figure 3. Note that in these figures, some transitions are again labelled by conditions, e.g. in figure 3(a) a transition from *initial* to *sendmail* is only possible if the channel *mbox* is empty and the channel *network* is not full. Local and global variables are updated at various points; most variable assignments (apart from *reset*, in which all local variables are reset to their original values) are omitted from the diagrams. In order to avoid continuous blocking of the *network*

channel, Client behaviour must occur in one of two indivisible loops. If the Client's mailbox is non-empty, the Client can send a message in which case variables are reset, and the Client returns to the *initial* state. Otherwise, if the *network* channel is not full and the Client's mailbox is empty, the Client may send a message via the *network* channel. Again variables are reset and the Client returns to the *initial* state. Unlike the automata of figures 2 and 3(b) an indivisible global transition (or *atomic* statement – see section 6) is represented by one of these loops. As a result, in figure 3(a) notice that the write statement from the *sendmail* state to the *endClient* state is unitalicised. This is because this transition happens *immediately* after the transition from *initial* to *sendmail*, at which point it is established that the *network* channel is not full. Thus a write to this channel will be always be enabled. In figures 2 and 3(b) on the other hand, each transition between the *abstract* states (*idle*, *dial* etc. or *start* and *deliver* respectively) is an indivisible global transition.

Fig. 3. Email components



(a) Client component with mailbox *mbox*

(b) Mailer component

5.1 Features of the Email System

Hall's email model [17] included a suite of 10 features. Consider 7 of these features.

Encryption If $Client[i]$ has encryption on, then if $Client[j]$ receives a message whose sender is $Client[i]$, then the message will be encrypted.
Decryption If $Client[i]$ has decryption on, then all messages received by $Client[i]$ will have been decrypted.
Autoresponse If $Client[i]$ has autorespond on, then if $Client[j]$ sends a message to $Client[i]$, and $Client[j]$ hasn't already received an automatic response from $Client[i]$, then $Client[j]$ will eventually receive a reply from $Client[i]$. Alternatively, $Client[i]$ eventually stops sending messages because network can't be accessed.

Forwarding If $Client[i]$ forwards messages to $Client[j]$, then it is possible for $Client[j]$ to receive messages not addressed to $Client[j]$ (or to the default value M).
Filtering If Mailer filters messages from $Client[i]$ to $Client[j]$ then it is not possible for $Client[j]$ to receive a message from $Client[i]$.
Mailhost If mailhost is on and $Client[i]$ is a non-valid name $Client[j]$ sends a message to $Client[i]$ then, if $i \neq j$, $Client[j]$ will eventually receive a message from *postmaster*.
Remail Suppose that $Client[i]$ has the remailer feature and has a pseudonym of k. If $Client[i]$ sends a message to $Client[j]$, then $Client[j]$ will eventually receive a message from k. Also, if $Client[j]$ sends a message to k, then if $j \neq i$, $Client[i]$ will eventually receive a message from $Client[j]$.

Again, we do not give the LTL for all properties, the first five are given in [8]. The LTL for *mailhost* and *remail* are as follows:

Mailhost: $[](p -> \langle\rangle q)$ assuming $i \neq j$, $p = (last\_sent\_from[j]\_to == i)$ and $q = (last\_del\_to[j]\_from == pseud(i))$.

Remail: the conjunction of:
$[](((\neg p) \wedge X(p)) -> X(\langle\rangle q))$, where $p = (last\_sent\_from[i]\_to == j)$ and $q = (last\_del\_to[j]\_from == pseud(i))$, and
$[](((\neg p) \wedge X(p)) -> X(\langle\rangle q))$, assuming $i \neq j$, $p = (last\_sent\_from[j]\_to == pseud(i))$ and $q = (last\_del\_to[i]\_from == j)$.
Again, the propositions refer to internal variables and should be self-explanatory.

6 Implementation in Promela

Both example systems have been implemented in Promela, the source language for the model-checker Spin. Promela is an imperative, C-like language with additional constructs for non determinism, asynchronous and synchronous communication, dynamic process creation, and mobile connections, i.e. communication channels can be passed along other communication channels.

In the telecomms example, each call component (see figure 2) is an instantiation of the (parameterised) proctype *User*. Similarly, in the email model, each $Client$ component (see figure 3(a)) and the $Mailer$ component (see figure 3(b)) are instantiations of a *Client* and *Network_Mailer* proctype. Code relating to each *abstract* state (e.g. *idle*, *dial* etc. and *deliver*, *start* etc.), is contained within an atomic statement. The atomic statements cannot block because there are no *read* or *write* statements from a possibly empty or full channel contained within the atomic statement, except possibly at the beginning of the statement. Thus the statement is either unexecutable, or the whole statement will be executed as one. This ensures that every global transition in the resulting model involves a change in the abstract state of one component.

In both examples, features are added to the code by way of *inline* statements (a Promela procedure mechanism). In the telecomms example the *feature_lookup*

inline encapsulates centralised intelligence about the state of calls, i.e. what is known as single point call control. Calls at pertinent places in the code result in different behaviour according to whether relevant features are *switched on*. In the email example, a separate inline is included in the code for each feature. In most cases, feature implementation merely involves calls to these inlines to determine if the relevant feature is switched on. In general, the presence of the features simply results in additional transitions or steps during one or more of the abstract states of the client or mailer components. The exception is autorespond, because this feature involves both *reading* – a message from a client channel, and *writing* – a message to the network channel. Both events are potentially blocking, hence cannot take place within one atomic step. Therefore, to implement this feature, we add an additional data structure to indicate whether or not a client requires to send an autoresponse. We enhance the *initial* state to include the possibility that an autoresponse message needs to be sent, and give priority to this over any other event.

7 Model Checking

Model checking is an technique for verifying finite state systems. Systems are specified using a modelling language and the model – or *Kripke structure* [9] associated with this specification is checked to verify given temporal properties. In this section we give a a brief explanation of Spin, followed by a formal definition of model checking and results of feature interaction analysis for fixed sized systems.

7.1 Reasoning with SPIN

Spin [18] is the bespoke model-checker for Promela and provides several reasoning mechanisms: assertion checking, acceptance and progress states and cycle detection, and satisfaction of temporal properties, expressed in LTL.

Spin translates each component defined in the Promela specification into a finite automaton and then computes the asynchronous interleaving product of these automata to obtain the global behaviour of the concurrent system. This interleaving product is essentially a Kripke structure (see below), describing the behaviour of the system. It is this Kripke structure to which we refer when we talk about the *model* of our system. The set of states of this Kripke structure is referred to as the *state-space* of the model.

7.2 Kripke Structures and Model-checking

Definition 2. *Let AP be a set of atomic propositions. A Kripke structure over AP is a tuple $\mathcal{M} = (S, S_0, R, L)$ where S is a finite set of states, S_0 is the set of initial states, $R \subseteq S \times S$ is a transition relation and $L : S \to 2^{AP}$ is a function that labels each state with the set of atomic propositions true in that state.*

For a given model $\mathcal{M}$, and temporal property ϕ, model checking allows us to show that $\mathcal{M} \models \phi$. This is known as the *model checking problem*.

7.3 Feature Interaction Analysis for Models of Small Size

We give our feature interaction detection results for the two example systems.

Tables 1 and 2 indicate the feature interactions obtained for a telephone system and an email system with a small number of User/Client components. A x denotes no interaction, S and M denote single and multiple component interactions, respectively. In this section we limit ourselves to at most 4 components. Indeed, unless checking for MC interactions between a pair of filtering or forward features, 3 suffice. However we show in section 9 that for complete analysis it would be necessary to consider 5 or more components in some situations. Note that properties relating to forwarding and (in the email system) mailhost assume that $i \neq j$. It is important that our results are analysed to ensure that no false interactions are recorded. For example, although the filtering property is violated when features $filter[0] = 1$ and $filter[0] = 2$ are both selected, this is due to the fact that we only allow screening lists to have length 1 (so the second feature *overrides* the first). Similarly, we do not record an SC interaction for two forwarding features.

Table 1. Feature interaction results for the telephone example

	CFU	CFB	OCS	ODS	TCS	RBWF	OCO	TCO	RWF
CFU	M	S,M	S,M	×	M	×	×	×	×
CFB	S,M	M	S,M	×	S,M	×	×	×	×
OCS	×	×	×	×	×	×	×	×	×
ODS	M	M	×	×	×	×	×	×	×
TCS	×	×	×	×	×	×	×	×	×
RBWF	×	×	×	×	×	×	×	×	×
OCO	×	×	×	×	×	×	×	×	×
TCO	×	×	×	×	×	×	×	×	S
RWF	×	×	×	×	×	×	×	×	×

Table 2. Feature interaction results for the email example

	Encrypt	Decrypt	Filter	Forward	Autoresp	Mailhost	Remail
Encrypt	×	S,M	×	×	×	×	×
Decrypt	×	×	×	×	×	×	×
Filter	×	×	×	M	×	×	×
Forward	×	×	×	M	×	×	×
Autoresp	×	×	S,M	S,M	×	S,M	S,M
Mailhost	×	×	S	S	×	M	M
Remail	×	×	S	S,M	×	S,M	M

7.4 Use of Perl Scripts

For each pair of features, set of feature parameters, associated property and set of property parameters, a relevant model needs to be individually constructed to ensure that only relevant variables are included and set. For each example system, we have developed two Perl scripts, for automatically configuring the model and for generating model-checking runs. These scripts greatly reduce the time to prepare each model and the scope for errors. The results reported above were obtained using these scripts (running overnight). It is important to note that a certain amount of simple symmetry reduction is incorporated within the Perl script to avoid repeating runs of configurations which are identical up to renaming of components.

8 Any Number of Components

An obvious limitation of the model checking approach is that only finite-state models can be checked for correctness. Sometimes however we wish to prove correctness (or otherwise) of *families* of (finite-state) systems. That is to show that, if $\mathcal{M}_N = \mathcal{M}(p_0||p_1||\ldots||p_{N-1})$ is the model of a system of N concurrent instantiations of a parameterised component p, then $\mathcal{M}_N \models \phi$ for all $N \geq 1$.

This is known as the *parameterised Model Checking problem* which is, in general, undecidable [2]. The verification of parameterized networks is often accomplished via theorem proving [25], or by synthesising network invariants [10, 23, 26]. Both of these approaches require a large degree of ingenuity.

In some cases it is possible to identify subclasses of parameterised systems for which verification is decidable. Examples of the latter mainly consist of systems of N identical components communicating within a ring topology [15, 16] or systems consisting of a family of N identical *user* components together with a *control* component, communicating within a star topology [24, 16, 20]. A more general approach [14] considers a general parameterised system consisting of several different classes of components.

One of the limitations of both the network invariant approach and the subclass approach is that it can only be applied to systems in which each component (contained in the set of size N) is completely independent of the overall structure of the system: adding an extra component (to this set) does not change the semantics of the existing components. A generalisation of data independence is used to verify arbitrary network topologies [12] by lifting results obtained for limited-branching networks to ones with arbitrary branching.

All of these methods fail when applied to asynchronously communicating components like ours, where components communicate asynchronously via shared variables.

We describe an abstraction technique that is applicable to asynchronously communicating components, namely it enables us to infer properties of a model of system of any size from properties of a finite *abstract* model. The key idea is to define an abstract component – an environment – which represents the (observable) behaviour of a number of components. For a given m, we then examine the

behaviour of the system consisting of m (slightly modified) components and the abstract component. From this behaviour, we can infer the general behaviour. In the following, we apply the approach to our two examples, and outline a proof of correctness.

8.1 The *abstract* Model for the Telephone System

Let us first consider the telephone system. For any feature f, we say that f is *indexed* by $I_f = \{i_0, \ldots i_{r-1}\}$ if the feature relates to $User[i_0], \ldots, User[i_{r-1}]$. For example if f is "$User[0]$ forwards calls to $User[3]$", then f is said to be indexed by 0 and 3. Similarly we say that a property ϕ is indexed by a the set I_ϕ where I_ϕ is the set of User ids associated with ϕ. For a (possibly empty) set of features $F = \{f_0 \ldots f_{s-1}\}$ and property ϕ, we define the *complete index set* I of $\{\phi\} \cup F$, to be $I_{f_0} \cup \ldots \cup I_{f_{s-1}} \cup I_\phi$.

Suppose that we have a system S of N telephone components (with or without features) where $S = p_0||p_1|| \ldots ||p_{N-1}$ with associated model $\mathcal{M}_N = \mathcal{M}(S)$. For any $m \leq N$ we define an abstract system $abs_t(m)$ where

$$abs_t(m) = p'_0||p'_1|| \ldots ||p'_{m-1}||Abstract_t(m) \text{ if } m < N$$

or

$$abs_t(m) = p'_0||p'_1|| \ldots ||p'_{m-1} \text{ otherwise.}$$

(For simplicity we will assume from now on that $m < N$.) In this system, the p'_i, for $0 \leq i \leq m-1$ are *modified* User components and behave exactly the same as the original (*concrete*) components, p_i, $0 \leq i \leq m-1$ except that, for $0 \leq i \leq m-1$:

1. component p'_i no longer writes to (the associated channels of) any of the components $p_m, p_{m+1}, \ldots, p_N$ (the *abstracted* components), but there is a non-deterministic choice whenever such a write would have occurred as to whether the associated channel is empty or full (thus, whether the write is enabled or not).
2. An initial call request from any *abstracted* component to p'_i now takes the form $(out\_channel, 0)$, regardless of which abstracted component initiated the call. When such a message arrives on p'_i's channel, p'_i may read it. Henceforth p'_i no longer reads from (the associated channels of) any of the abstracted components. Instead, p'_i makes a non-deterministic choice over the set of possible messages (if any) that could be present on such a channel.

The component $Abstract_t(m)$ encapsulates part of the observable behaviour of all of the abstracted components. The component $Abstract_t(m)$ has $id = m$ and an associated channel named *out_channel*. A call initiation from an abstracted component to a concrete component is replaced by a message of the form $(out\_channel, 0)$ from $Abstract_t(m)$ to the relevant channel (*zero*, *one* etc.) which is always possible, provided the channel is empty. Any other message passing from the abstracted components is now represented by the non-deterministic choice available to the modified components, as described above.

In particular, suppose that $S = p_0||p_1||\ldots||p_{N-1}$ is a system of telephone components in which at least the features F are present. If ϕ is a property and only components $p_0, p_1, \ldots, p_{m-1}$ are involved in the features F or in ϕ then, regardless of whether components $p_m, p_{m+1}, \ldots, p_{N-1}$ have associated features or not (with some conditions attached), we will show that if ϕ holds for the model associated with $abs_t(m)$ (namely $\mathcal{M}_{abs_t(m)}$), then it holds for $\mathcal{M}(S)$. This case is illustrated in figure 4.

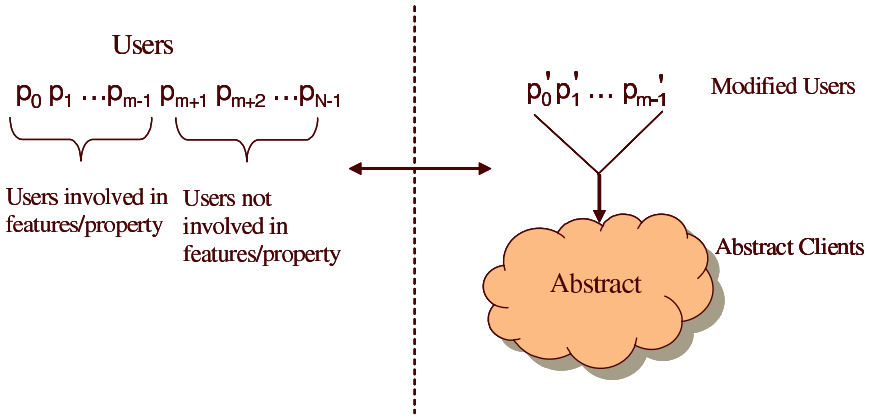


Fig. 4. Abstraction technique for N-User telephone model

In fact, because there is a bidirectional correspondence between $\mathcal{M}_{abs_t(m)}$ and

$$\mathcal{M}(p'_{i_0}||p'_{i_1}||\ldots||p'_{i_{m-1}}||Abstract_t(m'))$$

where $\{i_0, i_1, \ldots, i_{m-1}\}$ is a subset of $\{0, 1, \ldots, N-1\}$ of size m and m' the smallest element of $\{0, 1, \ldots, N-1\} \setminus \{i_0, i_1, \ldots, i_{m-1}\}$, we can extend this result to all cases where the total index set has size m. Thus we show:

Theorem 1. *Let $S = p_0||p_1||\ldots||p_{N-1}$ be a system of telephone components in which at least the features F are present, and ϕ a property.*

1. *If the total index set of $F \cup \{\phi\}$ is $\{0, 1, \ldots, m-1\}$ then if components $p_m, p_{m+1}, \ldots, p_{N-1}$ do not have any of the features CFU, CFB or TCS, $\mathcal{M}_{abs_t(m)} \models \phi$ implies that $\mathcal{M}(S) \models \phi$.*
2. *If the total index set of $F \cup \{\phi\}$ is $\{i_0, i_1, \ldots, i_{m-1}\}$ and σ the permutation that maps j to i_j for $0 \leq j \leq m-1$ and m to m' where m' is the smallest element of $\{0, 1, \ldots, N-1\} \setminus \{i_0, i_1, \ldots, i_{m-1}\}$, then if components $p_{\sigma(m)}, p_{\sigma(m+1)}, \ldots, p_{\sigma(N-1)}$ do not have any of the features CFU, CFB or TCS, $\mathcal{M}_{abs_t(m)} \models \phi$ implies that $\mathcal{M}(S) \models \sigma(\phi)$.*

Application of this theorem, for example, allow us to infer that many of the results of Table 1 (for networks of size 3 or 4) scale up to networks of arbitrary size. Proof of the theorem is outlined below.

Note that the features CFU, CFB and TCS are the only features in our feature set whose presence in the *partner* of a User component affects the behaviour of the User itself (the host). As can be seen from the proof below, this is the reason that their presence in the abstracted components is disallowed.

An outline of the Proof of Correctness of the Abstraction We will assume throughout that the components $p_0, p_1, \ldots, p_{m-1}$ do not have any features other than those contained in the set F. The first stage of the proof of correctness of Theorem 1 involves the construction of a reduced model $\mathcal{M}_r^m$ via data abstraction [11] for any $m \leq N$. First we give some definitions:

Definition 3. *Let $X = \{x_0, x_1, \ldots, x_{l-1}\}$ denote a set of variables such that each variable x_i ranges over a set D_i. Then $D = D_0 \times D_1 \times \cdots \times D_{l-1}$ is called the domain of X. A set of abstract values $D' = D'_0 \times D'_1 \times \cdots \times D'_{l-1}$ is called an abstract domain of X if there exist surjections $h_0, h_1, h_2, \ldots, h_{l-1}$ such that $h_i : D_i \to D'_i$ for all $0 \leq i \leq l-1$. If such surjections exist they induce a surjection $h : D \to D'$ defined by*

$$h((x_0, x_1, \ldots, x_{l-1})) = (h_0(x_0), h_1(x_1), \ldots, h_{l-1}(x_{l-1})).$$

In the following definition (taken from [9]) data abstraction is used to define a reduced structure whose variables are defined over an abstract domain:

Definition 4. *Let $\mathcal{M} = (S, R, S_0, L)$ be a Kripke structure with set of atomic propositions AP and set of variables X with domain D. If D' is an abstract domain of X and h the corresponding surjection from D to D' then h determines a set of abstract atomic propositions AP'. Let $\mathcal{M}'$ denote the structure identical to $\mathcal{M}$ but with set of labels L' where L' labels each state with a set of abstract atomic propositions from AP'. The structure $\mathcal{M}'$ can be collapsed into a reduced structure $\mathcal{M}_r = (S_r, R_r, S_0^r, L'_r)$ where*

1. *$S_r = \{L'(s) | s \in S\}$, the set of abstract labels.*
2. *$s_r \in S_0^r$ if and only if there exists s such that $s_r = L(s)$ and $s \in S_0$.*
3. *$AP_r = AP'$.*
4. *As each s_r is a set of atomic propositions, $L_r(s_r) = s_r$.*
5. *$R_r(s_r, t_r)$ if and only if there exist s and t such that $s_r = L'(s)$, $t_r = L'(t)$, and $R(s, t)$.*

The following lemma (which is a restriction of a result proved in [11]) shows how we may use a reduced structure $\mathcal{M}_r$ to deduce properties of a structure $\mathcal{M}$.

Lemma 1. *If $\mathcal{M}$ and $\mathcal{M}_r$ are a Kripke structure and a reduced Kripke structure as defined in definition 4 then for any LTL property ϕ, $\mathcal{M}_r \models \phi$ implies that $\mathcal{M} \models \phi$.*

We do not give full details of our reduced model $\mathcal{M}_r^m$ here, but instead give a brief description of the abstract domains involved.

The abstract domains of local variables of components $p_m, p_{m+1}, \ldots, p_{N-1}$ are the trivial set $\{true\}$. In $\mathcal{M}_N$ all other variables, apart from those associated with channel names or contents, have domains equal to the set $\{0, 1, \ldots, N-1\}$ (the set of component ids). Each of these variables have abstract domains equal to the set $\{0, 1, \ldots, m-1\}$ and a surjection from the original domain D to the abstract domain D' is given by $h_1 : D \to D'$ where

$$h_1(x) = \begin{cases} x & \text{if } x < m, \\ m & \text{otherwise} \end{cases}$$

for all $x \in D$. In $\mathcal{M}$, the domains of channel variables such as $self$ and $partner$, consist of the set of channel names $name[0], name[1], \ldots, name[N-1]$ (where $name[0]$, $name[1]$, etc. represent the channel names $zero$, one, etc.). The abstract domains for such variables is $name[0], name[1], \ldots, name[m]$ and the surjection h_2 is an obvious extension of h_1 above. Similarly abstract domains for the variables of contents of channels $name[0], name[1], \ldots, name[m-1]$ (a channel name and a status bit in each case) can be defined, and a surjection given in each case. The abstract domains for the variables of contents of channels $name[0], name[1], \ldots, name[m-1]$ are the trivial set.

From lemma 1 it follows that for any LTL property ϕ, $\mathcal{M}_r \models \phi$ implies that $\mathcal{M} \models \phi$.

The next stage of our proof involves showing that, for all $m \leq N$, $\mathcal{M}_r^m$ *simulates* $\mathcal{M}_{abs_t(m)}$. Again we provide some useful definitions:

Definition 5. *Given two structures $\mathcal{M}$ and $\mathcal{M}'$ with $AP \supseteq AP'$, a relation $H \subseteq S \times S'$ is a simulation relation between $\mathcal{M}$ and $\mathcal{M}'$ if and only if for all s and s', if $H(s, s')$ then*

1. *$L(s) \cap AP' = L'(s')$*
2. *For every state s_1 such that $R(s, s_1)$, there is a state s_1' with the property that $R'(s', s_1')$ and $H(s_1, s_1')$.*

If a simulation relation exists between structures $\mathcal{M}$ and $\mathcal{M}'$ we say that $\mathcal{M}'$ simulates $\mathcal{M}$ and denote this by $M \preceq M'$.

Lemma 2. *Suppose that $\mathcal{M} \preceq \mathcal{M}'$. Then for every LTL formula ϕ with atomic propositions in AP', $M' \models \phi$ implies $M \models \phi$.*

To prove that, for all $m \leq N$, $\mathcal{M}_r^m$ simulates $\mathcal{M}_{abs_t(m)}$, it is first necessary, for all $m \leq N$, to define a relation between the set of states of $\mathcal{M}_r^m$ (S_r^m say) and the set of states of $\mathcal{M}_{abs_t(m)}$ ($S_{a,t}^m$ say). Suppose V is the set of variables associated with $\mathcal{M}_N$ and V_r a reduced set of variables, such that V_r is identical to V except that the local and global variables associated with components $p_m, p_{m+1}, \ldots, p_{N-1}$ have been removed. The atomic propositions relating to $\mathcal{M}_N$ is the set $AP = \{x = y : x \in V \text{ and } y \in D(x)\}$, where $D(x)$ is the domain of x. Let us consider the alternative set of atomic propositions $AP' = \{x = y :$

$x \in V_r$ and $y \in D'(x)\}$, where $D'(x)$ is the abstract domain of x. If we let L_r denote the labelling function associated with AP', then we can define a relation H between S_r^m and $S_{a,t}^m$ as follows: For $s \in S_r^m$ and $s' \in S_{a,t}^m$, $H(s,s')$ if and only if $L_r(s) = L_r(s')$.

To show that H is a simulation relation, it is necessary to show that for all $(s,s') \in H$, every transition from (s,s_1) in $\mathcal{M}_r^m$ is matched by a corresponding transition (s',s_1') in $\mathcal{M}_{abs_t(m)}$, where $(s_1,s_1') \in H$. Every transition in $\mathcal{M}_r^m$ either only involves a change to the global variables or involves a change to the value of the local variables of one of the (concrete) components. If the former is true, then the transition involves an initial message being placed on the channel of one of the (concrete) components $p_0, p_1, \ldots, p_{m-1}$ by one of the components $p_m, p_{m+1}, \ldots, p_{N-1}$. This transition is reflected in $\mathcal{M}_{abs_t(m)}$ by a transition involving the *Abstract* component in which a message is placed on the channel of the concrete component. If t is a transition in $\mathcal{M}_r^m$ involving concrete component $p(i)$ then either t does not involve any component other than $p(i)$ or t involves only component p_i plus another concrete component or one or more of the following holds:

1. Component p_i is not currently in communication with another component and t involves a read from the channel of p_i as a result of the initiation of communication by one of the components $p_m, p_{m+1}, \ldots, p_{N-1}$.
2. Component p_i is currently in communication with one of the components $p_m, p_{m+1}, \ldots, p_{N-1}$ and t involves p_i reading a message from this component or
3. Component p_i is currently in communication with one of the components $p_m, p_{m+1}, \ldots, p_{N-1}$ and t involves a call to the *feature_lookup* function.

(Notice that a write to one of the components $p_m, p_{m+1}, \ldots, p_{N-1}$ does not involve a change of state, as the abstract domains associated with the channel contents of each of these components is trivial.)

Let $t = (s,s_1)$ and suppose that $H(s,s')$ for $s' \in S_{a,t}^m$. If $t = (s,s_1)$ does not involve any component other than p_i, or t involves p_i and another concrete component, there is clearly an identical transition $(s',s_1') \in \mathcal{M}_{abs_t(m)}$ such that $H(s,s')$.

If t involves a read from its channel of an initial message sent by one of the components $p_m, p_{m+1}, \ldots, p_{N-1}$ then t is reflected by a transition in $\mathcal{M}_{abs_t(m)}$ (p_i will still read such a message). However, if t involves any other read from one of the components $p_m, p_{m+1}, \ldots, p_{N-1}$, non-deterministic choice in p_i' (the corresponding component in $abs_t(m)$) ensures that an equivalent transition in $\mathcal{M}_{abs_t(m)}$ exists.

If p_i is currently involved in communication with one of the components $p_m, p_{m+1}, \ldots, p_{N-1}$ and t involves a call from p_i to the *feature_lookup* function then we must consider the cases of when components $p_m, p_{m+1}, \ldots, p_{N-1}$ have no associated features, and when they do have associated features, separately. (We have previously assumed that the components $p_0, p_1, \ldots, p_{m-1}$ do not have any features other than those contained in the set F.)

If components $p_m, p_{m+1}, \ldots, p_{N-1}$ have no associated features, then any guard g within the *feature_lookup* function that holds for a state in $\mathcal{M}_N$ (from which $feature\_lookup$ is called) will hold at the associated state in $\mathcal{M}_r^m$. If s is such a state and s' a state in $\mathcal{M}_{abs_t(m)}$ such that $H(s, s')$ then g holds at s' and it is clear that any transition t in $\mathcal{M}_r^m$ from s is reflected in $\mathcal{M}_{abs_t(m)}$.

However, if components $p_m, p_{m+1}, \ldots, p_{N-1}$ have associated features the situation is more difficult. Some guards within the *feature_lookup* inline depend only on whether the associated feature is present within the *host* component. Others depend on whether the feature is present within the *partner* component. Knowledge of the particular features that are present in the components $p_m, p_{m+1}, \ldots, p_{N-1}$ would be required to determine if the guards are true or not in the latter case. This scenario presents an open problem, which is beyond the scope of this paper. However, of our 9 features, guards relating to only 3 features depend on whether the partner component has the feature (namely CFU, CFB and TCS). Therefore, under the conditions of Theorem 1, we can conclude that there is a simulation relation between $\mathcal{M}_r^m$ and $\mathcal{M}_{abs_t(m)}$.

From Lemma 2 we can conclude that, for any LTL property ϕ, if components $p_m, p_{m+1}, \ldots, p_{N-1}$ do not subscribe to features CFU, CFB or TCS then, if $\mathcal{M}_{abs_t(m)} \models \phi$ then $\mathcal{M}_r^m \models \phi$ and the first part of Theorem 1 follows from Lemma 1.

It is straightforward to show that there is a correspondence between $\mathcal{M}_{abs_t(m)}$ and

$$\mathcal{M}(p'_{i_0}||p'_{i_1}||\ldots||p'_{i_{m-1}}||Abstract_t(m'))$$

where $\{i_0, i_1, \ldots, i_{m-1}\}$ is a subset of $\{0, 1, \ldots, N-1\}$ of size m and m' the smallest element of $\{0, 1, \ldots, N-1\} \setminus \{i_0, i_1, \ldots, i_{m-1}\}$. The second part of Theorem 1 follows as a result of this correspondence.

8.2 The *abstract* Model for the Email System

Suppose that we have a system S of $N-1$ Client components and a Mailer component where $S = M_0||p_1||p_2||\ldots||p_{N-1}$ with associated model $\mathcal{M}_N = M_S$. For any $m \leq N$ we define an abstract system $abs_e(m)$ where

$$abs_e(m) = M'_0||p'_1||p'_2||\ldots||p'_{m-1}||Abstract_e(m) \text{ if } m < N$$

or

$$abs_e(m) = M'_0||p'_1||\ldots||p'_{m-1} \text{ otherwise.}$$

In this system, M'_0 and the p'_i, for $1 \leq i \leq m-1$ are a *modified* Mailer component and *modified* Client components respectively. The p'_i behave exactly the same as the original (*concrete*) Client components except that they send/receive messages only to/from other concrete components or the abstract component (representing any of the other Client components). The $Mailer'$ component behaves exactly the same as the original $Mailer$ component except that the $Mailer'$ component no longer writes to (the associated channels of) any of the components $p_m, p_{m+1}, \ldots, p_{N-1}$ and a read from such components is replaced

by a non-deterministic choice. Thus the Mailer is modified in the same way that the p_i, $0 \leq i \leq m-1$ were modified in the telephone system abstraction.

The component $Abstract_e(m)$ encapsulates part of the observable behaviour of all of the abstracted components. The component $Abstract_e(m)$ has $id = m$ and can send messages to concrete components or to another abstracted component (via the *Mailer* component in each case).

In particular, suppose that $S = M_0||p_1||p_2||\ldots||p_{N-1}$ is a system of email components in which at least the features F are present. Let ϕ be a property and suppose that only components $p_1, p_2, \ldots, p_{m-1}$ are involved in the features F or in ϕ (we will assume that the Mailer component is not involved in the property, for ease of notation). Regardless of whether components $p_m, p_{m+1}, \ldots, p_{N-1}$ have associated features or not (again, with some conditions attached), we will show that if ϕ holds for the model associated with $abs_e(m)$ (namely $\mathcal{M}_{abs_e(m)}$), then it holds for $\mathcal{M}(S)$. This case is illustrated in figure 5.

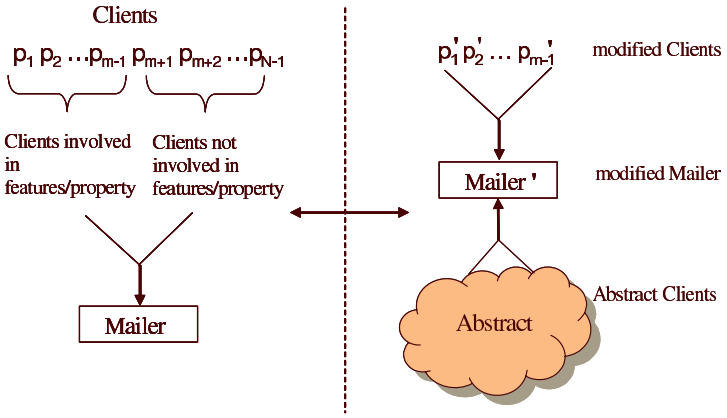


Fig. 5. Abstraction technique for N-Client email model

Because the inline functions relating to all of the features apart from encryption and decryption are called by the Mailer component (so behaviour of the Mailer depends on the features subscribed to by all parties in a communication, including abstract components), our results for the email case only hold when the abstracted components have no features apart from possibly the encryption or decryption features. We can prove the following:

Theorem 2. *Let $S = M_0||p_1||p_2||\ldots||p_{N-1}$ be a system of email components in which at least the features F are present, and ϕ a property.*

1. *If the total index set of $F \cup \{\phi\}$ is $\{1, 2, \ldots, m-1\}$ then if components $p_m, p_{m+1}, \ldots, p_{N-1}$ do not have any features apart from possibly encryption and/or decryption, then $\mathcal{M}_{abs_e(m)} \models \phi$ implies that $\mathcal{M}(S) \models \phi$.*

2. *If the total index set of* $F \cup \{\phi\}$ *is* $\{i_1, i_2, \ldots, i_{m-1}\}$ *and* σ *the permutation that maps* j *to* i_j *for* $1 \leq j \leq m-1$ *and* m *to* m' *where* m' *is the smallest element of* $\{1, 2, \ldots, N-1\} \setminus \{i_1, i_2, \ldots, i_{m-1}\}$*, then if components* $p_{\sigma(m)}, p_{\sigma(m+1)}, \ldots, p_{\sigma(N-1)}$ *do not have features apart from possibly encryption and/or decryption,* $\mathcal{M}_{abs_e(m)} \models \phi$ *implies that* $\mathcal{M}(S) \models \sigma(\phi)$.

The proof is similar to that of Theorem 1, and is omitted here.

9 The Abstract Approach and Feature Interaction analysis

The correctness of our approach, as presented above, is based upon theorems of the form

$\mathcal{M}_{abs(m)} \models \phi \Rightarrow \mathcal{M}(S) \models \phi$.

In the context of feature interaction detection, this means "if there is no interaction in the network of size m, with the abstract component, then we can infer that there is no interaction for *any* network (of arbitrary size)".

But what can we infer when there *is* an interaction? Namely, what can we infer when $\mathcal{M}_{abs(m)} \not\models \phi$? We cannot necessarily infer $\mathcal{M}(S) \not\models \phi$, because one is not a conservative extension of the other, i.e. there is only a simulation relationship, not a bisimulation relationship. For example, because of increased non-determinism, there may be additional loops in $\mathcal{M}_{abs(m)}$ which are not possible in any instance of $\mathcal{M}(S)$. Thus some liveness properties may actually hold in $\mathcal{M}(S)$, when they do *not* hold in $\mathcal{M}_{abs(m)}$. However, if the property does not hold for a network of size m (*without* the abstract component), i.e. $\mathcal{M}(p_{i_0}||p_{i_1}||\ldots||p_{i_{m-1}}) \not\models \phi$ (or $\mathcal{M}(p_{i_1}||p_{i_2}||\ldots||p_{i_{m-1}}) \not\models \phi$ in the email example) then we can infer $\mathcal{M}(S) \not\models \phi$. In practice, we have yet to encounter a false negative.

In the email example, for all combinations of features and associated property, $m \leq 4$ and full verification is possible. However, for some pairs of features in the telephone example, full analysis requires us to test scenarios where $m = 5$. For example, to fully analyse the pair of features CFU and TCS we must verify that, if $User[j]$ forwards to $User[k]$ and $User[l]$ screens calls from $User[m]$ then the CFU property (see [6]) holds. The CFU property has 3 parameters: j and k (as above) and a further parameter i. Hence, if i, j, k, l and m are all distinct, we have $m = 5$. In some situations where $m = 5$, and in a few (very rare) cases where $m = 4$, full verification is not possible under our current memory restriction of 3Gb.

10 Conclusions

Features are a structuring mechanism for *additional* functionality. When several features are invoked at the same time, for the same, or different components, the features may not interwork. This is known as *feature interaction*. In this paper we take a property based approach to feature interaction detection; this

involves checking the validity (or not) of a temporal property against a given system model. We have considered two example systems: a telecommunications system with peer to peer communication, and a client server email system.

A challenge for feature interaction analysis, indeed a challenge for reasoning about any system of components, is to infer properties for networks of *any* size, regardless of features associated with components, and the underlying communication structure. To solve this, for some cases, we have developed an inference mechanism based on abstraction. The key idea is to model-check a system consisting of a constant number (m) of components together with an *abstract* component representing any number of other (possibly featured) components. The approach is sound because there is a simulation between the fixed size system and any system, based on data abstraction. We have applied our approach to both examples and give a upper bound for the value of m (5).

The techniques developed here are motivated by feature interaction analysis, but they are also applicable to reasoning about networks of other types of components such as objects and agents, provided there is a suitable data abstraction and characterisations of the of observable behaviour of sets of components. The results can also inform testing. For example, the upper bound m allows one to configure (finite) tests which ensure complete coverage.

References

1. D. Amyot and L. Logrippo, editors. *Feature Interactions in Telecommunications and Software Systems VII*. IOS Press (Amsterdam), June 2003.
2. Krzysztof R. Apt and Dexter C. Kozen. Limits for automatic verification of finite-state concurrent systems. *Information Processing Letters*, 22:307–309, 1986.
3. L. G. Bouma and H. Velthuijsen, editors. *Feature Interactions in Telecommunications Systems*. IOS Press (Amsterdam), May 1994.
4. M. Calder and E. Magill, editors. *Feature Interactions in Telecommunications and Software Systems VI*. IOS Press (Amsterdam), 2000.
5. M. Calder, E. Magill, and S. Kolberg, Reiff-Marganiec. Feature interaction: A critical review and considered forecast. *Computer Networks*, 41/1:115 – 141, 2003.
6. M. Calder and A. Miller. Using SPIN for feature interaction analysis - a case study. In M.B. Dwyer, editor, *Proceedings of the 8th International SPIN Workshop (SPIN 2001)*, volume 2057 of *Lecture Notes in Computer Science*, pages 143–162, Toronto, Canada, May 2001. Springer-Verlag.
7. Muffy Calder and Alice Miller. Automatic verification of any number of concurrent, communicating processes. In *Proceedings of the 17th IEEE International Conference on Automated Software Engineering (ASE 2002)*, pages 227–230, Edinburgh, UK, September 2002. IEEE Computer Society Press.
8. Muffy Calder and Alice Miller. Generalising feature interactions in email. In Amyot and Logrippo [1], pages 187–205.
9. Edmund M. Clarke, Orna Grumberg, and Doron Peled. *Model Checking*. The MIT Press, 1999.
10. E.M. Clarke, O. Grumberg, and S. Jha. Verifying parameterized networks using abstraction and regular languages. In Insup Lee and Scott A. Smolka, editors, *Proceedings of the 6th International Conference on Concurrency Theory (CONCUR*

'95), volume 962 of *Lecture Notes in Computer Science*, pages 395–407, Philadelphia, PA., August 1995. Springer-Verlag.
11. E.M. Clarke, O. Grumberg, and D Long. Model checking and abstraction. *ACM Transactions on Programming Languages and Systems*, 16(5):1512–1542, January 1994.
12. S.J. Creese and A.W. Roscoe. Data independent induction over structured networks. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'00)*, volume II, Las Vegas, Nevada, USA, June 2000. CSREA Press.
13. P. Dini, R. Boutaba, and L. Logrippo, editors. *Feature Interactions in Telecommunication Networks IV*. IOS Press (Amsterdam), June 1997.
14. E. Allen Emerson and Vineet Kahlon. Reducing model checking of the many to the few. In David A. McAllester, editor, *Automated Deduction - Proceedings of the 17th International Conference on Automated Deduction (CADE 2000)*, volume 1831 of *Lecture Notes in Computer Science*, pages 236–254, Pittsburgh, PA, USA, June 2000. Springer-Verlag.
15. E. Allen Emerson and Kedar S. Namjoshi. Reasoning about rings. In *Conference Record of the 22nd Annual ACM Symposium on Principles of Programming Languages (POPL '95)*, pages 85–94, San Francisco, California, January 1995. ACM Press.
16. Steven M. German and A. Prasad Sistla. Reasoning about systems with many processes. *Journal of the ACM*, 39(3):675–735, July 1992.
17. R.J. Hall. Feature interactions in electronic mail. In Calder and Magill [4], pages 67–82.
18. Gerard J. Holzmann. The model checker Spin. *IEEE Transactions on Software Engineering*, 23(5):279–295, May 1997.
19. *IN Distributed Functional Plane Architecture*, recommmendation q.1204, ITU-T edition, March 1992.
20. C. Norris Ip and David L. Dill. Verifying systems with replicated components in Murϕ. *Formal Methods in System Design*, 14:273–310, 1999.
21. K. Kimbler and L.G. Bouma, editors. *Feature Interactions in Telecommunications and Software Systems V*. IOS Press (Amsterdam), September 1998.
22. M. Kolberg, E. H. Magill, D. Marples, and S. Reiff. Results of the second feature interaction contest. In Calder and Magill [4], pages 311–325.
23. R. P. Kurshan and K.L. McMillan. A structural induction theorem for processes. In *Proceedings of the eighth Annual ACM Symposium on Principles of Distrubuted Computing*, pages 239–247. ACM Press, 1989.
24. Robert P. Kurshan, M. Merritt, A. Orda, and S.R. Sachs. A structural linearization principle for processes. *Formal Methods in System Design*, 5(3):227–244, December 1994.
25. A. Roychoudhury and I. V. Ramakrishnan. Automated inductive verification of parameterized protocols. In Gérard Berry, Hubert Comon, and Alain Finkel, editors, *Proceedings of the thirteenth International Conference on Computer-aided Verification (CAV 2001)*, volume 2102 of *Lecture Notes in Computer Science*, pages 25–37, Paris, France, July 2001. Springer-Verlag.
26. Pierre Wolper and Vinciane Lovinfosse. Properties of large sets of processes with network invariants (extended abstract). In J. Sifakis, editor, *Proceedings of the International Workshop in Automatic Verification Methods for Finite State Systems*, volume 407 of *Lecture Notes in Computer Science*, pages 68–80, Grenoble, France, June 1989. Springer-Verlag.