OPTIMISING COMMUNICATION STRUCTURE FOR MODEL CHECKING.

Peter Saffrey
Department of Computing Science, University of Glasgow, Glasgow

Key words to describe this work: Model checking, formal verification, communication structure, state space explosion

Key Results: An example of how communication structure can effect the size of a resultant state space.

How does the work advance the state-of-the-art?: A new technique for tackling concurrent systems which are intractable to verify.

Motivation (Problems addressed): Many communicating concurrent systems are currently intractable. Communication structure has not yet been exploited to address this problem.

Introduction

Model checking is a formal technique that can be used in the analysis of *concurrent systems*. Model checking proves properties for a given system using *state space exploration*, an exhaustive search of all system behaviour. However, for some systems, the large combination of possible behaviours generate intractably large state spaces. This is known as the *state space explosion problem.*

We attack the state space explosion problem by exploiting *communication structure*: the mechanism used to model the communication between concurrent processes.

Approach Motivation

Figure 1 shows the usual procedure used to apply model checking.

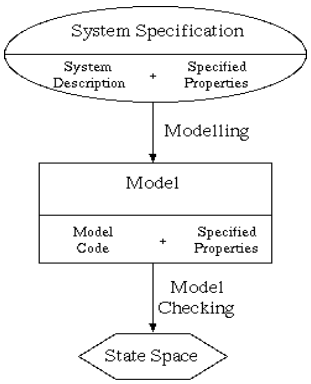

Figure 1: The usual modelling Procedure

A *system specification* is *modelled* in a language designed for model checking. A model checker then explores the state space for that model. Modelling is analogous to the conversion of a specification into software, often referred to as *programming.*

Figure 2 shows an alternative view which motivates our work.

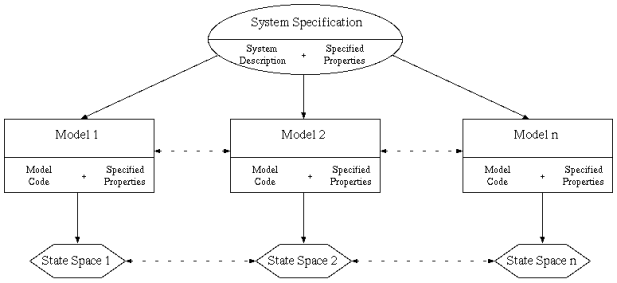

Figure 2: An alternative view of the model checking procecure

In this view, there are a number of alternative models, each based on the same specification. Using the programming analogy, alternative programs based on the same specification will differ in terms of speed, memory footprint, ease of maintenance or other measure of efficiency. In model checking the alternative models may differ in terms of state space size.

Our method aims to choose a *communication structure* that conforms to the specification and results in the smallest possible state space. We also address the relationship between different communication structures for the same system, illustrated in figure 2 by the horizontal arrows connecting models.

Communication Structure

In our method, communication is modelled using *channels*, first-in-first-out buffers that can be shared between any number of processes. We deal solely

with *asynchronous* communication, where the sending and receiving of messages occur at different times.

Channels can often be used in a variety of ways to form the desired connections between processes. Figure 3 shows 2 *channel diagrams* illustrating alternative communication structures for an example system, in this case a set of Internet Service Providers (ISPs) exchanging e-mails with each other. Note these diagrams illustrate *models* of the system, not final system architectures.

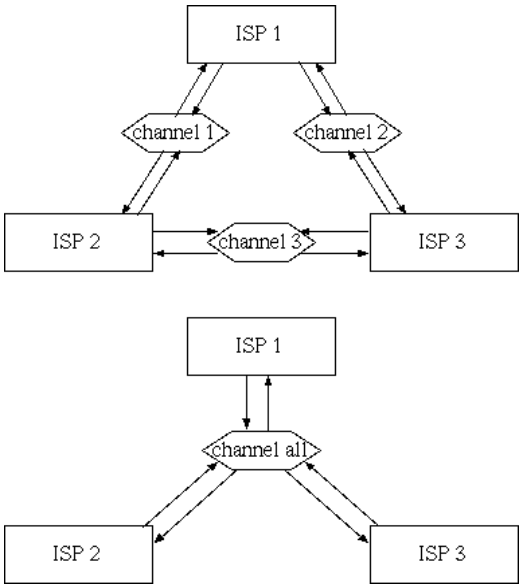

Figure 3: Two channel structures for an e-mail system. Top: one channel per connection; bottom: one channel shared by all.

In each channel diagram, a rectangle represents a process, and a hexagon a channel, with directional arcs indicating which processes send and receive on each channel. In figure 3, the *initial* channel structure (top) uses a single channel to mediate the communication between each pair of processes. The candidate channel structure (bottom) uses a single channel to mediate all communication. We assume all channels are of size 1 and that nothing other than the channel structure is altered. By reducing the number of channels, there will be fewer possible combinations of messages in transit at any one time. This should result in a smaller state space.

Reducing the number of channels is not always the most effective way to reduce the state space size. In some cases, a channel structure can be designed to increase the effectiveness of reduction techniques, or to take advantage of the construction of a particular system.

Property Preservation

Although the two channel structures shown in figure 3 form the same connections between processes, the behaviour of the two structures is not identical. For example, in the candidate structure, only one message can be passed at a time, whereas the initial structure allows multiple messages.

We have devised a method for showing *property preservation* between channel structures, which we will describe briefly here.

Any behaviour which is possible for one structure, but not possible for another is a *difference* behaviour. To show that a property is preserved between the two structures, we must show that the difference behaviour is irrelevant to the verification of that property. In order to achieve this, we try to *emulate* the *effect* of difference behaviour with *non*-difference behaviour. The *effect* of a behaviour is defined with respect to the property in question.

Results

Using the model checker *Spin*, we verified the example communication structures from figure 3.

Table 1 shows the number of states (to 3 significant figures) explored when verifying for **deadlock**. We also verify the *linear temporal logic* (LTL) property **notify** which checks that a valid e-mail will always arrive.

	Initial	**Candidate**
deadlock	1.14e+06	1.19e+05
notify	?	9.11e+06

Table 1 Results for initial and candidate communication structures

In the case of **deadlock**, the state space size of the candidate structure is only 10% of the initial structure. The property **notify** was intractable with the initial structure, but with the candidate structure, it can be verified.

Conclusion

Altering the communication structure of a concurrent system can clearly reduce the resultant state space. However, further work is needed to determine which communication structures result in the smallest state spaces.