# Automatic verification of any number of concurrent, communicating processes

Muffy Calder and Alice Miller
Department of Computing Science
University of Glasgow
Glasgow, Scotland.

E-mail: muffy,alice@dcs.gla.ac.uk

## Abstract

*The automatic verification of concurrent systems by model-checking is limited due to the inability to generalise results to systems consisting of any number of processes. We use abstraction to prove general results, by model-checking, about feature interaction analysis of a telecommunications service involving any number of processes. The key idea is to model-check a system of constant number ($m$) of concurrent processes, in parallel with an "abstract" process which represents the product of any number of other processes. The system, for any specified set of selected features, is generated automatically using Perl scripts.*

## 1 Introduction

The automatic verification of concurrent systems by model-checking has traditionally been limited due to the inability to generalise results to systems consisting of *any* number of processes. For example, we may be able to show that a property holds for 4 processes, i.e. for $p_0||p_1||p_2||p_3$, but how can we deduce that (if at all) the property holds for $p_0||p_1||p_2||p_3||\ldots||p_{n-1}$, for an arbitrary n? It is not possible to demonstrate this with straight-forward model-checking [1].

Our application domain is feature interaction analysis [2]. Analysis involves examining a system of basic service processes, running concurrently, some with additional features, to determine whether or not certain properties hold. An interaction may be indicated when a property holds in the presence of one feature alone (i.e. all but one process offers only the basic service) but is violated in the presence of more than one feature. It is important to know when analysis results do (and do not) scale up.

For example, for a system of four concurrent service processes, with different combinations of pairs of features, the goal of the analysis is to prove (or disprove) that $M(p_0||p_1||p_2||p_3) \models \phi[0,1,\ldots,t]$ where $M(p_0||p_1||p_2||p_3)$ is the finite-state model of the parallel composition of processes $p_0, p_1, p_2, p_3$ (instances of a parameterised process $p$) and $\phi[0,1,\ldots,t]$ is a temporal logic formula containing free variables indexed by $0,1,\ldots,t$, where $0 \leq t \leq 2$. The $p_i$ communicate peer to peer, asynchronously. In general, the $p_i$ are not isomorphic (because they have different sets of features enabled).

We have demonstrated a number of such results [3] for a basic telecommunications service with features modelled in Promela. The properties are specified in linear temporal logic (LTL) and verified using the SPIN model checker [6]. The $\phi[0,1,\ldots,t]$ express properties about feature behaviour (e.g. if process 1 has call forwarding to process 2 and process 0 initiates a call to process 1, then eventually a call from process 0 to process 2 will be attempted). In some cases it is necessary to consider more than 4 processes (up to 6) to fully capture all possible combinations.

The problem is how to generalise such results to any number of concurrent, communicating processes, i.e. to demonstrate that the property $\phi$ holds *regardless* of the number of processes involved (providing this number is sufficiently large). In this paper, we offer a solution based on abstraction.

We give a technique to prove that, for a fixed $m$ and $0 \leq t \leq m-1$, for any $n$, if $p_m, p_{m+1}, \ldots, p_{n-1}$ are isomorphic (they have no features enabled), then

$$M(p_0||p_1||p_2||p_3\ldots||p_{n-1}) \models \phi[0,1,\ldots,t].$$

The technique involves representing the behaviour of $p_m||\ldots||p_{n-1}$ by an abstract process, *Abs*. A model of the $m$ *concrete* processes $p_0, p_1, \ldots, p_{m-1}$ together with the abstract process is generated automatically from a model of the concrete processes together with a single basic call process $p$. For example if $m = 3$, a model of $p_0||\ldots||p_2||Abs$ is generated from a model of $p_0||\ldots||p_2||p$. In this case, for any $0 \leq t \leq 2$ we show, by model-checking, that $M(p_0||p_1||p_2||Abs) \models \phi[0,\ldots,t]$ and can hence infer that $\forall n. M(p_0||p_1||p_2||p_3\ldots||p_{n-1}) \models \phi[0,\ldots,t]$.

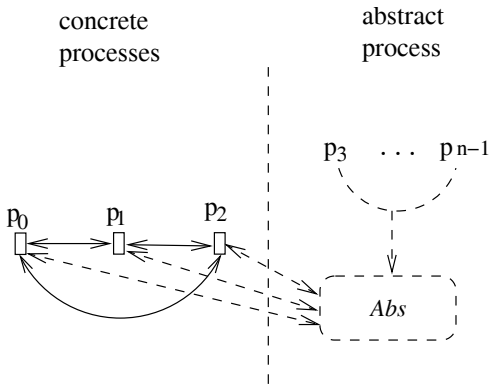


**Figure 1. Representing n processes**

The technique is summarised by Figure 1. The concrete processes are $p_0, p_1, p_2$. Communication between concrete processes is via channels (denoted by rectangles) and is unchanged. Note that the contents of a communications channel fundamentally determines process behaviour. The *observable* behaviour of processes $p_3 \ldots p_{n-1}$ is represented by *Abs* in the following way. Communication to/from a concrete process from/to any other process takes place via a *virtual* channel. Rather than concrete processes reading/writing to this (virtual) channel and behaving accordingly, each possible read is replaced by a non-deterministic choice over the possible contents of such a channel. In this way all possible behaviours are explored. (A write to such a channel is no longer relevant.)

The number of concrete processes that are required depends on the number of distinct variables that occur in the feature descriptions and the property to be verified. We have shown [4] that if two features are enabled in total, then for our specific set of properties at most 5 concrete processes are required (that is, $m \leq 5$).

The generation of the initial model on $m + 1$ processes (for any specified set of selected features) is automatically generated from a template, using Perl scripts. Similarly, the conversion from an $m + 1$ process model to an $m$ process plus abstract model is performed automatically.

## 2 Basic call service and features

### 2.1 Basic call

The basic call service permits call set-up and tear-down between two parties. Call control is asymmetric: one party has *originating* behaviour, and controls the call, the other has *terminating* behaviour. Our model follows the *IN* (Intelligent Networks) model, distributed functional plane.

Figure 2 gives a diagrammatic representation of an abstract automaton for the basic call service behaviour (the full implementation is somewhat more complicated). 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. Trivial behaviour is omitted.

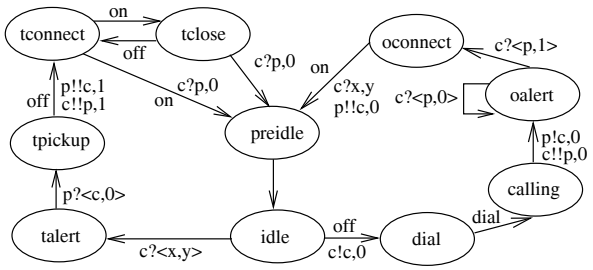


**Figure 2. Basic call service behaviour**

**Figure 3. Interpretation of channel states**

| *channel A* | *interpretation* |
|---|---|
| empty | A is free |
| (A,0) | A is engaged, but not connected |
| (B,0) | B is terminating party, attempting connection |
| (B,1) | if A contains (A,1) then A and B are connected |

Originating and terminating automata influence each other's behaviour through communication via (shared) channels. In Figure 2 the channels are referred to as $c$, for the channel associated with that process, and $p$, for the channel associated with the partner process. $p$ is chosen *non-deterministically*. 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. Channels may contain channels.

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). The interpretation of messages is given in Table 3.

### 2.2 Features

Since the purpose of this paper is not to describe the features but to to illustrate the abstraction process, we therefore refer only to the **CFU** feature – *call forward unconditional*.

For illustration, we provide the LTL formula for CFU but not the detailed description of the (parameterised) Promela process *User*, its variables and associated proposi-

tions. $att(i,k)$ denotes the proposition "a call is attempted from $User[i]$ to $User[j]$" which is itself defined in terms of the global variables associated with the $User$ process (see [3]).

**Property 1 – CFU** Assume that $User[j]$ has CFU $User[k]$. *If* $User[i]$ *rings* $User[j]$ *then a connection between* $i$ *and* $k$ *will be attempted before* $User[i]$ *has handset on.*
LTL: $[](p \rightarrow (r\mathcal{U}q))$
$p$ = $((dialed[i] == j)\&\&(User[i]@calling))$,
$r$ = $att(i,k)$, $q$ = $(dev[i] == on)$.

## 3 Feature validation and interaction analysis

The basic idea of feature interaction analysis is detecting when features behave as expected in isolation, but not in the presence of each other. Analysis involves feature *validation* (checking a feature in isolation) and then checking feature *tuples* for violation of expected behaviour.

### 3.1 Analysis of any number of call processes

For a given number of features present, and the property to be verified, there is a fixed number of processes $m$ for which proof of the property is sufficient to prove the property for any number of processes. For example, suppose we wish to prove the (CFU) property above with $i = 0$, $j = 1$, $k = 2$ for the model representing the behaviour of $User[0]||User[1]||\ldots||User[n-1]$ when $User[1]$ has CFU to $User[2]$ and the other users are basic call processes, for any $n$. Proof of the property for $n = 4$ seems to be sufficient (the three processes involved in the feature and the property, plus an external process). But why and when is it is sound to make such a conclusion? Clearly, any external process (that is, any process not involved in the feature or the property) can only affect the behaviour of any of the processes involved in the feature or property via communication to or from such a process. Therefore, as long as all possibilities of such communication is considered, the (internal) behaviour of the external processes does not affect the truth (or otherwise) of the property. The following theorem (stated here without proof) clarifies this reasoning.

**Generalisation Theorem** Suppose that $N_n = N_n(p_0, p_1, \ldots, p_{n-1})$ is a network of $n$ processes such that, for $m \leq i \leq n-1$, $p_i$ has no features enabled. If $M(N_m||Abs) \models \phi[0,1,\ldots,t]$ where $M(N_m||Abs)$ is the finite-state model representing $N_m$ acting concurrently with the abstract process $Abs$ (described earlier), $\phi[0,1,\ldots,t]$ is a temporal logic formula containing free variables indexed by $0,1,\ldots,t$, where $0 \leq t \leq m-1$, then $M(N_n) \models \phi[0,1,\ldots,t]$.

The consequences of the theorem are the technique outlined in Figure 1 where we represent the behaviour of $p_m||\ldots||p_{n-1}$ by *Abs*. A model of the $m$ *concrete* processes $p_0, p_1, \ldots, p_{m-1}$ together with the abstract process $Abs$ is generated automatically from a model of the concrete processes together with a single basic call process $p$. For example if $m = 3$, a model of $p_0||p_1||p_2||Abs$ is generated from a model of $p_0||p_1||p_2||p$. In this case, for any $0 \leq t \leq 2$ we show, by model-checking, that $M(p_0||p_1||p_2||Abs) \models \phi[0,\ldots,t]$ and can hence infer that $\forall n.M(p_0||p_1||p_2||p_3\ldots||p_{n-1}) \models \phi[0,\ldots,t]$.

We refer to the $m$ process model plus abstract process model as the $N$-*users* model. Due to lack of space we do not give details here of how it is generated, but the full Perl script to convert any $(m+1)$-users model (with $m = 3$) to an $N$-users model is given in [4].

**Table 1.**

| 3-user | Depth | States | Mem | Time | State-vector |
|---|---|---|---|---|---|
| | 0.09 | 0.07 | 0.3 | 2.8 | 96 |
| 4-user | Depth | States | Mem | Time | State-vector |
| | 7.0 | 4.2 | 16.1 | 80 | 116 |
| *N*-user | Depth | States | Mem | Time | State-vector |
| | 2.0 | 0.8 | 3.2 | 17.5 | 116 |

## 4 Results

### 4.1 Validation of single features

We have validated each feature for the $N$-users model. For the benefit of readers with a knowledge of SPIN, in Table 1, we provide results for the CFU property for the 3-user model, the 4-user model and the $N$-user model respectively. For all verification runs we used a Sun Ultra™ 80 workstation with four 450MHz UltraSPARC-II CPUs and 2GB of main memory running the Solaris 7 operating system. Notice how the values of *Depth* ($\times 10^6$), (stored) *States* ($\times 10^5$), *Mem* (Mbytes, with compression), *Time* (user + system, in seconds) and *State-vector* (bytes) for the $N$-user model lie in-between those values corresponding to the 3-user model and 4- user model respectively. [1]

### 4.2 Pairwise analysis of features

Once, for a given pair of features and a given property, the number of concrete users required has been established, it is straightforward to perform pairwise analysis. We give here a simple example where the number of concrete users required is 3 and $User[1]$ has CFU to $User[2]$ and $User[2]$ has CFU to $User[0]$.

[1] Note that the *Depth* parameter indicates the number of steps (including individual steps within atomic statements) involved in the longest path, which is more than the number of states encountered on such a path.

There is an interaction. The scenario offered as a counter-example (by SPIN) is when an attempted call from $User[0]$ is forwarded to $User[0]$, not to $User[2]$.The same scenario is offered during verification of the corresponding 4-user model. Even with two features, the number of concrete users required may be greater than 3. For example, consider the same property when the second feature above is replaced by $User[3]$ has CFU to $User[4]$. This would require 5 concrete user processes.

## 5 Discussion

The results above clearly demonstrate the feasibility of the abstraction technique for this application domain – the model checking requirements are well within the capability of our machine. The transformation to a $N$-user model is relatively straightforward: we need only consider the *communication* between the external processes and the concrete processes. On the other hand, an induction approach [5, 8] requires the construction of an inductive invariant. This is a non-trivial exercise as it involves incorporating the behaviour of the entire system within the invariant.

The abstraction approach is sound because the resulting abstraction is conservative. While the approach is *potentially* unconservative, because the resulting system offers *all* possible communication between concrete and abstract processes, the communications protocol is strong enough to prevent any incorrect communication from taking place. Overall soundness is assured because processes only communicate via specified shared variables and interprocess communication is mediated by a strong protocol. Formalisation of this argument to a general setting is further work.

While methods based on a combination of abstraction and model-checking [7, 9] have been applied in other domains, we are unaware of any previous generalisation results in the feature interaction domain.

## 6 Conclusions

The automatic verification of concurrent systems by model-checking has traditionally been limited due to the inability to generalise results to systems consisting of any number of processes. This may be a serious limitation because it is often important to show that results do, or do not, scale up.

We show a technique that can be used to prove general results about an arbitrary number of processes. The technique does not involve explicit induction, and consequently is rather simpler to apply. The key idea is to consider a system of constant number ($m$) of concurrent processes, in parallel with one *abstract* process which represents the product of any number of other processes.

We have applied the technique to feature interaction analysis of a telecommunications service. The general system, for any specified set of selected features, is generated automatically using Perl scripts. Empirical results demonstrate feasibility of the approach and that interaction results scale up.

## Acknowledgments

The authors would like to thank Ken McMillan for valuable discussions relating to this work.

## References

[1] Krzysztof R. Apt and Dexter C. Kozen. Limits for automatic verification of finite-state concurrent systems. *Information Processing Letters*, 22:307–309, 1986.

[2] M. Calder and E. Magill, eds. *Feature Interactions in Telecommunications and Software Systems*, volume VI. IOS Press, Amsterdam, 2000.

[3] M. Calder and A. Miller. Using SPIN for feature interaction analysis - a case study. In *Proceedings of SPIN 2001*, *LNCS*, vol. 2057, Springer-Verlag. pp. 143–162, 2001.

[4] M. Calder and A. Miller. Feature validation for any number of processes. TR2002-110, Dept. of Computing Science,University of Glasgow, 2002.

[5] E.M. Clarke, O. Grumberg, and S. Jha. Verifying parameterized networks using abstraction and regular languages. In *Proceedings of CONCUR '95*, *LNCS*, vol. 962, Springer-Verlag. pp. 395–407, 1995.

[6] Gerard J. Holzmann. The model checker Spin. *IEEE Transactions on Software Engineering*, 23(5):279–295, May 1997.

[7] C. Norris Ip and David L. Dill. Verifying systems with replicated components in mur$\phi$. *Formal Methods in System Design*, 14:273–310, 1999.

[8] R. P. Kurshan and K.L. McMillan. A structural induction theorem for processes. In *Proceedings of 8th ACM Symposium on Principles of Distributed Computing*, pages 239–247. ACM Press, 1989.

[9] K. L. McMillan. Parameterized verification of the flash cache coherence protocol by compositional model checking. In Proceedings of CHARME 2001, *LNCS*, vol. 2144, pp. 179–195, 2001.