Formal Verification of Arbitrary Network Topologies*

S. J. Creese and A. W. Roscoe

Oxford University Computing Laboratory
Wolfson Building, Parks Road
Oxford OX1 3QD, UK

Abstract *We show how data independence results can be used to generalise an inductive proof from binary to arbitrary branching tree networks. The example used is modelled on the RSVP Resource Reservation Protocol. Of particular interest is the need for a separate lower-level induction which is itself closely tied to data independence. The inductions combine the use of the process algebra CSP to model systems and their specifications, and the FDR tool to discharge the various proof obligations.*

Keywords: induction, data independence, CSP, FDR, model-checking.

1 Introduction

Our ability to formally verify systems of realistic size is inevitably restricted (in part) by the limitations of our chosen tool support. For example, it is not possible to use a finite space model checker to reason about a system which may be of arbitrary size (since its parameterisation might lead to an unboundedly large state space). In these cases the largest check which could be performed would depend on the maximum size the model checker could handle. Whilst this can provide some confidence in the system's integrity, it is not a proof that the system meets its specifications for an arbitrary sized parameter/number of components. This paper presents a technique which is designed to overcome these limitations for certain types of scalable systems. In particular, the technique uses a unique combination of induction and data independence to formally establish properties of arbitrary branching networks. The technique is implemented using the process algebra CSP[6, 11] to model systems, and the FDR model checker tool [5] to help reason about them, though we have no doubt that our ideas would work in other contexts as well.

CSP is a process algebra which is useful for describing systems that interact by communication. The collection of mathematical models and associated semantics that make up CSP facilitate the capture of a wide range of process behaviours. The theory of refinement in CSP allows correctness conditions to be encoded as refinement checks between processes. Refinement is transitive; if process S is refined by process R (written $S \sqsubseteq R$), and R is refined by T then S is refined by T. The FDR tool takes a machine-readable dialect of CSP_M as its input syntax, and can be used to check refinements as well as determinism, deadlock freedom and livelock freedom of processes.

This paper is organised as follows: we briefly survey existing work on structural induction and the theory of data independence we will be using; we then present the method; finally, we present our conclusions.

2 Structural Induction

It has been demonstrated that induction is a method which can be successfully used in the analysis of distributed systems. Kurshan and

*The work reported in this paper was supported by DERA Malvern and the US Office of Naval Research

McMillan [7] and Wolper and Lovinfosse [14] published similar work using structural induction to reason about systems with unboundedly large numbers of identical components. Both methods require an invariant to be defined and rely on proof obligations which correspond to the base case and inductive step (using model checkers to discharge them). The inductive step(s) corresponding to rules for construction of the associated networks. In [1, 2, 12] a similar type of structural induction scheme was perfected for CSP, utilising the FDR tool to discharge the various proof obligations. It is this type of structural induction which forms part of the technique we present below.

The basic induction method involves using FDR to discharge proof obligations which take the form of refinement relations between processes. Consider (1) and (2) below:

$$P \sqsubseteq Core \quad (1)$$
$$P \sqsubseteq P[right \longleftrightarrow left]Node, \quad (2)$$

where the alphabet of $Core$ and P is $\{right\}$, and the alphabet of $Node$ is $\{left, right\}$. The linked parallel operator used in (2), ($[right \longleftrightarrow left]$), has the effect of piping the two processes: renaming the two channels to the same, putting the relevant processes in parallel over that channel, and hiding the communications between them.

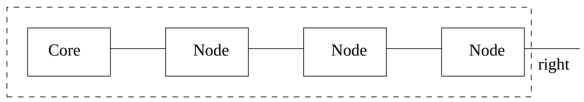

Figure 1: *A chain of nodes with all events inside the dashed box hidden.*

Refinement (1) corresponds to the base case of the induction, and (2) the inductive step. It is clear by transitivity of refinement that (1) and (2) can be combined to prove that a *Core* process can be piped, on its *right* channel, into the *left* channel of any arbitrary number of *Node* processes piped together as given in (2), and the resulting network will always satisfy property P. Figure 1 above shows a typical system one could create.

In [1, 2] this induction technique was used to prove various end-to-end properties of arbitrary binary tree networks which modelled aspects of the resource reservation protocol RSVP[1]. RSVP is a protocol designed to support reservation for high-bandwidth multicasts over IP networks. Resource reservations are created and maintained along each link of a previously determined mulitcast route, where routes consist of multiple sources and receivers connected by arbitrary numbers of intermediate nodes. Messages requesting amounts of bandwidth originate at receivers and are passed upstream towards the source. At any node, if a resource reservation is in place for an amount of bandwidth then it may be shared by all receivers downstream, removing any unnecessary duplication and reducing network traffic. If at any intermediate node a request is rejected a reject message is passed downstream and the request discarded. Otherwise, requests are propagated as far as the closest point along the way to the source where a reservation level greater than or equal to it has been made.

It is this traffic-reducing property which is modelled in [1, 2]. Here the protocol nodes are binary in that they possess two downstream channels and one upstream, in contrast a source process only has a downstream channel (since it is the ultimate destination). The property proven is that the receivers always receive a response to each unique request for bandwidth (or, that the interface presented by the network to a receiver at any point has the behaviour of a source process independently of how many intermediate nodes lie along the path between source and receiver). It is established that this correct behaviour is presented on a particular interface no matter what events are performed elsewhere on the network. This is achieved by lazily abstracting all the network's behaviour which is not on the path between the source and receiver under consideration. To lazily abstract a channel means to hide the channel in such a way that we do not

[1] A description of RSVP can be found at `http://www.isi.edu/div7/rsvp/.index.html`.

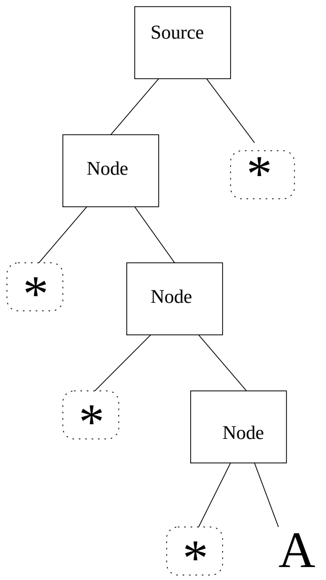

Figure 2: *A Binary Tree. Dotted boxes containing * represent the lazy abstraction of all events on that channel.*

assume that hidden events must occur. It provides the best way of formulating what a process looks like to an observer who can only see a subset of its alphabet (see Chapter 12 of [11]).

So in Figure 2 the correct behaviour presented on the downstream branch at A is independent of the behaviour of the rest of the network at the points marked $*$, achieved by the lazy abstraction of all events occurring at the points marked $*$. In this case there would be two refinement relations corresponding to the inductive step, one for each of the downstream channels of a *Node* where the other has been lazily abstracted away.

Using this type of structural induction it would be possible to prove properties of an arbitrary tree constructed from certain nodes. However, in such a proof the maximum degree of branching will be fixed, and separate inductive cases need to be model-checked for each branching degree. The technique we present in this paper allows *all* degrees of branching to be dealt with at once with a few checks, by using the techniques of *data independence* to generalise arguments.

3 Data Independence

A system is said to be data independent of a data-type variable T when it makes sense for any non-empty substituted type and it satisfies structural rules, meaning that it handles the type in a relatively simple way. The process representing the system may not perform any operations on values of that type, it can only input them, store them, output them, and perform equality tests between them. Many communications protocols are data independent since they simply pass around data ensuring that only desirable recipients are able to extract it; their behaviour is entirely independent of the data itself.

Data independence ideas have been developed for a number of notations, having first been studied formally in [13], but we use here the theory developed for CSP by Lazić and Roscoe (see [8, 9] and Section 15.2 of [11]). They have developed theorems which state that for certain (data independent) systems it is possible to verify that the system possesses certain properties, for all instances of its independent types, by performing a specific finite number of checks with finite instances of the types. Data independence theorems frequently allow us to generate thresholds for given checks: a size of type T such that one or more checks of a property for this and perhaps smaller types will imply that the property holds for all T. Space unfortunately prevents us from quoting in detail the various theorems and definitions of data independence, for which we refer the reader to [8, 9, 11].

However, of particular relevance to the technique we present in this paper is the ability to consider data independent processes with the addition of constant symbols and (in general many-valued) predicates on variable types The theorems used put some constraints on the use of these predicates. They must be uninterpreted, in that they should be treated as symbols, and the verification will establish a property for all possible interpretations[2]. The

[2] Where there are mutliple predicate symbols, or con-

predicate must be a function into a fixed finite type (otherwise the problem would become intractably infinite), which in our case will be $\{true, false\}$. Again we must refer the reader to [8, 9, 10] for relevant theorems and definitions.

4 Data Independent Structural Induction

We take as our starting point the protocol model in [2]. Our aim is to show that any arbitrary tree network of N-branching nodes will offer correct behaviour, (specified by property P), on any interface to a receiver. In the case of the binary trees we lazily abstracted one of the two downstream channels of any intermediate node in the network, to achieve N-branching node trees we need to establish the structural induction for all-but-one of any number of channels lazily abstracted. So, we need to establish (3) and (4) below for a *Node* with any N *down* channels, where *AlphDown* is the set of all downstream communication *except* that labelled by a particular constant C (the name of the arbitrary channel we are leaving visible).

$$P \sqsubseteq Source \quad (3)$$
$$P \sqsubseteq \mathcal{L}_{Alphdown}(P[down \longleftrightarrow up]Node), \quad (4)$$

$\mathcal{L}_A(P)$ represents the lazy abstraction of the events in A from P. Superficially we cannot do this using a finite state model checker like FDR, since it requires an infinite number of checks, whose state spaces grow with N. Careful use of data independence, however, enables it.

Let the type T under consideration represent the names of the downstream channels of a node. So long as P and *Node* can be constructed in such a manner as to be data independent of T, (and also satisfy various other rules), it may be possible to establish a threshold on the size of T for (3) and (4). FDR can

stant symbols as well, it is possible and frequently desirable to check only those cases in which these are in a defined relationship. For example we might want to check the cases in which one predicate is contained in another and a constant belongs to their difference.

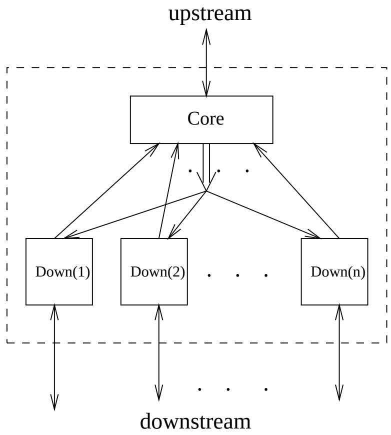

Figure 3: *(a) An n-branching node. Note that all events internal to the node, inside the dashed box, are hidden.*

then be used to check refinement relations corresponding to (3) and (4) for all sizes smaller than or equal to the threshold, thus establishing the structural induction for all possible sizes of T, and so for any arbitrary number of N-branching nodes.

The *Node* process is constructed from a process representing its core behaviour (*Core*) and the parallel combination of processes service a downstream channel (*Down*). Each of the *Down* processes must be data independent in the type T of all *Down* processes, as must *Core* (though the *Down* process as its own name as a constant symbols. Figure 3 shows a node with n downstream channels. Each of the *Down* processes is responsible for maintaining state on the reservations already in place on the node, and can issue responses to requests as appropriate. If a request is to be propagated upstream then the *Down* process sends the request to the *Core*, on a local request channel. The *Core* process then propagates this upstream. In order to perform effective merging of requests, and so achieve the desired traffic reduction properties, each of the *Down* processes needs to know of responses to requests made on the other downstream channels of the node. Figure 3 indicates how this is achieved; all replies from upstream are sent to the *Core* which in turn forwards these over the one reply channel to all of its *Down* processes (this is thus a *broadcast* communication, as indicated by the different way this channel is treated in this and later figures). We can hide all of the internal events of a node since no other processes in the network will ever need to synchronise on them. However, we are unable to use this construction of a node in (4) because indexed parallel composition over data independent types is prohibited in the theory of data independence we are using[3].

In order to perform our structural induction we need to use a process in (4) which is equivalent to a node with all but one of its downstream channels lazily abstracted, which is not itself constructed of the parallel composition of *Down* processes. Since the lazy abstraction of events on a channel allows any possible set of events to occur, including the empty set, it would seem intuitive that we could push the lazy abstraction higher up into the node and instead lazily abstract all the events which *Down* uses to send its requests to *Core*. If we could show that a node with all but one of its downstream channels lazily abstracted is equivalent to one which is not fully constructed, in that not all of the *Down* processes are present on the local request channels of *Core*, where all possible requests which could have come from those *Down* processes have been lazily abstracted, then we would have a data independent representation of a N-branching node which could be used as *Node* in (4). We need to show that the nodes (a) and (b) in Figure 4 are equivalent. We can't use data independence directly for the reasons discussed above, however, we can show this using a combination of induction and data independence:

Lemma 1 *Let the node (a) in Figure 4 be NODEa, and node (b) be NODEb, then* $NODEa \equiv NODEb$.

We prove this using a technique called data independent induction: see [3] for a more detailed explanation. We use data independence to prove a separate induction for each size of the type; discharging the base and step cases for all sizes simultaneously with a finite collection of refinement checks. We will use this type of induction to show that if we have a *Core* process with a set of n *Down* processes on a proper subset of its downstream request channels, then the process that results from abstracting the downstream events of this collection of *Down* processes is the same as results from lazily abstracting the same set of channels from *Core* directly, (i.e. without the

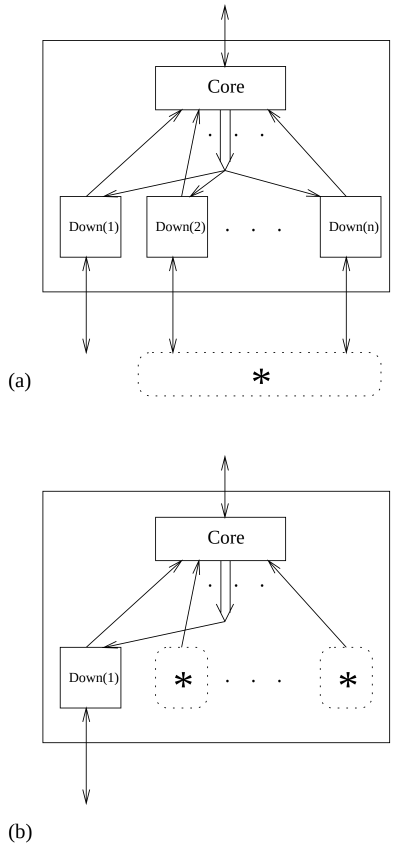

Figure 4: *(a) A node with n downstream channels, all but one lazily abstracted. (b) A node with n local request channels for Core all but one lazily abstracted away, the remaining one having a Down process attached to it. In both (a) and (b) the channel which Core uses to propagate replies to the Down processes is represented by a large single arrow.*

[3]This is largely because allowing such compositions would allow one to count the type T.

Down process). The result shown in Figure 4 is then implied by the case where there are *Down* processes on all but 1 of the local downstream request channels.

The base case ($n = 0$) of this induction is trivial as the processes which the induction claims are equal are syntactically the same. The step case is proved by showing that a single *Down* process, whose external interface is abstracted, can be removed and replaced by the abstraction of its request channel to *Core*. This is illustrated in Figure 5: ABS represents a set of request channels already abstracted.

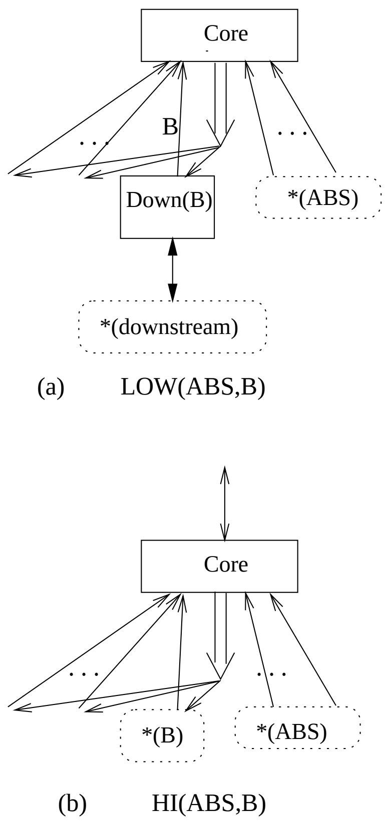

Figure 5: *We can replace the lazy abstraction of a downstream channel from Down by the removal of Down and the lazy abstraction of all events on its local request output channel to Core. Lazy abstraction of channels is denoted by dotted curved cornered rectangles.*

In order to prove equivalence of two processes in CSP it is sufficient to show that each refines the other in the *failures/divergences* model of CSP. We use data independence theorems to generate a threshold, this time on the size of T, where ABS is treated as a predicate mapping each member to true if it is in the set. We can then show that the following refinements hold for all sizes of T for all possible configurations of ABS, equal to the threshold and below it, then we have shown the two to be equivalent for all T:

$$HI(T, ABS, B) \sqsubseteq_{FD} LOW(T, ABS, B) \quad (5)$$
$$LOW(T, ABS, B) \sqsubseteq_{FD} HI(T, ABS, B) \quad (6)$$

As with all the examples in the paper, the threshold is small and is 3: the constant B outside ABS, one member of T inside ABS and one outside. This actually only leaves two checks for each of (5) and (6) which handle the cases of ABS being empty and of size one. This pair of lemmas easily justifies the induction, so proving Lemma 1, which in turn allows us to use the $NODEb$ in (4).

Thanks to the replacement of $NODEa$ by $NODEb$ (3) and (4) now become properties which can be proved by data independent reasoning (again with a small threshold). Thus a few refinement checks combined with our earlier results show that any branching width of node can validly be used in our RSVP structural induction. Hence, a much wider range of network topologies have been proven to have the desired property.

5 Conclusions

We have shown how data independence can be used to lift results obtained for limited-branching networks to ones with arbitrary branching. The most interesting part of the proof was caused by the parallel nature of the individual nodes and the consequent need to use a separate level of induction to eliminate most of this parallelism. Since, at least conceptually, many system components have simple message-handling processes resident on many channels, we imagine that this lower-level induction will have analogues in other applications. Indeed, there are reasonably close analogues with the second main example of [3] in which an arbitrary number of channels are multiplexed along a single pair. However, in many cases it will be possible to use data independence to deal directly (namely without the low-level induction) with arbitrary branching.

The way data independence is effectively used, in the low-level induction, to verify a separate induction for each size of type, is described in more detail in [3, 4]. The examples there are in some ways more ambitious since in most cases all nodes are supplied with the others' identities (as a type) and can use these in non-trivial ways.

Ongoing developments in the symbolic handling of data in refinement checks on FDR should be a great help in discharging the model checking obligations generated by our techniques. In effect this should completely automate the application of data independence. It would no longer be necessary to calculate thresholds (and the ad hoc arguments used to bring the threshold down), as all of the analysis would be done at "run-time" and automatically. Furthermore, there is every hope that the checks would complete much faster, as this method should reduce many equivalence-classes of essentially similar states down to a single one.

Acknowledgements

We would like thank Ranko Lazić for his work on data independence and Joy Reed for earlier collaborative work on this example which, indeed, she brought to our attention.

References

[1] S.J. Creese, *An Inductive Technique for FDR*, Master's thesis, Oxford University, 1997.

[2] S.J. Creese and J. Reed, *Verifying End-to-End Protocols Using Induction with CSP/FDR*, Paper accepted for presentation at FMPPTA'99, Puerto Rico, April 1999.

[3] S.J. Creese and A.W. Roscoe, *Verifying an infinite family of inductions simultaneously using data independence and FDR*, Paper submitted to FORTE/PSTV '99.

[4] S.J. Creese and A.W. Roscoe, Oxford University Computing Laboratory Technical Report, PRG-TR-1-99.

[5] *Failures-Divergence Refinement: FDR2 User Manual*, Formal Systems (Europe) Ltd, 1992-7.

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

[7] R.P. Kurshan and K. McMillan, *A Structural Induction Theorem for Processes*, Proceedings of 8th Symposium on Principles of Distributed Computing, Edmonton, 1989.

[8] R.S. Lazić, *A semantic study of data-independence with applications to the mechanical verification of concurrent systems*, Oxford University D.Phil thesis, 1999.

[9] R.S. Lazić and A.W. Roscoe, *Verifying Determinism of Concurrent Systems Which Use Unbounded Arrays*, Proceedings of INFINITY'98, Aalborg, Denmark, July 1998, extended version as Oxford University Computing Laboratory TR-2-98.

[10] R.S. Lazić and A.W. Roscoe, *Data independence with predicate symbols*, This volume.

[11] A.W. Roscoe, *The theory and practice of concurrency*, Prentice Hall, 1998.

[12] J.N. Reed, D. M. Jackson, B. Deianov and G. M. Reed, *Automated Formal Analysis of Networks: FDR Models of Arbitrary Topologies and Flow-Control Mechanisms*, Proceedings of ETAPS'98, Lisbon, Portugal, to appear in IEEE Transactions on Software Engineering, March 1998.

[13] P. Wolper, *Expressing interesting properties of programs in propositional temporal logic*, 184-193, Proceedings of the 13^{th} ACM POPL, 1986.

[14] P. Wolper and V. Lovinfosse, *Verifying Properties of Large Sets of Processes with Network Invariants (Extended Abstract)*, Proceedings of the International Workshop on Automatic Verification Methods for Finite State Machines, LNCS 407, 1989.